RB Code Reports

Software Metrics for REALbasic

User Guide

Version 2.0.4

True North Software

Table of Contents

Introduction

RB Code Reports provides software metrics for your REALbasic application. It gives you a way to objectively evaluate the maintainability of your project and improve its performance. It analyzes your project and produces a set of reports including the following.

- Statistics
- Optimizations
- Cyclomatic Complexities
- Warnings
- Signatures
- Spell Checking
- Custom User Defined Reports

You can use these reports to clean up and optimize your code and show you what methods need to be refactored.

System Requirements

- XP or later.
- Mac OS X 10.4 or later.

RB Code Reports will probably work on earlier systems, but is not officially supported.

Installation

Mac

Uncompress zip file downloaded from the www.TrueNorthSoftware.com web site and then drag the application folder to your hard disk.

PC

Uncompress the file downloaded from the www.TrueNorthSoftware.com web site and run the installer and follow the on screen instructions.

Unregistered Copy Restrictions

Unregistered copies of RB Code Reports are fully functional except that certain reports will not be displayed and numerical information will be replaced with Xs. See the sample reports for examples of what the reports look like.

Customer Support

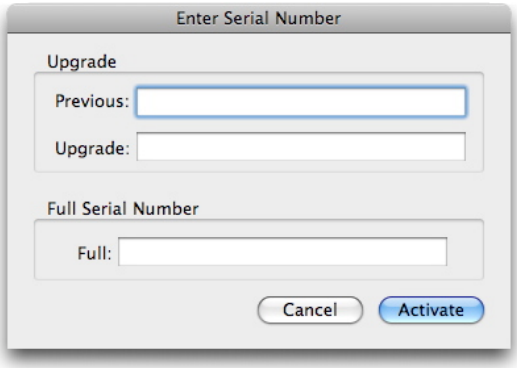
If you need technical support with RB Code Reports, send email to our support email.

support@truenorthsoftware.com

Buying a License

After you physically install RB Code Reports on your machine, you will need to acquire a license in order to use the full functionality. To purchase a single user license open the About RB Code Reports window as shown below.

Click on the Buy button and follow the on screen instructions to obtain a serial number. Once a serial number has been purchased, you will need to activate it. To activate the serial number, click on the Activate button. Enter the serial number into the dialog shown below.



Usage

How to Use

To use RB Code Reports, you need to first export your project to the REALbasic XML project format. Select the “Save As...” menu item in REALbasic and choose the XML format option.

Warning: The REALbasic IDE behavior has recently change for the “Save As....” menu item. Previously doing a save as to the XML format would act like an export and retain the current file as the target file. The IDE now retargets the to the XML file and any changes you make after the save as operation will now go to the XML file instead of the original file. So you need to close the project in the REALbasic IDE and reopen it to retarget the original file.

Once the project has been saved in the XML format, then either drag and drop the XML file on the RB Code Reports application or use the “Open...” menu item to select the XML file. RB Code Reports will then process the XML file and create the reports and then show the statistics report in a new window. At any time you can select the desired report you want from the “Reports” menu. If you want to show all the reports in the window, then select the “Show All” menu item. You may also save the contents of a window to an RTF file. Select the “Save” menu item to save the report to a file.

You may also open RTF files that you create with RB Code Reports with the “Open...” menu item.

Reports

Statistics Report

The statistics report shows the following stats for your code.

- LOC - Total lines of code.
- NCSL - Non-commentary source lines.
- Code Density - Measure of how densely packed the source lines are.
- Comment Density - The ratio of comment lines to code lines.
- Number of windows.
- Number of classes.
- Number of modules.
- Number of interfaces.

The following items are counted as a line of code.

- Blank lines.
- Comment lines.
- Executable lines.
- Signatures.
- Constants.
- Properties.

In general, items you can change are counted as a line of code.

LOC is the measure of all lines of code in a project which includes blank lines, comment lines, and code lines.

NCSL is the measure of the non-commentary source lines in a project and is the true measure of how many code lines are in project because it only includes executable code lines (it skips blank and comment lines).

