

ENVY[®]/QA R1.3

Software Quality Assurance Tools

User's Guide

IBM VisualAge for Smalltalk

The information in this document is subject to change without notice and should not be construed as a commitment by Object Technology International Inc. While precautions have been taken to ensure the accuracy of the contents of this document, Object Technology International Inc. assumes no responsibility for any errors or omissions.

This document is protected by copyright with all rights reserved. No part of this document may be copied by any means whatsoever without the prior written consent of Object Technology International Inc.

ENVY is a registered trademark of Object Technology International Inc.

VisualAge is a trademark of International Business Machines Corporation.

Windows is a registered trademark of Microsoft Corporation.

Microsoft Word is a registered trademark of Microsoft Corporation.

FrameMaker is a registered trademark of Frame Technology Corporation.

Other product names may be trademarks of their respective companies.

Copyright © 1996 - 1999 Object Technology International Inc. (OTI).

OTI is a wholly owned subsidiary of IBM Canada, Ltd.

Object Technology International Inc.

2670 Queensview Drive

voice: (613) 820-1200

Ottawa, Ontario

fax: (613) 820-1202

Canada K2B 8K1

This is the **ENVY/QA R1.3** User's Guide for IBM VisualAge for Smalltalk.

Document number: ED2900, Revision 1.0012

Date printed: June 2, 1999

Document Status: R1.3

Table of Contents

1	About This Guide	1
1.1	How to Use This Guide.....	1
1.2	How This Guide Is Organized.....	2
1.3	Conventions Used in This Guide.....	2
2	About ENVY/QA	3
2.1	Code Critic.....	4
2.2	Code Metrics	4
2.3	Code Coverage	4
2.4	Code Publisher	5
2.5	Code Formatter	5
2.6	Extensible Framework	5
3	Installing ENVY/QA	7
3.1	Release Files	7
3.2	Installing the Release Files	8
4	Code Critic	9
4.1	Overview.....	9
4.2	Who Should Use This Tool	9
4.3	Loading Code Critic.....	10
4.4	Unloading Code Critic.....	10
4.5	Guided Tour	11
4.5.1	Creating a Problem	11
4.5.2	Running Reviews	12
4.5.3	Browsing the Results	12
4.5.4	Fixing the Problems	13
4.6	Code Critic and the Development Browsers	13
4.6.1	Reviewing Configuration Maps	14
4.6.2	Reviewing Applications	14
4.6.3	Reviewing Classes	15
4.6.4	Reviewing Methods	15
4.7	Code Critic Options Dialog	16
4.8	Selecting Which Reviews to Run	16
4.9	Configuration Map Reviews	18
4.9.1	Missing Config Comment/Notes	18
4.10	Application Reviews	18
4.10.1	Missing WasRemovedCode	18
4.11	Class Reviews	18
4.11.1	Duplicate Pool Dictionaries	19
4.11.2	Extends Base Class	19
4.11.3	Missing Class Comment.....	19

4.11.4	Missing Dependent Method.....	19
4.11.5	Not Categorized Methods.....	19
4.11.6	Poorly Named State Variables.....	20
4.11.7	Subclass Responsibility.....	20
4.11.8	Subclasses Base Class.....	20
4.11.9	Unreferenced Class.....	20
4.11.10	Unused Pool Dictionaries.....	20
4.11.11	Unused State Variables.....	20
4.12	Method Reviews.....	21
4.12.1	Compiler Warnings.....	21
4.12.2	Could Be Cascaded.....	21
4.12.3	Could Use Self.....	21
4.12.4	Defeats Compiler Optimization.....	22
4.12.5	Direct State Variable Access.....	22
4.12.6	Identical to Inherited Method.....	22
4.12.7	Inefficient Convenience Method.....	23
4.12.8	Magic Values.....	23
4.12.9	Missing #yourself.....	23
4.12.10	Missing Method Comment.....	23
4.12.11	Missing Primitive Fail Code.....	23
4.12.12	Not Implemented in Superclass.....	24
4.12.13	Poorly Named Method.....	24
4.12.14	Poorly Named Variables.....	24
4.12.15	Public/Private Inconsistency.....	24
4.12.16	References Development Classes.....	24
4.12.17	References Global Variables.....	24
4.12.18	References outside Prereq Chain.....	25
4.12.19	References Own Class.....	25
4.12.20	Reimplements System Method.....	25
4.12.21	Sends System Method.....	25
4.12.22	Sent But Not Implemented.....	25
4.12.23	Should Call Superclass.....	26
4.12.24	Should Not Be Implemented.....	26
4.12.25	Should Use isEmpty.....	26
4.12.26	Too Many Consecutive Concatenations.....	26
4.12.27	Too Many Consecutive Messages.....	27
4.12.28	Unnecessary #isNil or #notNil.....	27
4.12.29	Unnecessary Parentheses.....	27
4.12.30	Unsent Method.....	27
4.12.31	Unused Arguments.....	28
4.13	Customizing the Reviews.....	28
4.13.1	Common Properties.....	28
4.13.2	Specific Properties.....	29
4.13.3	Modifying Properties.....	32
4.14	Saving and Loading Your Settings.....	33
4.15	Using the Code Critic Results Browser.....	34
4.15.1	Layout of the Code Critic Results Browser.....	34
4.15.2	Launching the Code Critic Results Browser.....	35
4.15.3	Sorting by Type.....	35
4.15.4	Sorting by Severity.....	35

