Telelogic Logiscope

TestChecker - Testing on a Target Machine

Version 6.5

Before using this information, be sure to read the general information under "Notices" section, on page **21**.

This edition applies to **VERSION *6.5*, *TELELOGIC LOGISCOPE*** (product number *5724V81*) and to all subsequent releases and modifications until otherwise indicated in new editions.

Table of Contents

About this manual

Audience

This reference manual in intended for **Telelogic® Logiscope™ TestChecker** users such as software developers, project managers or quality engineers who want to perform structural based testing and test coverage analysis and on a remote machine.

Overview

Section 1 explains the concepts involved in the instrumentation of an application.

Section 2 explains how code is instrumented.

Section 3 describes the file format used to store execution results.

Section 4 discusses the possible means to transfer the execution results to **Logiscope TestChecker**.

Section 5 presents general considerations to tailor the instrumented application in order to accommodate some common difficulties.

How to use this manual

This manual is a complement to the ***Telelogic Logiscope TestCkecker Getting Started***. Reading this document first is highly recommended.

Conventions

The following typographical conventions are used in this manual:

italics	names of textual elements (filename), notes, documentation titles.
`typewriter`	screen and file examples.

Contacting IBM Rational Software Support

Support and information for Telelogic products is currently being transitioned from the Telelogic Support site to the IBM Rational Software Support site. During this transition phase, your product support location depends on your customer history.

Product support

- If you are a heritage customer, meaning you were a Telelogic customer prior to November 1, 2008, please visit the Logiscope Support Web site.

  Telelogic customers will be redirected automatically to the IBM Rational Software Support site after the product information has been migrated.

- If you are a new Rational customer, meaning you did not have Telelogic-licensed products prior to November 1, 2008, please visit the IBM Rational Software Support site.

Before you contact 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.

Other information

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

Bibliography

[TCL94] JOHN K. OUSTERHOUT

Tcl and the Tk Toolkit - Addison-Wesley Professional Computing Series

1994 ISBN 0-201-63337-X

[TCL03] BRENT WELCH, KEN JONES, JEFFREY HOBBS

Practical Programming in Tcl and Tk (4th Edition) – Prentice Hall

2003 ISBN 0-130-38560-3

1. Overview

An instrumented application is produced by modify the source code of the application. Extraneous instructions are inserted at the beginning of function, at every Decision to Decision Point (DDP), that is at every test or loop, at every function call point and, if applicable, at every complex boolean expression (only for MC/DC).

The extra instructions are simple: they consist only in function calls to external functions. These functions are to be defined by the support libraries, which are responsible to format the events into a form that can be understood by **Logiscope TestChecker**.

The border between what is done by the instrumented code and what is done by the support library is hazy, especially for the C language, for which the instrumentation may be heavily customized.

1.1. Instrumented code

The inserted instructions allow to record events of interest during execution of the source code:

- Entering a function. The data associated with this event are the **Logiscope** name of the function and the date of the **Logiscope** analysis that produced the instrumented code.
- Executing a DDP(other than the first) of a function. The data associated with this event are the **Logiscope** name of the function and the number of the ddp.
- Calling a function. The data associated with this event are the **Logiscope** name of the calling function and the **Logiscope** name of the called function.
- Executing a complex boolean expression (for MC/DC). The data associated with this event are the **Logiscope** name of the function, the number of the condition in the function, the truth value of the condition and a vector of truth values of the inner conditions.

1.2. Support libraries

Every execution event detected by the instrumented code is directed to a function that must be defined by a support library. The library is responsible for determining how to communicate with **Logiscope TestChecker**, and to format the event data to fit the communication mean.

The support library must define one interface function for every event type.

All the delivered support libraries may be customized in order to accommodate specific needs. This is the simplest way to tailor the instrumented application to specific contexts and objectives.

The only constraints is to respect the interfaces used by the instrumented code.