Code Density is the measure of how tightly packed is the code. In other words, did the programmer use blank lines to make the code more readable? In the following code example no blank lines were used so the code density would be 100%.

```
Dim p as Integer
if y >= 0 and y < mHeight then
  p = mFillStackPtr
  mFillStack(p,0) = y
  mFillStack(p,1) = xleft
  mFillStack(p,2) = xright
  mFillStack(p,3) = dy
  mFillStackPtr = mFillStackPtr+1
end if
```

A good target goal for code density is between 50% to 60%. This means the code has blank lines separating logical chunks of code. If you are inspecting code, you should flag methods that exceed 80% code density.

Comment density is the measure of how many comments a method has. Theoretically you would want to comment every line of code which would lead to a ratio of 1 comment per NCSL.

In practice it will be some what less than that. You will have to decide what is the cut off threshold for comment density. Generally speaking anything less than 0.30 is suspect.

The first entry in the report is the “Project” entry. This shows the project wide counts of LOC NCSL, windows, classes, interfaces, and modules within the project. Then each class, window, and module follows the project entry. Any methods will also be broken down to show the individual LOC, NCSL, Code Density, and Comment Density.

You can specify in the preferences window if you want to show method level statistics. Shown below is a partial sample of a statistic report.

Statistics Report

Project:
Interfaces = 2
Modules = 30
Classes = 85
Windows = 11
LOC = 68868
NCSL = 33095

Window: AboutWindow
LOC = 216, NCSL = 106

Control: *CloseItButton.Action()*
LOC = 5, NCSL = 2, Code Density = 50.0%, Comment Density = 1.00

Control: *BuyButton.Action()*
LOC = 19, NCSL = 8, Code Density = 61.1%, Comment Density = 0.57

Control: *ActivateButton.Action()*
LOC = 33, NCSL = 15, Code Density = 62.5%, Comment Density = 0.43

Control: *VersionNumberLabel.Open()*
LOC = 13, NCSL = 8, Code Density = 66.7%, Comment Density = 0.14

Optimizations Report

The optimization report suggest changes you can make to your code which will speed up your code.

Currently there is only one check.

- If a method contains a looping construct such as “for-next”, “while-wend”, or “do-loop”, and there is no "#pragma DisableBackgroundTasks" in the method, then it will be shown in this report with the message, “Adding ‘#pragma DisableBackgroundTasks’ may speed up the method.” Adding this pragma to the top of the method disables thread context switching. Thread context switching is very expensive and can greatly slow down your code.

Shown below is a partial sample of an optimization report.

Optimization Report

Module: **CyclomaticComplexity**

Method: *getSourceComplexity(lines() as string) As integer*

• Adding "#pragma DisableBackgroundTasks" may speed up the method.

Method: *cyclomaticComplexityReport(sc as SourceCode) As ReportData*

• Adding "#pragma DisableBackgroundTasks" may speed up the method.

Method: *createModuleReport(sc as SourceContainer) As ReportData*

• Adding "#pragma DisableBackgroundTasks" may speed up the method.

Class: **FTRBScriptIndentThread**

Event: *Run()*

• Adding "#pragma DisableBackgroundTasks" may speed up the method.

Class: **FTSpellCheckThread**

Event: *Run()*

• Adding "#pragma DisableBackgroundTasks" may speed up the method.

Cyclomatic Complexity Report

The McCabe cyclomatic complexity metric is the measure of number of linear segments of code within a method. Linear segments are defined by the number of decision points within the method. Decision points are defined as branching points such as "if...then...else" statements. The more branching and looping decision points, the more complicated the code. The cyclomatic complexity metric can be used to estimate the number of tests required to test the functionality of a method or to judge the psychological complexity of the code.

Risk Evaluation

The following ranges are generally accepted as the threshold levels of complexity for a method.

- 1-10 simple, without much risk.
- 11-20 more complex, moderate risk.
- 21-50 complex, high risk.
- Greater than 50, untestable (very high risk).

If you have a high risk method, then you should refactor the method since highly complex methods are generally unmaintainable.

Methods within a class are sorted by complexity in descending order. When the application is in demo mode, the results are randomly shuffled.

Shown below is a partial sample of a cyclomatic complexity report.