4.15.5	Viewing a Problem	36
4.15.6	Ignoring Result.....	36
4.15.7	Removing Results	36
4.15.8	Refreshing Results	37
4.15.9	Saving and Loading Results	37
4.15.10	Saving and Loading Ignore Sets	37
4.15.11	Reporting Results	38
4.15.12	Exporting Results to Text.....	38
4.15.13	Exporting to Spreadsheet.....	38
4.15.14	Printing Results	38
4.15.15	Adding a Comment to the Results	38
4.15.16	Opening an Applications Browser.....	39
4.15.17	Opening Other Browsers	39
4.15.18	Menus	39
4.15.18.1	File Menu	39
4.15.18.2	Edit Menu.....	40
4.15.18.3	Review Menu	40
4.15.18.4	Result Menu.....	40
4.15.18.5	Info Menu	41
4.16	Advanced Concepts.....	42
4.16.1	Framework Configuration Maps	42
4.16.2	Code Critic Framework Classes	42
4.16.3	Implementing Your Review Subclass	42
4.16.3.1	Mandatory Methods	43
4.16.3.2	Optional Methods.....	43
4.16.4	Example.....	43
5	Code Metrics.....	47
5.1	Overview.....	47
5.2	Who Should Use This Tool.....	47
5.3	Loading Code Metrics	48
5.4	Unloading Code Metrics	48
5.5	Guided Tour	49
5.5.1	Creating a Class to Measure	49
5.5.2	Running the Metrics	50
5.5.3	Browsing the Results	50
5.6	Code Metrics and the Development Browsers	50
5.6.1	Measuring Configuration Maps.....	51
5.6.2	Measuring Applications	51
5.6.3	Measuring Classes	52
5.6.4	Measuring Methods	52
5.7	Code Metrics Options Dialog.....	53
5.8	Selecting Which Metrics to Run.....	53
5.9	Application Metrics	55
5.9.1	All Defined Classes	55
5.9.2	All Dependent Applications.....	55
5.9.3	All Extended Classes	55
5.9.4	All Prerequisites	56
5.9.5	All Subapplications	56
5.9.6	Defined Classes	56

5.9.7	Dependent Applications	56
5.9.8	Extended Classes	56
5.9.9	Memory Size (Including Subapplications).....	56
5.9.10	Memory Size for Applications	57
5.9.11	Prerequisites	57
5.9.12	Subapplications	57
5.10	Class Metrics	57
5.10.1	Accessors	57
5.10.2	All Class Methods	58
5.10.3	All Instance Methods	58
5.10.4	All Instance Variables	58
5.10.5	All Subclasses	58
5.10.6	Class Coupling.....	58
5.10.7	Class Methods	59
5.10.8	Class Response	59
5.10.9	Class Variables	59
5.10.10	Cyclomatic Complexity.....	59
5.10.11	Depth of Hierarchy.....	59
5.10.12	Direct Variable Accesses	60
5.10.13	Global/Pool References	60
5.10.14	Instance Methods	60
5.10.15	Memory Size for Classes	60
5.10.16	New Methods.....	60
5.10.17	Pool Dictionaries	61
5.10.18	Ratio API/Internal	61
5.10.19	Ratio Public/Private	61
5.10.20	Refined Methods.....	61
5.10.21	Specialization Index.....	61
5.10.22	Subclasses	62
5.11	Method Metrics.....	62
5.11.1	Lines of Code	62
5.11.2	Lorenz Complexity	62
5.11.3	Memory Size for Methods	62
5.11.4	Method Density.....	63
5.11.5	Statements.....	63
5.12	Customizing the Metrics	63
5.12.1	Common Properties	63
5.12.2	Specific Properties	64
5.12.3	Modifying Properties.....	65
5.13	Saving and Loading Your Settings	67
5.14	Using the Code Metrics Results Browser.....	67
5.14.1	Layout of the Code Metrics Results Browser.....	67
5.14.2	Launching the Code Metrics Results Browser.....	68
5.14.3	Hiding In-Range Results	69
5.14.4	Removing Results.....	69
5.14.5	Refreshing Results	69
5.14.6	Saving and Loading Results	70
5.14.7	Reporting Results	70
5.14.8	Exporting Reports to Text	70
5.14.9	Exporting to Spreadsheet.....	70

5.14.10	Printing Results	71
5.14.11	Adding a Comment to the Results	71
5.14.12	Opening an Applications Browser.....	71
5.14.13	Opening Other Browsers	71
5.14.14	Menus	72
5.14.14.1	File Menu.....	72
5.14.14.2	Edit Menu.....	72
5.14.14.3	Metric Menu.....	72
5.14.14.4	Result Menu.....	73
5.14.14.5	Info Menu	74
5.15	Advanced Concepts.....	74
5.15.1	Framework Configuration Maps	74
5.15.2	Code Metrics Framework Classes	74
5.15.3	Implementing Your Metric Subclass.....	75
5.15.3.1	Mandatory Methods	75
5.15.3.2	Optional Methods.....	75
5.15.4	Example.....	75
6	Code Coverage.....	79
6.1	Overview.....	79
6.2	Who Should Use This Tool.....	79
6.3	Loading Code Coverage	80
6.4	Unloading Code Coverage	80
6.5	Guided Tour	80
6.5.1	Creating an Application	80
6.5.2	Selecting the Applications to Watch.....	82
6.5.3	Watching Your Application	82
6.5.4	Ignoring Methods	82
6.6	Code Coverage and the Development Browsers.....	83
6.6.1	Coverage on Configuration Maps.....	83
6.6.2	Coverage on Applications	83
6.7	Using the Code Coverage Browser	84
6.7.1	Layout of the Code Coverage Browser.....	84
6.7.2	Launching the Code Coverage Browser.....	85
6.7.3	Status Area.....	85
6.7.4	The Control Buttons	85
6.7.5	Refreshing the Browser	86
6.7.6	Ignoring Components.....	86
6.7.7	Restoring Components.....	87
6.7.8	Saving and Loading Your Coverage Setup.....	87
6.7.9	Reporting Results	88
6.7.10	Exporting Reports to Text	88
6.7.11	Exporting to Spreadsheet.....	88
6.7.12	Printing Results	89
6.7.13	Adding a Comment to the Results	89
6.7.14	Menus	89
6.7.14.1	File Menu.....	89
6.7.14.2	Edit Menu.....	89
6.7.14.3	Applications Menu	89
6.7.14.4	Classes Menu	92