2. Instrumentation

2.1. C

The instrumentation process for C uses the program **log_cc**, the startup syntax of which is described in the *Logiscope RuleChecker & QualityChecker C Reference Manual*:

```
log_cc -inst master.c
```

produces a *master.inst.c* and a *master.inst.h* without instrumentation for MC/DC.

```
log_cc -inst -cond master.c
```

produces a *master.inst.c* and a *master.inst.h* with instrumentation for MC/DC.

The application header files are not instrumented by **log_cc**: the *.inst.c* file contains the whole translation unit for the C file, thus the instrumentation of the header files used in *master.c* is included in *master.inst.c*.

The *master.inst.h* file is generated by a TCL file, which is evaluated by **log_cc**.

Let's have a look at the instrumentation produced for the following code:

```
void main(int argc, char* argv[])
{
  char inst;
  int result;
  /* if a parameter is present, the machine code is displayed */
  if (argc > 1)
    JACKPOT = 1;

  while (!instruction()); /* to display game rules*/
  player = TRUE;
  game_won = FALSE;

  format_output("Do you want to guess, or make up the code,",0);
  format_output(" g/m [default is g] -> ",0);

  if ((inst = getchar()) != '\n')
    while (getchar() != '\n');

  /*result used for FullMCDC test */
  result =(inst == 'm' || inst == 'M');
```

The resulting *master.inst.c* file is (with MC/DC instrumentation):

```
void main ( int argc , char * argv [ ] )
{

VLG_MCDC_DEF_0(VLG_CM_NEST_COMP8,VLG_SZ_VECT_COMP8);
VLG_CD1(main,8)
 {
char inst ;
int result ;

if ( argc > 1 )
{
VLG_CDX(main,8,2)
JACKPOT = 1 ;
}
else
 VLG_CDX(main,8,3)
{
int vlgbrk = 0;
while ( VLG_CM_0(0,main,8,88,4,1,1, ! VLG_EVAL_0(0, 0,
(VLG_CALL(main,8,1,9,1),instruction()))) )
{
VLG_CDX(main,8,4)
;
}
if (!vlgbrk) VLG_CDX(main,8,5)
}
player = 1 ;
game_won = 0 ;
(VLG_CALL(main,8,2,3,2),format_output ( "Do you want to guess, or
make up the code," , 0 ) );
(VLG_CALL(main,8,3,3,2),format_output ( " g/m [default is g] -> " ,
0 ) );
if ( ( inst = ( -- ( ( & _iob [ 0 ] ) ) -> _cnt >= 0 ? 0xff & *
( ( & _iob [ 0 ] ) ) -> _ptr ++ : (VLG_CALL(main,8,4,2,3),_filbuf
( ( & _iob [ 0 ] ) ) )) ) != '\n' )
{
VLG_CDX(main,8,6)
{
int vlgbrk = 0;
```

```
while ( ( -- ( ( & _iob [ 0 ] ) ) -> _cnt >= 0 ? 0xff & * ( ( & _iob
[ 0 ] ) ) -> _ptr ++ : (VLG_CALL(main,8,5,2,3),_filbuf ( ( & _iob
[ 0 ] ) ) )) != '\n' )
{
VLG_CDX(main,8,7)
;
}
if (!vlgbrk) VLG_CDX(main,8,8)
}
}
else
 VLG_CDX(main,8,9)
result = VLG_CM_0(0,main,8,99,0,2,2, (VLG_EVAL_0(0, 0, inst == 'm')
|| VLG_EVAL_0(0, 1, inst == 'M'))) ;
```

(Note that the macros are expanded in the instrumented C code).

When instrumenting the C code, Logiscope introduces macro calls in the code:

- `VLG_CD1`: entry of the function.
- `VLG_CDX`: another ddp.
- `VLG_CALL`: a function call.
- `VLG_MCDC_DEF_0`: initialization of the data structure needed to keep tracks of the MC/DC events.
- `VLG_CM_0`: a complex boolean expression.
- `VLG_EVAL_0`: an inner condition in a complex boolean condition.