Cyclomatic Complexity Report

Module: **CyclomaticComplexity**

015 : isDecisionPoint(token as string) As boolean

Class: **FormattedText**

025 : handleOptionDeleteWord(forwards as boolean)
024 : handleDeleteKeys(key as string) As boolean
023 : callKeyDown(key as string) As boolean
023 : filterKeys(key as string) As boolean
020 : callDropObject(obj as DragItem, action as integer)
019 : findSelectionRectangle(ByRef x as integer, ByRef y as integer, ByRef width as integer, ByRef height as integer)
018 : callMouseDrag(x as integer, y as integer)
015 : handleDoubleClick(x as integer, y as integer)
014 : callMouseDown(x as integer, y as integer) As boolean
014 : adjustScrollbars()
014 : print(g as graphics, printer as PrinterSetup, firstPage as integer = -1, lastPage as integer = -1)
013 : handleUpArrowKey()
013 : handleDownArrowKey()
012 : scrollToCaret()
011 : setMousePointer(x as integer, y as integer)

Warnings Report

The Warnings report shows possible maintenance items within your code. You can turn on/off these warnings in the preferences window.

The following items are checked with the Warnings report.

- Methods that exceed a specified NCSL limit. Methods that exceed certain NCSL are more likely to be unmaintainable and should be refactored.
- DIM statements that have been buried within the code. If it is your policy to place DIM statements at the top of method so that they are easy to find, then this warning shows you which methods violate that policy.

Shown below is a partial sample of a warnings report.

Warnings Report

Class: FormattedText

Method: *update(compose as boolean, force as boolean, selectionOnly as integer = 0)*

• Misplaced DIM statements were found.

Method: *findSelectionRectangle(ByRef x as integer, ByRef y as integer, ByRef width as integer, ByRef height as integer)*

• Method exceeds line count threshold (128 lines).

Method: *callKeyDown(key as string) As boolean*

• Misplaced DIM statements were found.

Module: FTCPluginMethods

Method: *pictureToPNGString(p as picture) As string*

• Misplaced DIM statements were found.

Method: *binaryToHex(binaryData as string) As string*

• Misplaced DIM statements were found.

Method: *pictureToJPEGString(p as picture) As string*

• Misplaced DIM statements were found.

Module: FTUtilities

Method: *getDoubleClickTime() As integer*

• Misplaced DIM statements were found.

Class: RTFContext

Method: *dump() As string*

• Misplaced DIM statements were found.

Signatures Report

The signatures report is used for extracting constant, property, event definition, event, and method signatures from a REALbasic project. You would use this report if you are creating documentation for your software.

Shown below is a partial sample of a signatures report.

Signature Report

Window: AboutWindow

Events:

Open()

Methods:

setRegistrationStatus()

Class: App

Constants:

kEditClear
kFileQuit
kFileQuitShortcut
SERIAL_NUMBER_PORT
SERIAL_NUMBER_SOCKET

Properties:

customReports() As CustomReportScript
demoMode As boolean = true
prefs As PreferenceDocument
sco As SpellCheckOptions
serialNumberSocket As LocalSerialNumberSocket
userDictionary As FolderItem
uww As UserWordsWindow
win32AppMutex As WindowsMutexMBS

Events:

Close()
EnableMenuItems()
Open()
OpenDocument(item As FolderItem)
UnhandledException(error As RuntimeException) As Boolean

Menu Handlers:

FileCloseAll() As Boolean
FileOpen() As Boolean
FilePreferences() As Boolean

HelpAboutRBCodeReports() As Boolean
HelpShowConsole() As Boolean
HelpUserGuide() As Boolean
SpecialNewUserDictionary() As Boolean
SpecialReloadCustomScripts() As Boolean
SpecialShowUserWords() As Boolean

Methods:

aboutAction()
checkLocalLicenses()
closeUserDictionary()
Constructor()
getCustomReport(index as integer) As CustomReportScript
getCustomReportCount() As integer
getFormula() As string
getPreferenceString() As string
getPrefs() As PreferenceDocument
getSpellCheckOptions() As SpellCheckOptions
getUserDictionaryPath() As string
hasUserDictionary() As boolean
isDemoMode() As boolean
isWin32ApplicationRunning(mutexName as string)
loadCustomReports()
loadCyclomaticPreferences()
loadGeneralPreferences()
loadPreferences()
loadSpellCheckPreferences()
loadStatisticPreferences()
loadUserDictionary(fi as FolderItem)
loadWarningPreferences()
openReportAction(fi as FolderItem)
openUserDictionary()
saveUserDictionary(fi as FolderItem)
setDemoMode(state as boolean)
stimulate()
updateUserDictionary()
updateWindows()