6.7.14.5	Methods Menu	92
6.7.14.6	Info Menu	93
7	Code Publisher	95
7.1	Overview.....	95
7.2	Who Should Use This Tool.....	96
7.3	Loading Code Publisher	96
7.4	Unloading Code Publisher	96
7.5	Guided Tour	97
7.5.1	Publishing Configuration Maps	98
7.5.2	Publishing Applications.....	98
7.5.3	Publishing Classes	99
7.5.4	Code Publisher Output Options Dialog.....	100
7.6	Output Formats.....	101
7.6.1	HTML—Hypertext Markup Language	101
7.6.2	LaTeX—A Document Preparation System.....	101
7.6.3	MIF—Maker Interchange Format.....	102
7.6.4	OTIML—OTI Markup Language.....	102
7.6.5	RTF—Rich Text Format.....	102
7.7	Code Publisher Settings Dialog	103
7.8	Application Publishing Options	104
7.9	Class Publishing Options	104
7.10	Method Publishing Options.....	105
7.11	Cross-Reference Publishing Options	105
7.12	General Publishing Options	105
7.13	Saving and Loading Your Settings	106
8	Code Formatter.....	107
8.1	Overview.....	107
8.2	Who Should Use This Tool.....	107
8.3	Loading Code Formatter	108
8.4	Unloading Code Formatter	108
8.5	Guided Tour	108
8.5.1	Creating a Class and Method	108
8.6	Code Formatter and the Development Browsers.....	109
8.6.1	Formatting Configuration Maps	110
8.6.2	Formatting Applications.....	110
8.6.3	Formatting Classes	110
8.6.4	Formatting Methods	111
8.7	Formatting Source	111
8.8	Code Formatter Settings Dialog	112
8.9	Formatting Blocks	114
8.9.1	Block Arguments on the Start Line.....	114
8.9.2	Block Variables on a New Line.....	114
8.9.3	Line up Brackets	115
8.9.4	Space between Square Brackets	115
8.9.5	Start Open Brackets on the Same Line.....	116
8.10	Formatting Comments.....	116
8.10.1	Format Comments in the Body of the Code.....	116
8.10.2	Format Method Header Comment.....	117

8.11	Formatting Conditional Statements	117
8.11.1	Apply Rule (Line up #ifTrue:ifFalse:) to All	117
8.11.2	Indent #ifFalse: (When Not Lined Up)	118
8.11.3	Line up #ifTrue:ifFalse:.....	119
8.12	Formatting Indentation	119
8.12.1	Indent Long Keywords in Cascades	120
8.12.2	Indent Nested Receivers	120
8.13	Formatting Keywords	120
8.13.1	Always Put a Single Keyword on the Same Line	121
8.13.2	Always Split Keywords	121
8.13.3	First Selector of Long Keywords on the Same Line.....	121
8.13.4	Keep #to:do: on the Same Line	122
8.13.5	Keep Right-Hand Side of an Assignment (:=) on the Same Line	122
8.14	Formatting Line Breaks	123
8.14.1	Blank Line between Comments and Temps	123
8.14.2	Blank Line between Selector and Comments	123
8.14.3	Blank Line between Temps and the Body of the Method.....	124
8.14.4	Retain Blank Lines	124
8.15	Formatting Parentheses	125
8.15.1	Line up Array Parentheses	125
8.15.2	Line up Parentheses	125
8.15.3	Space between Parentheses.....	126
8.15.4	Start Open Array Parentheses on the Same Line	126
8.15.5	Start Open Parentheses on the Same Line	127
8.16	General Formatting.....	127
8.16.1	Add Optional Periods	127
8.16.2	Add a Space after Return (^)	128
8.16.3	Use the Maximum Width (Ignore the Window Width)	128
8.17	Advanced Options	129
8.17.1	Formatting Code Indentation.....	129
8.17.1.1	Code.....	130
8.17.1.2	Code Continuation	130
8.17.2	Formatting Comment Indentation	130
8.17.2.1	Beside Line of Code	130
8.17.2.2	Comment.....	131
8.17.2.3	Comment Continuation	131
8.17.3	Formatting Margins	132
8.17.3.1	Maximum Width.....	132
8.17.3.2	Minimum Width for Comments	132
8.18	Saving and Loading Your Settings	133
9	Troubleshooting	135
	Bibliography.....	137
	Index.....	138

List of Figures

Figure 1 — Code Critic Options dialog	16
Figure 2 — Code Critic Settings dialog.....	17
Figure 3 — Code Critic Advanced Settings dialog.....	32
Figure 4— Code Critic Results Browser.	34
Figure 5 — Code Metrics Options dialog.....	53
Figure 6 — Code Metrics Settings dialog.....	54
Figure 7 — Code Metrics Advanced Settings dialog.....	66
Figure 8 — Code Metrics Results Browser.....	68
Figure 9 — Code Coverage Browser.	84
Figure 10 — Code Publisher Output Options dialog.....	100
Figure 11 — Code Publisher Settings dialog.....	103
Figure 12 — Code Formatter Settings dialog.....	113
Figure 13 — Formatter Advanced Settings dialog.....	129