These macros, and the support data structure are defined in the *master.inst.h*, which is produced by a TCL script.

Example: the simplest form of the `VLG_CDX` macro generating a *.trc* file would be:

```
#define VLG_CDX(name, functionIndex, ddpNumber) \
     fprintf(TRCFILE, “X\n%s\n%d\n”, #name, ddpNumber);
```

This is a bit faulty, since the name of the function is not a correct **Logiscope** name.

The standard definition is:

```
#define VLG_CDX(name,num,num_cdd) \
          vlg_c_cdx(vlg_arrayfunc[num], num_cdd, PARAM);
```

and the *master.inst.h* file defines the array `vlg_arrayfunc`:

```
static char *vlg_arrayfunc[] = {
  "**"
  ,"rest"
```

```
  ,"_filbuf"
  ,"format_output"
  ,"setcolors"
  ,"time"
  ,"srand"
  ,"rand"
  ,"master/main" /* functionIndex is 8 */
  ,"instruction"
  ,"player_plays"
  ,"machine_plays"
  ,"exit"
};
```

Support library

The C support library is located in *instr\src\vlgtchk.c*.

Support libraries adapted for multi tasked applications under **PSOS** and **VxWorks** real time OSes may be purchased separately. They are located in *instr\rtos\psos_12.zip* and *instr\rtos\vxworks_12.zip* respectively.

2.2. C++

The instrumentation process for C uses the program **lginst**, the startup syntax of which is described in the help file *bin\lginst.hlp*:

```
lginst -lang C++ Hangman.cpp
```

produces a *Hangman.inst.cpp* file. The C++ instrumenter does not support MC/DC.

Contrary to the C instrumenter, the C++ instrumenter instruments individually the header files:

```
lginst -lang C++ Hangman.cpp
```

produces a *Hangman.inst.h* file, analogous to the *Hangman.inst.cpp* file.

Let's have a look at the instrumentation produced for the following code:

```
BOOL CHangman::CheckLetter(char Letter)
{
     BOOL LetterAdded   = FALSE;
     int Size  =0;
     int Index =0;

     Size = m_CurrentWord.GetLength();     // Get length of current
word
     for(Index=0; Index<Size; Index++)     // Step through word to
check
```

```
     {
          if( m_CurrentWord[Index] == Letter )  // If we hit a
letter then
          {
               m_CurrentGuess.SetAt( (Index*2), Letter);  // Set
current guess to
               LetterAdded = TRUE;        // letter and change bool
          }
     }
     if( LetterAdded == FALSE )
          DecrementGuessRemain();         // If no letter added
decrement guesses remaining

     IncrementTotalGuesses();        // Increment total guesses so
far
     return( LetterAdded );                // Return TRUE/FALSE if
letter added or not
}
```

The resulting *Hangman.inst.cpp* file is:

```
BOOL CHangman::CheckLetter(char Letter)
{
/* function begin */
char *vlg_funcname = "CHangman::CheckLetter::37";
VLG_DDP1(vlg_funcname, "10/17/02-11:59:56");
{

     BOOL LetterAdded   = FALSE;
     int Size  =0;
     int Index =0;

     Size = m_CurrentWord.GetLength();     // Get length of current
word
     for(Index=0;VLG_COND(vlg_funcname, (int) ( Index<Size), 2, 3);
Index++)   // Step through word to check
     {
          if(VLG_COND(vlg_funcname, (int) ( m_CurrentWord[Index] ==
Letter ), 4, 5))      // If we hit a letter then
          {
               m_CurrentGuess.SetAt( (Index*2), Letter);  // Set
current guess to
```

```
                LetterAdded = TRUE;        // letter and change bool
            }
        }
        if(VLG_COND(vlg_funcname, (int) ( LetterAdded == FALSE ), 6, 
7))
            DecrementGuessRemain();  // If no letter added decrement 
guesses remaining


        IncrementTotalGuesses();   // Increment total guesses so far
        {
        /* return */
        return( LetterAdded );
        }                // Return TRUE/FALSE if letter added or not
}/* function end */
}
```

(Note that the macros are NOT expanded in the instrumented C++ code).