Spelling Report

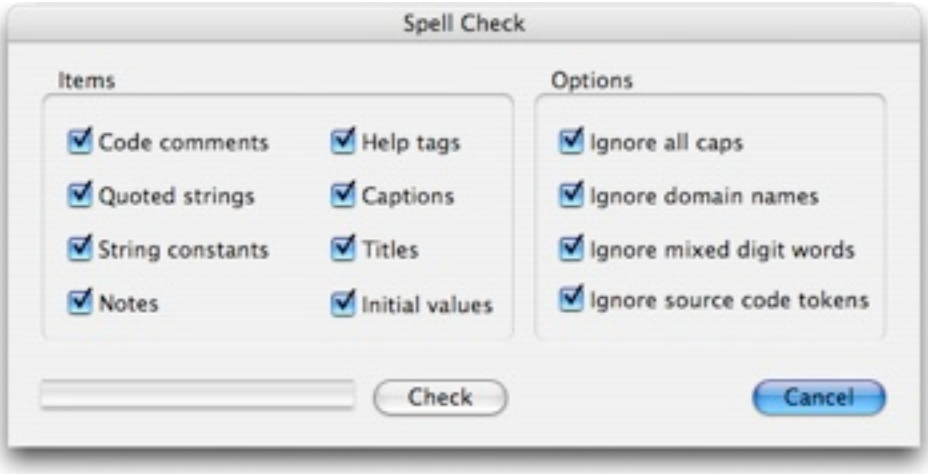
The spell check report reports spelling errors within your project. It use a 100,000 word american english dictionary to check the words. It will check the following items.

- **Code comments** - Check comments within all methods.
- **Quoted strings** - Check quoted strings within the source code.
- **String constants** - Check string constants.
- **Notes** - Check the note items within modules, classes, and windows.
- **Help tags** - Check the help tags for controls.
- **Captions** - Check the captions for controls.
- **Titles** - Check window titles.
- **Initial values** - Check the initial value fields for controls.

In addition, you can set the following options for the spell checker.

- **Ignore all caps** - Ignore words that use all capital letters.
- **Ignore domain names** - Ignore domain names such as a web address.
- **Ignore mixed digit words** - Ignore words that contain both letters and digits.
- **Ignore source code tokens** - Ignore words harvested from the source code.

You can also manually run the spell check by choosing the “Check Spelling...” menu item. You will be presented with the following dialog in which you can choose the options. Click the “Check” button to initiate the spell check.



You set the default preferences for this dialog in the preferences window.

You can create a user dictionary to hold words that are not in the main dictionary. In order to use a user specified dictionary, you must first create one with the “New User Dictionary” menu item. Once the dictionary is created, you must select it in the “User Dictionary” preference in the preference window. Select the “Show User Words” menu item to add or delete words from the dictionary.

Shown below is a partial sample of a spell check report.

Spell Check Report

Class: App
Method: *Constructor()*
Comment - Line 11: (plugin) Register the MBS plugin.

Method: *isWin32ApplicationRunning(mutexName as string)*
Comment - Line 13: (mutex) Create the mutex.
Comment - Line 16: (mutex, apps) Try to create the mutex to see if any other apps are running

Constant: *kFileQuitShortcut*
Constant - Line 2: (Cmd) Cmd+Q

Constant: *kFileQuit*
Constant - Line 2: (xit) E&xit

Module: ConsoleUtilities
Method: *getConsoleTimestamp() As boolean*
Comment - Line 12: (timestamps) Return the current state of using timestamps in the log.

Custom Reports

Custom reports are user defined reports which are generated from RBScripts. These scripts are located in a folder called “Custom” next to the RB Code Reports application and have a suffix of “crs” in order to be recognized by the program. You can put as many report scripts in this folder as you want.