List of Tables

Table 1 — Reviewing applications.....	15
Table 2 — Reviewing classes.....	15
Table 3 — Review groupings.....	17
Table 4 — Enabling reviews.....	18
Table 5 — Default severity levels.....	29
Table 6 — Specific review properties.....	31
Table 7 — Code Critic property values and their descriptions.....	33
Table 8 — Elements of the Code Critic Results Browser.....	35
Table 9 — Subclassing CtMeasure.....	42
Table 10 — Measuring applications.....	52
Table 11 — Measuring classes.....	52
Table 12 — Metrics groupings.....	54
Table 13 — Enabling metrics.....	55
Table 14 — Default lower and upper thresholds.....	64
Table 15 — Specific metric properties.....	65
Table 16 — Code Metrics property values and their descriptions.....	66
Table 17 — Elements of the Code Metrics Results Browser.....	68
Table 18 — Subclassing CtMeasure.....	75
Table 19 — Watching applications.....	83
Table 20 — Elements of the Code Coverage Browser.....	85
Table 21 — Ignoring methods in a component.....	87
Table 22 — Restoring all methods in a component.....	87
Table 23 — Publishing applications.....	99
Table 24 — Publishing classes.....	99
Table 25 — Elements of the Code Publisher Output Options dialog.....	100
Table 26 — MIF output files.....	102
Table 27 — Groupings of publisher options.....	103
Table 28 — Enabling publisher options.....	104
Table 29 — Application publishing options.....	104
Table 30 — Class publishing options.....	105
Table 31 — Method publishing options.....	105
Table 32 — Cross-reference publishing options.....	105
Table 33 — General publishing options.....	106
Table 34 — Formatting applications.....	110
Table 35 — Formatting classes.....	111
Table 36 — Categories of formatter options.....	113
Table 37 — Enabling formatter options.....	114
Table 38 — Property values and their descriptions.....	129

1 About This Guide

This guide describes how to set up and use **ENVY/QA**, which is a suite of software quality-assurance tools packaged in a flexible framework. **ENVY/QA** comprises:

- Code Critic
- Code Metrics
- Code Coverage
- Code Publisher
- Code Formatter
- User-extensible QA framework

1.1 How to Use This Guide

This guide is written for experienced users of **ENVY/Developer**.

Because **ENVY/QA** is a multiplatform product, screen shots and the conventions for capitalizing menu items may vary, depending on the Smalltalk platform and operating system you are using. This document includes screen shots and menu items created with a generic Smalltalk platform and a Windows presentation manager.


1.2 How This Guide Is Organized


The remainder of this guide is divided into several chapters. The next chapter introduces **ENVY/QA**. The following chapter describes the installation procedure. The remaining chapters describe each **ENVY/QA** tool.

1.3 Conventions Used in This Guide

The following typographical conventions are used in this guide.

- Class names are shown as *OrderedCollection*.
- Configuration maps are shown as **ENVY/QA for VisualAge Smalltalk**.
- Method selectors are shown as **#at:put:**.
- Menu options are shown as **Edit** → **Copy**.
- Browsers are shown as **Application Manager**.
- Variables are shown as temperature.
- Filenames are shown as FILE1 .TXT.
- Keys are shown as SHIFT.

 A hint, clarification, or comment is shown like this.

 A warning is shown like this.

Sample code is displayed as:

```
self at: aName put: aValue
```

2 About ENVY/QA

ENVY/QA is a suite of software quality-assurance tools packaged in a flexible framework. The tools comprise:

- Code Critic
- Code Metrics
- Code Coverage
- Code Publisher
- Code Formatter
- User-extensible QA framework

ENVY/QA provides the critical quality-assurance tools that development teams need to consistently produce high-quality production software with low defect counts. Managers can analyze development team productivity; they can publish Smalltalk applications to promote reuse within the enterprise. Source-code formatting supports enterprise-wide standards.

ENVY/QA R1.3 loads into the following development environments:

- IBM VisualAge for Smalltalk

ENVY/QA uses the following terminology for code elements.

Configuration map

A collection of applications.

Application

A collection of class fragments and (sub)applications.

Class

A class definition and all class and instance methods.

Class fragment

The portion of a class within a given application. This consists of any methods defined by the class within the application. If the class is defined by the application, the class fragment contains the class definition.

Method

The implementation for a method selector.

2.1 Code Critic

Code Critic analyzes methods, classes, applications, and configuration maps for potential common problems. You can view results on line, print them, or export them to text or spreadsheet format.

Code Critic's heuristics minimize the number of false defects, allowing you to focus on the important problems. Code Critic is **ENVY/Packager**-aware, using **ENVY/Packager** methods defined by applications. You can customize Code Critic reviews on line to allow for subtle differences in project priorities. You can customize settings, save them on disk, and load them from disk.

Code Critic provides an open and extensible framework that lets you easily create new code reviews to be included automatically in the user interface.

2.2 Code Metrics

Code Metrics gathers static metrics on methods, classes, applications, and configuration maps. You can define thresholds for each metric. You can view all results or focus on methods outside the thresholds. You can customize settings, save them on disk, and load them from disk.

You can print result reports or export them to text format or to spreadsheet format. You can customize report sections.

Results contain summary statistics and fine-grained data that let you focus on the system at a high level or focus on individual areas.

Code Metrics provides an open and extensible framework that lets you easily create new code metrics to be included automatically in the user interface.