The instrumentation introduces macro calls in the C++ code:

- `VLG_DDP1(functionName, analysisDate)`: a function entry.
- `VLG_COND(functionName, expressionValue, ddpIfTrue, ddpIfFalse)`: another ddp of the function.

Support library

The C++ support library is located in *instr\src\vlgtchk.c*.
These macros are defined in the *instr\include\log_inst.h* file, that may be customized to accommodate different needs.

Support libraries adapted for multi tasked applications under **PSOS** and **VxWorks** real time OSes may be purchased separately. They are located in *instr\rtos\psos_12.zip* and *instr\rtos\vxworks_12.zip* respectively. The current version of these library needs minor tweaking to be used with C++.

2.3. Java

The instrumentation process for Java uses the program **lginst**, the startup syntax of which is described in the help file *bin\lginst.hlp*:

```
lginst -lang Java Hangman.java
```

produces a *Hangman.inst.java* file. The instrumentation does not support MC/DC.

Let's have a look at the instrumentation produced for the following code:

```
    public void init() {
        int i;
```

```
        // load in dance animation
      danceMusic = getAudioClip(getCodeBase(), "dance.au");
      danceImages = new Image[40];

      for (i = 1; i < 8; i++) {
          Image im = getImage(getCodeBase(), "T" + i + ".gif");

          if (im == null) {
           break;
          }
          danceImages[danceImagesLen++] = im;
         }

         // load in hangman image sequnce
        hangImages = new Image[maxTries];
        for (i=0; i<maxTries; i++) {
         hangImages[i] = getImage(getCodeBase(), "h"+(i+1)+".gif");
        }

        // initialize the word buffers.
        wrongLettersCount = 0;
        wrongLetters = new char[maxTries];

        secretWordLen = 0;
        secretWord = new char[maxWordLen];

        word = new char[maxWordLen];

        wordFont = new java.awt.Font("Courier", Font.BOLD, 24);
      wordFontMetrics = getFontMetrics(wordFont);

      resize((maxWordLen+1) * wordFontMetrics.charWidth('M') +
maxWordLen * 3,
            hangImagesHeight * 2 + wordFontMetrics.getHeight());
     }
```

The resulting *Hangman.inst.java* file is:

```
     public void init() {
```

```
    /* function begin */
    String vlg_funcname = "Hangman::init::120";
    VlgInstrument.ddp1(vlg_funcname, "10/17/02-11:59:56");
    {

        int i;

        // load in dance animation
     danceMusic = getAudioClip(getCodeBase(), "dance.au");
     danceImages = new Image[40];

     for (i = 1;VlgInstrument.cond(vlg_funcname, ( i < 8), 2, 3); i+
+) {
         Image im = getImage(getCodeBase(), "T" + i + ".gif");

         if (VlgInstrument.cond(vlg_funcname, (im == null), 4, 5)) {
          break;
         }
         danceImages[danceImagesLen++] = im;
        }

        // load in hangman image sequnce
        hangImages = new Image[maxTries];
        for (i=0;VlgInstrument.cond(vlg_funcname, ( i<maxTries), 6,
7); i++) {
         hangImages[i] = getImage(getCodeBase(), "h"+(i+1)+".gif");
        }

        // initialize the word buffers.
        wrongLettersCount = 0;
        wrongLetters = new char[maxTries];

        secretWordLen = 0;
        secretWord = new char[maxWordLen];

        word = new char[maxWordLen];

        wordFont = new java.awt.Font("Courier", Font.BOLD, 24);
```

```
     wordFontMetrics = getFontMetrics(wordFont);


     resize((maxWordLen+1) * wordFontMetrics.charWidth('M') +
maxWordLen * 3,
            hangImagesHeight * 2 + wordFontMetrics.getHeight());
    }/* function end */
    }
```

The instrumentation introduces function calls in the code:

- `VlgInstrument.ddp1(String funcName, String anlysisDate)`: a function entry.
- `VlgInstrument.cond(String funcName, boolean conditionValue, int ddpIfTrue, int ddpIfFalse)`: another ddp of the function.