To create an RBScript to be used for creating a custom report, create a file in the “Custom” folder with a “.crs” extension and use your favorite editor to edit the script. When you start up the application, it will automatically scan the Custom folder for “.crs” files and load them in. When you open a xml project file, it will run all the custom scripts against the project data. To view the output of the custom reports, choose “Reports->Custom” from the menu. You may reload all the custom scripts by choosing “Special->Reload Custom Scripts” menu item. After you reload the scripts, you will need to choose “Reports->Run All Reports” to regenerate the custom reports.

The title of the report in the display will be the name of the script file without the “.crs” suffix. A “sample.crs” script is included to show you how to walk though all the data structures within a project.

RBScript context methods:

The following methods are provided for accessing and manipulating project data.

Project Data Access Methods:

There are three basic data structure to deal with. At the top level is the “container” which represents a window, class, or module. The second is the “control” which are controls in a window. The third is everything else contained within a window, class, or module which includes constants, properties, methods, menu handlers, and notes. Note, all the indexes specified for parameters and results are zero based unless specifically stated otherwise.

Container Methods:

getContainerCount() As integer

Returns the number of containers in the project data. This is a one based result.

getContainerEventDefinition(containerIndex as integer, index as integer) As string

Returns the specified event definition.

getContainerEventDefinitionCount(containerIndex as integer) As integer

Returns the number of event definitions in the specified container. This is a one based result.

getContainerName(containerIndex as integer) As string

Return the name of the container.

getContainerSuperClass(containerIndex as integer) As string

If the container is a window or class, return the super class of that container.

getContainerTitle(containerIndex as integer) As string

Return the tile of the container. Note, only makes sence for windows.

getContainerType(containerIndex as integer) As string

Return the container type so that you will know how to handle the data within the container.

Return values: “Window” or “Module”

Control Methods:

getControlCaption(containerIndex as integer, index as integer) As string

Get the caption of the control.

getControlCount(containerIndex as integer) As integer

Returns the number of controls in the specified container. This is a one based result.

getControlHelpTag(containerIndex as integer, index as integer) As string

Get the help tag of the control.

getControlInitialValue(containerIndex as integer, index as integer) As string

Get the control’s initial value.

getControlMethodCount(containerIndex as integer, controlIndex as integer) As integer

Returns the number of methods in the specified container. This is a one based result.

getControlMethodSignature(containerIndex as integer, controlIndex as integer, methodIndex as integer) As string

Get the method’s signature.

getControlMethodSource(containerIndex as integer, controlIndex as integer, methodIndex as integer) As string

Get the method’s signature including the signature.

getControlName(containerIndex as integer, index as integer) As string

Get the name of the control.

Item Methods:

getItemConstant(containerIndex as integer, index as integer) As string

Get the constant name.

getItemConstantDefinition(containerIndex as integer, itemIndex as integer, index as integer) As string

Get the constant definition. Remember, a constant may have more than one definition.

getItemConstantDefinitionCount(containerIndex as integer, index as integer) As integer

Returns the number of constant definitions for the specified constant. This is a one based result.

getItemCount(containerIndex as integer) As integer

Returns the number of items in the specified container. This is a one based result.

getItemProperty(containerIndex as integer, index as integer) As string

Get the specified property.

getItemSignature(containerIndex as integer, index as integer) As string

Get the method's signature.

getItemSource(containerIndex as integer, index as integer) As string

Get the method's signature including the signature.

getItemType(containerIndex as integer, index as integer) As integer

Return the item type so that you will know how to handle the data within the within the item.

Return values:

1 = method
2 = constant
3 = property
4 = event
5 = menu handler
6 = note

isContainerClass(containerIndex as integer) As boolean

Is the container a class?

isContainerInterface(containerIndex as integer) As boolean

Is the container a class interface?

Regular Expressions:

The regular expression methods are extensions of the built in RegEx and RegExMatch classes. Underneath the hood there is one RegEx and one RegExMatch object created for you to use

and these methods access that single objects. So you must be careful when calling these methods in your script to make your usage of them is atomic. This means you should not call these methods and then call methods that use the RegEx methods and then call the RegExMatch methods after returning from the method called. You will get unpredictable results!