2.3 Code Coverage

Code Coverage lets you monitor a set of applications during regression testing. You interact with the applications in the normal manner through a user interface test plan or by executing automated test

suites. As the application's methods are executed, Code Coverage tracks the unexecuted methods and focuses your attention on the untested areas of the system. Unexecuted methods indicate potential gaps in the test plan.

You can save and load settings that include information about the applications being tested. This lets you set up future coverage sessions quickly to ensure that changes to your test suites are achieving the required level of software coverage.

Code Coverage produces printed or file-based reports that detail the coverage results. You can customize report sections.

2.4 Code Publisher

Code Publisher produces typeset-quality manuals from applications, classes, and methods. The report structure is highly customizable. You can easily create documents that include only the API methods and their comments. You can produce in-depth manuals containing code, cross-reference tables, and quick look-up indexes to be used during code reviews.

Code Publisher can export to these formats:

- LaTeX
- RTF
- MIF
- HTML
- SGML (OTIML DTD)

HTML manuals are internally hyperlinked to let you navigate easily on line. To further improve readability, embedded GIF images are included in HTML output.

2.5 Code Formatter

Code Formatter lets you format Smalltalk source code. Custom controls let you define the formatting style. You can preview a style to determine quickly the effect on the code.

In addition to letting you format methods, Code Formatter lets you format entire classes, class hierarchies, and applications.

2.6 Extensible Framework

ENVY/QA contains an open and extensible tools framework that lets you develop new QA tools easily. A tool registers the types of objects on which it operates. The framework then ensures automatically that the

tool is displayed in the appropriate development browsers. You can build new QA tools without learning the details of the browsers.

3 Installing ENVY/QA

ENVY/QA is delivered in a standard library export file that contains several configuration maps. The main configuration map for the VisualAge environment is:

ENVY/QA for VisualAge Smalltalk

This configuration map is contained in a Smalltalk-specific library export file. The installation process is a normal import and load sequence. The supplied configuration map is self-contained and no other application support files are required, with the exception of the publishing template and style-support files.

3.1 Release Files

ENVY/QA is shipped on a CD-ROM that contains all supported platforms for a specific Smalltalk vendor. In this way, developers can change platforms easily without exchanging media or licenses; however, single-user license restrictions still apply.

3.2 Installing the Release Files

Please see the `README.TXT` file on your CD-ROM for installation instructions for **ENVY/QA**.

After installation, newly-opened development browsers have a **Tool** menu option on their menus. In addition, the Transcript's **ENVY** or **Smalltalk Tools** pulldown menus have the new options **Tools** and **Tool Settings**.

4 Code Critic

4.1 Overview

Code Critic lets you detect potential problems automatically in a body of code. It is integrated fully with the existing development browsers and provides an ideal environment for integrating software quality assurance directly into the development process.

Code Critic has an extensible set of reviews. A review is a specific type of measure that executes over code elements, and either completes successfully or produces warnings.

Warnings detected by Code Critic do not necessarily indicate wrong code. You should use Code Critic as an automated first pass to identifying potential problems. You should then decide whether an identified warning is a problem within the scope of ongoing development.

The **Code Critic Results Browser** lets you view Code Critic's results and correct problems.

4.2 Who Should Use This Tool

Developers can use Code Critic to:

- standardize coding styles and improve the consistency of code among team members

- quickly identify and correct common bugs
- focus on potential areas for detailed code inspections

Project managers can use Code Critic to:

- identify design or implementation dependencies
- obtain a quick summary of the state of a component

Release engineers can use Code Critic to:

- gauge the overall quality of a component
- identify potential packaging problems

4.3 Loading Code Critic

Code Critic is installed automatically as part of the complete installation of **ENVY/QA** (see Chapter 3).

If you want to use only Code Critic:

1. Open the **Configuration Maps Browser**.
2. Load the most recent edition of the **QA Code Critic** configuration map with required maps.

When Code Critic is loaded:

- the system menu has new submenus:
 - **Tool** → **Review...**
 - **Tool Settings** → **Review...**
- newly opened development browsers have a **Tool** → **Review** menu

You are now ready to use Code Critic.

4.4 Unloading Code Critic

To unload Code Critic:

1. Open the **Configurations Maps Browser**.
2. Unload the loaded edition of the **QA Code Critic** configuration map.

✘ Before unloading Code Critic, make sure that all **Code Critic Results Browsers** are closed.

3. If Code Critic is the only **ENVY/QA** tool loaded, you can also unload (in order) the following configuration maps:
 - (a) **QA Code Tools (CC/CM) Framework**
 - (b) **QA Framework**

4.5 Guided Tour

In this section, we guide you through the process of:

- creating a problem
- running reviews
- browsing the results
- fixing the problems

4.5.1 Creating a Problem

Create the following class and methods. Note that the method **named:age:** intentionally introduces a typing error to demonstrate a message generated by the code review. Please type the method as shown.

Person class

```
Object subclass: #Person
  instanceVariableNames: 'name age '
  classVariableNames: "
  poolDictionaries: "
```

Person public class methods

```
named: aString age: anInteger
  "Answer a new instance of the receiver whose name is aString
  and age is anInteger."

  ^self new
    mane: aString;
    age: anInteger
```

Person public instance methods

```
age
  ^age

happyBirthday
  "The receiver has had a birthday.
  Display a Happy Birthday greeting, and make him/her a year older."

  age := age + 1.
  System message: ('Happy ', self age printString, ' Birthday, ', self name, '!')

name
  "Answer the name (String) of the receiver."

  ^name
```

Person private instance methods

```
age: anInteger
    "Set the age (Integer) of the receiver to anInteger."

    age := anInteger

name: aString
    "Set the name (String) of the receiver to aString."

    name := aString
```

4.5.2 Running Reviews

After you have added the *Person* class and its methods, run Code Critic as follows.