The Java support library is located in *instr\jv\VlgInstrument.java* and *instr\jv\VlgTrace.java*.

2.4. Ada

The Ada instrumentation is described in the *Telelogic Logiscope TestChecker Getting Started* manual.

The Ada support libraries are located in the *data\audit_ada\instrument.ada* (Ada95) and *data\audit_ada\instrument83.ada* (Ada83).

It is possible to customize these code files to accommodate different needs.

3. File formats

Two file formats may be loaded in **Logiscope TestChecker** to describe test results for a project. The first, and historical, one is the *.dyn* format; this is the format in which **Logiscope TestChecker** saves the tests. The second, much more verbose, but of great importance for our purpose since it is easier to fiddle with is the *.trc* format.

3.1. .dyn files

This file format is compact, but is difficult to modify and produce. This is the default file format output by the support libraries when the instrumented programs are not launched from **Logiscope TestChecker**.

Let's look at a *.dyn* file produced for the *Hangman* sample, interspersed with explanations in italics:

```
<archive VD2.0>
*NA*
... This is "current application" in french.
... Do not change this.
Application_courante
*CV*
... List of test suites in this file (there always be
... only one test suite in the file).
CURRENT_SUITE
*CM*
CURRENT_SUITE
... List of tests in the test suite named CURRENT_SUITE.
TEST_1 09/13/00-13:50:12
TEST_2 09/13/00-13:51:02
... Coverage results for test TEST_1.
*MO*
TEST_1 09/13/00-13:50:12
... Catalog of components (functions) executed during
... TEST_1.
.NM.
1 CHangman32App::CHangman32App::26 09/13/00-11:42:46
2 CHangman32App::InitInstance::41 09/13/00-11:42:46
3 CPictureButton::CPictureButton::18 09/13/00-11:42:46
... And so on for every component of the test catalog.
.CC.
... Component Changman32App::CHangman32App::26 has
... executed its first ddp.
```

```
1 1
... Component CHangman32App::InitInstance::41 has
... executed its ddp numbered 1, 3 and 4, but not 3.
2 1 0 1 1
3 1
... And so on for every component of the test catalog.
... Then the content is repeated for every test in
... the test suite.
```

3.2. .trc files

This format is more verbose than the *.dyn* format, but is easier to manipulate and produce. It consists of one record for each occurrence of one of these events:

- Entering a function.
- Executing a ddp (other than the first) of a function.
- Calling a function.
- Executing a complex boolean expression (for MC/DC only).

A *.trc* file is produced by the support library *vlgtchk.c* if the environment variable `VLGTYP` is set to `TRACKS`.

This is also the format that is used natively by **Logiscope TestChecker** to retrieve the execution events from an instrumented application that it launches.

Let's have a look at this file format for the *Mstrmind* sample, heavily edited and interspersed with explanations in italics:

```
... Entering function master/main. The function has been
... analyzed on January the 29th, 1999.
1
master/main
01/29/99-12:05:36
... Executing ddp number 3 of the function master/main.
X
master/main
3
... Calling function instruction from function master/main.
P
master/main
instruction
... Entering function instruction.
```

```
1
instruction
01/27/99-15:51:08
... Complex conditions executed (inst == 'm' || inst == 'M')
... this is complex condition number 2 in the function
... master/main. The result was true (1), and the first
... condition was true, and the second not evaluated (1-).
C
master/main
2
1
1-
```

4. Communicating with Logiscope TestChecker

4.1. Using TcGatWay

TcGatWay is a specialized application, designed to appear as an instrumented application to **Logiscope TestChecker**. This tool merely pass back all information received on a TCP socket or a serial link to **Logiscope TestChecker**.