Properties:

RECaseSensitive As boolean
REDotMatchAll As boolean
REGreedy As boolean
RELineEndType As integer
REMatchEmpty As boolean
REReplaceAllMatches As boolean
REReplacementPattern As string
RESearchPattern As string
RESearchStartPosition As integer
REStringBeginIsLineBegin As boolean
REStringEndIsLineEnd As boolean
RETreatTargetAsOneLine As boolean
RESubExpressionCount As integer

Methods:

REReplace() As string
REReplace(targetString as string) As string
REReplace(targetString as string, searchStartPosition as integer) As string

RESearch()
RESearch(targetString as string)
RESearch(targetString as string, searchStartPosition as integer)

REMatchReplace() As string
REMatchReplace(replacementPattern as string) As string
RESubExpressionStartB(matchNumber as integer) As integer
RESubExpressionString(matchNumber as integer) As string

Utility Methods:

join(sourceArray() as string, delimiter as string) As string
split(source as string, delimiter as string) As string()

The equivalent if the join and split methods in the RB framework.

splitLines(s as string) As string()

Split the source code into an array of string divided by line endings.

stripComments() As string()

Strip out all comments (‘, //, REM) from the lines and leave only the code.

stripCode(lines() as string) As string()

Strip out all the code and leave only the comments.

Style Text and Paragraphs:

Your report data may be displayed as styled text. The following properties and methods allow you to add styled text to the report.

Paragraph Properties:

alignment As integer

Paragraph alignment: left = 0 or 1, center = 2, right = 3

paragraphBackgroundColor As Color = &cFFFFFF

The color displayed behind the paragraph.

Styled Text Properties:

bold As boolean
font As string = "Arial"
fontSize As integer = 12
italic As boolean
textBackgroundColor As Color = &cFFFFFF
textColor As Color = &c000000
underline As boolean

Methods:

newParagraph(indentLevel as integer, pageBreak as boolean)

Create new paragraph. indentLevel is the left margin of the paragraph in increments of 0.25 inches. pageBreak indicates if this paragraph should begin on a new page. The paragraph will be customized with the current values of the global properties of alignment and paragraphBackgroundColor.

newStyleRun(text as string)

Add text to the current paragraph. The text will be styled with the current setting of the global properties bold, font, fontSize, italic, textBackgroundColor, textColor, and underline.

prt(line as string)

Send a line of text to the report output.

prt()

Send a blank line to the report output.

reset()

Reset the style run attributes to their default values.

Debug Methods:

console(line as string)

Send a message to the program console.

showConsole()

Show the program console.

Preferences

The following preferences are available within RB Code Reports.

Statistics

- **Summary only** - Exclude method level statistics.

Cyclomatic Complexity

- **Threshold** - When running the Cyclomatic Complexity report, only report items that are equal or greater than this threshold.

Warnings

- **Large methods** - Check for methods that exceed the specified limit for NCSL.
- **Misplaced DIM statements** - Check for DIM statements that have been place within the body of the code.

Spell Checking

- **Code comments** - Check comments within all methods.
- **Quoted strings** - Check quoted strings within the source code.
- **String constants** - Check string constants.
- **Notes** - Check the note items within modules, classes, and windows.
- **Help tags** - Check the help tags for controls.
- **Captions** - Check the captions for controls.
- **Titles** - Check window titles.
- **Initial values** - Check the initial value fields for controls.
- **Ignore all caps** - Ignore words that use all capital letters.
- **Ignore domain names** - Ignore domain names such as a web address.
- **Ignore mixed digit words** - Ignore words that contain both letters and digits.
- **Ignore source code tokens** - Ignore words harvested from the source code.
- **User Dictionary** - Path to the user dictionary. You create a user dictionary with the “New User Dictionary” menu item.

Legal

Copyright Notices

End User License Agreement

CAREFULLY READ THE FOLLOWING LICENSE AGREEMENT. YOU ACCEPT AND AGREE TO BE BOUND BY THIS LICENSE AGREEMENT BY CLICKING THE ICON LABELED "I ACCEPT" THAT IS DISPLAYED BELOW. IF YOU DO NOT AGREE TO THIS LICENSE, CLICK THE ICON LABELED "CANCEL" AND YOUR ORDER WILL BE CANCELED, THE SOFTWARE WILL NOT BE DOWNLOADED AND YOU WILL NOT BE CHARGED.

License Grant

"You" means the person or company who is being licensed to use the Software or Documentation. "We," "us" and "our" means "True North Software Corporation."