1. Open a **Classes Browser** and select the *Person* class.
2. Select **Classes** → **Tool** → **Review** → **Class...**
3. When the **Code Critic Options** dialog opens, click **OK**.

Code Critic runs all reviews on the *Person* class.

4.5.3 Browsing the Results

After the reviews run, the **Code Critic Results Browser** displays the results. To browse the results:

1. In the left list, select review **Direct state variable access**.

The right list shows the results of the review type.

2. In the right list, select method **Person>>#happyBirthday**.

The text area shows the source code of the method and highlights `age`. This indicates that the instance variable `age` is accessed directly without an accessor. The status area above the source area provides additional information about the problem.

3. Select the method again.

The next problem is highlighted automatically. You can also click **Next** to advance to the next problem.

4.5.4 Fixing the Problems

When the text of the method is displayed in the text area:

1. Correct the problem by modifying the method text to:

```
happyBirthday
    "The receiver has had a birthday.
    Display a Happy Birthday message, and increment his/her age."

    self age: self age + 1.
    System message: ('Happy ', self age printString, ' Birthday, ', self name, '!')
```

2. Save the changed method.

When the problem is corrected, Code Critic automatically removes the method from the **Code Critic Results Browser**.

3. Select the review **Missing method comment** and method **Person>>#age**. (The method is missing a method comment.)
4. Modify the method to the following and save it:

```
age
    "Answer the age (Integer) of the receiver."
    ^age
```

5. Select the review **Sent but not implemented** and method **Person class>>#named:age:**. (The method has a spelling error: **#mane:** should be **#name:**.)
6. Correct the spelling error and save the method. The method is now:

```
named: aString age: anInteger
    "Answer a new instance of the receiver whose name is aString
    and age is anInteger."

    ^self new
        name: aString;
        age: anInteger;
        yourself
```

7. Correct the remaining problems in a similar manner.

4.6 Code Critic and the Development Browsers

Code Critic is integrated fully with the development browsers. You can review code elements using **Tool** → **Review** in applicable development browsers.

Code Critic measures your code's compliance to project guidelines and standard practices. You can review the following types of code elements:

- method
- class
- class fragment
- class hierarchy
- application
- application hierarchy
- configuration map

4.6.1 Reviewing Configuration Maps

To review a configuration map:

1. Open the **Configuration Maps Browser**.
2. Select a configuration map edition.
3. Select **Editions** → **Tool** → **Review**.
4. In the **Code Critic Options** dialog, click **OK** to start the review.

4.6.2 Reviewing Applications

You can review applications from the following browsers:

- **Configuration Maps Browser**
- **Application Manager**
- **Applications Browser**
- **Classes Browser**
- **Class Hierarchy Browser**
- **Visual Age Organizer**

To review an application

1. Open one of the browsers mentioned above.
2. Select the applications to be reviewed.

3. Select a menu option:

Item to review	Menu option
applications	Applications → Tool → Review → Application... ¹
applications and their loaded subapplications	Applications → Tool → Review → Include All Subapplications..

Table 1 — Reviewing applications.

4. In the **Code Critic Options** dialog, click **OK** to start the review.

4.6.3 Reviewing Classes

You can review classes from the following browsers:

- **Application Manager**
- **Application Browser**
- **Applications Browser**
- **Classes Browser**
- **Class Hierarchy Browser**
- **Visual Age Organizer**

To review a class:

1. Open one of the browsers mentioned above.
2. Select the classes to be reviewed.
3. Select a menu option:

Item to review	Menu option
classes	Classes → Tool → Review → Class...
classes and all their subclasses	Classes → Tool → Review → Include All Subclasses...
class fragments	Classes → Tool → Review → Class Fragment...

Table 2 — Reviewing classes.

4. In the **Code Critic Options** dialog, click **OK** to start the review.

4.6.4 Reviewing Methods

You can review methods from the following browsers:

- **Application Browser**
- **Applications Browser**

¹ In the **Configuration Maps Browser**, select **Applications** → **Tools** → **Review**.

- **Classes Browser**
- **Class Browser**
- **Class Hierarchy Browser**
- **Methods Browser**

To review a method:

1. Open one of the browsers mentioned above.
2. Select the methods to be reviewed.
3. Select **Method** → **Tool** → **Review**.
4. In the **Code Critic Options** dialog, click **OK** to start the review.

4.7 Code Critic Options Dialog

When you select **Tool** → **Review**, the **Code Critic Options** dialog opens. From this dialog you can run either all reviews (by clicking **All**), or a specific set of reviews (by clicking **Settings...**).

If you have run Code Critic previously and saved a set of ignored problems, you can choose to reuse this ignore set by toggling **Use Ignore Set**. Problems from the ignore set are ignored automatically if they occur again.

Click **OK** to start the review.

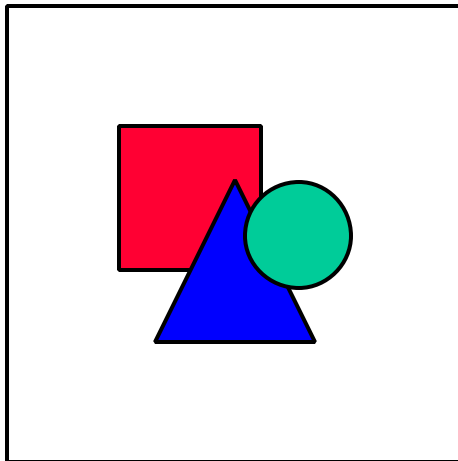


Figure 1 — Code Critic Options dialog.