TcGatWay startup syntax is different for serial links on **Microsoft Windows** and **UNIX**.

On **Microsoft Windows**:

```
TcGatWay  [-serial <port> [-mode <mode>] ] |
          [-tcp [-reuse]] |
          [[-tcp] [-reuse] -host <host> [-port <port>]] |
          [[-tcp] [-reuse] [-host <host>] -port <port>]
          -prefix <string>
```

- *-serial* designates the serial port (COM1, COM2, etc).
- *-mode* designates the mode of operation of the serial port in usual **Microsoft Windows** syntax.

On **UNIX**:

```
TcGatWay  [-serial -in <fd>] |
          [-tcp -in <fd>] |
          [-tcp [-reuse]] |
          [[-tcp] [-reuse] -host <host> [-port <port>]] |
          [[-tcp] [-reuse] [-host <host>] -port <port>]
          -prefix <string>
```

- *-serial* means that the file file descriptor designated by the *-in* option is to be used as the serial input. The serial port must have been configured beforehand with the command `stty`.
- *-tcp* means that a TCP socket is to be used. The default host is `localhost`, the default port is `6309`. On **UNIX** systems, an already opened TCP socket may be used by specifying its file descriptor with the *-in* option. The instrumented application is supposed to connect to the the port used by **TcGatWay**.

To use **TcGatWay** with **Logiscope TestCheker**, a customized support library must be developed and linked with the instrumented application. The library must connect to the TCP port of the `hostname`, or the serial link, where **TcGatWay** has been launched from **Logiscope TestCheker**, and then send the execution events in the *.trc* format on this communication link.

TcGatWay is useful in demo conditions, or when setting up things. Its interactive nature does not turn it into the solution of choice for production environments. In these cases, it is easier to work with files.

4.2. Using files

As outlined above, the easiest format to work with is the *.trc* format. If the target has a file system, it is sufficient to store the execution results in a file, and transfer the file to the host at the end of the test.

The file may then be loaded in **Logiscope TestChecker** to analyze the coverage of the test.

Any communication mean between the target machine and the host that can transfer text streams is adequate for this task.

5. Special cases

5.1. Multi tasking OSes and/or multi processor machines

The *.trc* format allows the different event records to be interspersed freely, but the records must not be broken.

A multi tasked application must then take special caution to not break the atomicity of the event records. Several solutions are available:

- Synchronization; but this may disturb the expected time behavior of the application, and this may forbid to instrument the interrupt service routines.
- One file (or stream) of event reports per thread of execution; this may complicate the sending of the event reports if real time streams are used instead of pipes. This may also forbid to instrument the code of the file system driver.

No single solution is a best fit for all situations. It is often necessary to examine closely the inner workings of the application and the coverage measurement goals to find the appropriate solution for a specific situation.

But, whatever the solution needed, the great flexibility of the articulation between the instrumented code and the support library allows to implement it.

5.2. Tight environments

The instrumented application has more code than the original application. This may lead to troubles if the target environment does not have enough program memory to accommodate the instrumented application.

To reduce the program space needed by the instrumented application, it is possible to reduce the number of event kinds sent by the application: in C, the call graph coverage is often not needed, thus it suffice to `#define` out the `VLG_CALL` macro.

If this is not sufficient, it will be necessary to design a special instrumentation and library to drastically reduce the memory requirements of the instrumented program. This involves the design of a new format to store and transfer the execution events; then on the host, *.trc* file must be created from this new format.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send written license inquiries to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send written inquiries to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions. Therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software
IBM Corporation
1 Rogers Street
Cambridge, Massachusetts 02142
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Copyright license

Trademarks