We hereby grant you a nonexclusive license to use one copy of the Software on any single computer, provided the Software is in use on only one computer at any time. The Software is "in use" on a computer when it is loaded into temporary memory (RAM) or installed into the permanent memory of a computer—for example, a hard disk, CD-ROM or other storage device.

If the Software is permanently installed on the hard disk or other storage device of a computer (other than a network server) and one person uses that computer more than 80% of the time, then that person may also use the Software on a portable or home computer.

Title

We remain the owner of all right, title and interest in the Software and related explanatory written materials ("Documentation").

Archival or Backup Copies

You may copy the Software for back up and archival purposes, provided that the original and each copy is kept in your possession and that your installation and use of the Software does not exceed that allowed in the "License Grant" section above.

Things You May Not Do

The Software and Documentation are protected by United States copyright laws and international treaties. You must treat the Software and Documentation like any other copyrighted material—for example, a book. You may not:

- copy the Documentation,
- copy the Software except to make archival or backup copies as provided above,
- modify or adapt the Software or merge it into another program,
- reverse engineer, disassemble, decompile or make any attempt to discover the source code of the Software,
- place the Software onto a server so that it is accessible via a public network such as the Internet, or

- sublicense, rent, lease or lend any portion of the Software or Documentation.

Transfers

You may transfer all your rights to use the Software and Documentation to another person or legal entity provided you transfer this Agreement, the Software and Documentation, including all copies, updates and prior versions to such person or entity and that you retain no copies, including copies stored on computer.

Limited Warranty

We warrant that for a period of 90 days after delivery of this copy of the Software to you:

the media on which this copy of the Software is provided to you will be free from defects in materials and workmanship under normal use, and the Software will perform in substantial accordance with the Documentation.

To the extent permitted by applicable law, THE FOREGOING LIMITED WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES OR CONDITIONS, EXPRESS OR IMPLIED, AND WE DISCLAIM ANY AND ALL IMPLIED WARRANTIES OR CONDITIONS, INCLUDING ANY IMPLIED WARRANTY OF TITLE, NONINFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, regardless of whether we know or had reason to know of your particular needs. No employee, agent, dealer or distributor of ours is authorized to modify this limited warranty, nor to make any additional warranties.

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

Limited Remedy

Our entire liability and your exclusive remedy for breach of the foregoing warranty shall be, at our option, to either: return the price you paid, or repair or replace the Software or media that does not meet the foregoing warranty if it is returned to us with a copy of your receipt.

IN NO EVENT WILL WE BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING FROM THE USE OR THE INABILITY TO USE THE SOFTWARE (EVEN IF WE OR AN AUTHORIZED DEALER OR DISTRIBUTOR HAS BEEN ADVISED OF THE POSSIBILITY OF THESE DAMAGES), OR FOR ANY CLAIM BY ANY OTHER PARTY.

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

Term and Termination

This license agreement takes effect upon your use of the software and remains effective until terminated. You may terminate it at any time by destroying all copies of the Software and Documentation in your possession. It will also automatically terminate if you fail to comply with any term or condition of this license agreement. You agree on termination of this license to destroy all copies of the Software and Documentation in your possession.

Confidentiality

The Software contains trade secrets and proprietary know-how that belong to us and it is being made available to you in strict confidence. ANY USE OR DISCLOSURE OF THE SOFTWARE, OR OF ITS ALGORITHMS, PROTOCOLS OR INTERFACES, OTHER THAN IN STRICT ACCORDANCE WITH THIS LICENSE AGREEMENT, MAY BE ACTIONABLE AS A VIOLATION OF OUR TRADE SECRET RIGHTS.

General Provisions

1. This written license agreement is the exclusive agreement between you and us concerning the Software and Documentation and supersedes any prior purchase order, communication, advertising or representation concerning the Software.
2. This license agreement may be modified only by a writing signed by you and us.
3. In the event of litigation between you and us concerning the Software or Documentation, the prevailing party in the litigation will be entitled to recover attorney fees and expenses from the other party.
4. This license agreement is governed by the laws of the State of Minnesota.
5. You agree that the Software will not be shipped, transferred or exported into any country or used in any manner prohibited by the United States Export Administration Act or any other export laws, restrictions or regulations.