4.8 Selecting Which Reviews to Run

When you are familiar with the different types of reviews, you may find that some reviews are more useful than others at different stages of

development. In such cases, you may want to run only some of the reviews.

You can select specific reviews in the **Code Critic Settings** dialog shown below.

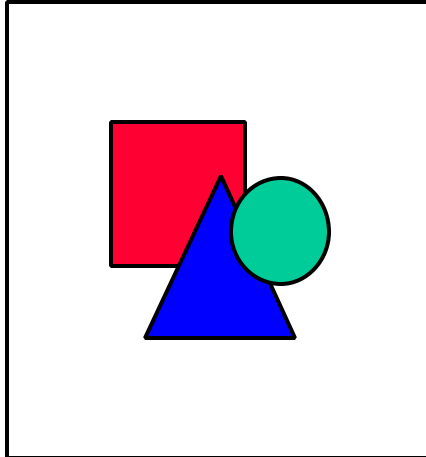


Figure 2 — Code Critic Settings dialog.

The **Code Critic Settings** dialog groups reviews into the categories shown in the following table.

Category	Explanation
Inheritance	class inheritance problems, including public/private inconsistencies and class extensions of base classes
Missing	missing code elements, including missing method comments and missing dependent method
References	improper references between code elements, including references to globals and references outside the prerequisite chain
Style	coding style, including code that could be cascaded and poorly named methods
System	low-level system problems, including methods that send system messages and methods that reimplement system methods
Unused	unused code elements, including unused pool dictionaries and unused state variables

Table 3 — Review groupings.

To select specific reviews:

1. In the **Code Critic Options** dialog, select **Settings...**, or from the Transcript, select **Tool Settings** → **Review...**
2. Select a category in the left list.

The list on the right shows a group of check boxes representing each review in that category.

3. Do one of the following:

Desired result	Action
enable all reviews in a category	click All
disable all reviews in a category	click None
enable or disable specific reviews	click individual check boxes
reset to their default values all reviews in all categories	click Reset All

Table 4 — Enabling reviews.

4. Click **OK** to accept the selected reviews or **Cancel** to abort the selection.

4.9 Configuration Map Reviews

This section describes the review that you can run on configuration maps.

4.9.1 Missing Config Comment/Notes

Warns if a configuration map does not have a comment or notes.

Advice

Use configuration map comments and notes to provide a brief description of the configuration map.

4.10 Application Reviews

This section describes the reviews that you can run on applications.

4.10.1 Missing WasRemovedCode

Warns if a subapplication has `toBeLoadedCode` but does not have `wasRemovedCode`.

Advice

If a subapplication performs operations before it can be loaded, it should probably also perform operations to clean up after it is unloaded. Place the appropriate code into the application's `wasRemovedCode` inherited user field by using **#wasRemovedCode:**

4.11 Class Reviews

This section describes the reviews that you can run on classes.

4.11.1 Duplicate Pool Dictionaries

Warns if a class declares a pool dictionary already declared by one of its superclasses.

Advice

Remove the pool dictionary from the subclass.

4.11.2 Extends Base Class

Warns if a class is an extension of a base class.

Advice

Avoid extending generic base classes. Extending base classes introduces additional complexity when receiving new releases of the base software. Also, multiple applications extending a base class can potentially collide in the use of the method name. If an extension becomes necessary, consult your project's guidelines for conventions to use when adding extensions to base classes.

4.11.3 Missing Class Comment

Warns if a class does not have a comment.

Advice

Class comments help to explain the class. Consult your project's guidelines to determine the format of a class comment.

4.11.4 Missing Dependent Method

Warns if a class implements a method but does not implement its dependent method.

For example, if a class implements `=`, it should typically implement `#hash`.

Advice

Add the missing dependent method.

4.11.5 Not Categorized Methods

Warns if a class has methods that are not categorized.

Advice

You can use method categorization to help organize the methods of a class. Consult your project's guidelines to determine the naming conventions.

4.11.6 Poorly Named State Variables

Warns if a class has state variables (instance, class, or class instance) that are poorly named.

For example, temp1 is a poorly named instance variable.

Advice

Give state variables meaningful names.

4.11.7 Subclass Responsibility

Warns if a class does not override a superclass method that contains **#subclassResponsibility**.

Advice

Failure to override a subclass responsibility method could result in a walkback.

4.11.8 Subclasses Base Class

Warns if a class is a subclass of a base class.

Advice

Avoid subclassing generic system classes. You have no control over the implementation of these classes. Make sure that you are subclassing because the objects are logically related, rather than physically related. For example, a telephone book is not a *Dictionary* but it might have some of its state represented using a *Dictionary*.

4.11.9 Unreferenced Class

Warns if the class is not visibly referenced.

Advice

If the class is not an abstract superclass and is not used either directly or indirectly, you should remove it.

4.11.10 Unused Pool Dictionaries

Warns if the class includes a pool dictionary whose variables are not referenced by any of the methods in the class or in its subclasses.

Advice

If appropriate, remove the pool declaration.

4.11.11 Unused State Variables

Warns if a state variable defined by the class is not referenced by any methods in the class or in its subclasses.

Advice

Remove the state variable.

4.12 Method Reviews

This section describes reviews that you can run on methods.

4.12.1 Compiler Warnings

Warns if a method has code or temporaries that are:

- unused
- read before written
- written but not read
- not optimized

Advice

Unused variables waste execution space. Variables read before being written could result in a walkback. Variables written but not read either waste execution space or indicate a potential bug if the variable was intended to be used. Move variables that are not optimized into the context of a block within the method.

4.12.2 Could Be Cascaded

Warns if a method could use cascade messages instead of multiple individual message sends.

Advice

Cascaded statements can be faster than individual message sends. Use:

```
self a; b.
```

instead of:

```
self a. self b.
```

In some situations, a cascaded message can reduce the readability of your code. Before cascading your code, consider the effect the change will have on readability.

4.12.3 Could Use Self

Warns if a method is sent to the superclass explicitly but the method is not implemented in any of the superclasses.

Advice

Do not use the superclass if the subclass does not implement the method being called. In this situation, use `self`. For example, the

subclass calls `super xxxx`, where `xxxx` is not implemented in the subclass. Instead, write it as `self xxxx`.

4.12.4 Defeats Compiler Optimization

Warns if an optimized method is misused.

Advice

This warning occurs with code of the form:

```
ifTrue: aVariableHoldingABlock.
```

The system recognizes special selectors (for example, `ifTrue:[]`) and their blocks, and optimizes them. The above form defeats these compiler optimizations. Review the code for an alternative approach; for example, use:

```
ifTrue:[aVariableHoldingABlock value].
```

4.12.5 Direct State Variable Access

Warns if a method directly references a state variable (instance, class or class instance variables).

Accessor methods are not flagged because they are expected to directly reference state variables. Delayed initialization accessors are not pure accessors (that is, they perform computation) and therefore are flagged. Instance methods acting as class variable accessors are also flagged because class-variable access is typically done using class methods.

Advice

Accessors can help preserve the encapsulation principle of object-oriented programming. Direct accesses can also increase maintenance or limit the ability of a subclass to extend or refine the class's behavior.

4.12.6 Identical to Inherited Method

Warns if a method's implementation is identical to that of its superclass.

Advice

Before you remove the duplicate method from the subclass, make sure the duplication did not occur because the subclass behavior was not properly refined. In certain situations, you might be implementing a class method that is identical to an inherited method from *Class* or *Behavior* (for example, `#initialize`). When you browse the hierarchy of your class, the development browsers might not display these classes' superclasses (*Class*, *Behavior*).

4.12.7 Inefficient Convenience Method

Warns if an inefficient convenience method is implemented.

Advice

Do not implement a convenience method that will prevent compiler optimization.

4.12.8 Magic Values

Warns if the method contains hard-coded numeric constants.

Advice

A hard-coded number has little semantic value and may increase maintenance problems. Carefully document hard-coded numbers. Place them into methods that return them or place them into class pools.

4.12.9 Missing #yourself

Warns if an expression is missing **#yourself**.

Advice

In an assignment statement (:=), the message **#yourself** is often forgotten at the end of an instance creation followed by several cascaded messages. Use:

```
aCollection := OrderedCollection new add: a; add: b; yourself
```

and *not*:

```
aCollection := OrderedCollection new add: a; add: b
```

4.12.10 Missing Method Comment

Warns if a method does not have a comment at the beginning of the method.

Advice

Comment methods to explain their purpose, arguments, and return value. Consult your project's guidelines for the format of method comments.

4.12.11 Missing Primitive Fail Code

Warns if a primitive method does not have code to handle failures.

Advice

Add code to handle the failure.

4.12.12 Not Implemented in Superclass

Warns if a message is sent to the superclass explicitly but the method is not implemented in the superclass.

Advice

Do not use the superclass, because the method is not implemented in any of the superclasses. Check the spelling of the method.

4.12.13 Poorly Named Method

Warns if a method is poorly named. For example, **#selfLocation** is a poorly named method.

Advice

Use meaningful method names to increase the readability of your code.

4.12.14 Poorly Named Variables

Warns if an argument or temporary variable is poorly named.

Advice

Use meaningful variable names to increase the readability of your code.

4.12.15 Public/Private Inconsistency

Warns if a method is:

- public in its class and private in its superclass
- or
- private in its class and public in its superclass

Advice

Review the methods and, if appropriate, make them consistent.

4.12.16 References Development Classes

Warns if a method references a development class.

Advice

Referencing a development class can cause problems when a product is prepared for shipment. These references can cause unnecessary code bulk. Also, some development classes (for example, *Compiler*) cannot be used in a runtime application.

4.12.17 References Global Variables

Warns if a method references a global variable.

Advice

If a class has too many references to global variables, the class may rely on too much shared information, which may be a design problem.

4.12.18 References outside Prereq Chain

Warns if a method references a non-visible class. Visible classes are those found in the method's application, in its subapplications, or along the application's prerequisite chain.

Advice

Correct the application's prerequisites to include the appropriate application containing the referenced class.

4.12.19 References Own Class

Warns if a method references its own class.

Advice

Avoid referencing the receiver's class directly. Use `self class`. Directly referencing the receiver's class can limit the ability of subclasses to refine or extend behavior.

4.12.20 Reimplements System Method

Warns if a method implements a system (low-level) method.

Advice

Implementing methods, such, as **#basicAt:** and **#basicNew** can be risky. Carefully review these implementors and use an alternative approach if possible.

4.12.21 Sends System Method

Warns if a method sends any of the messages that should typically be avoided (for example, **#basicAt:**).

Advice

Avoid using system (low-level) messages.

4.12.22 Sent But Not Implemented

Warns if any messages are sent but not implemented by a visible class. Visible classes are those found in the method's application, along its prerequisite's chain, and in its subapplications. You can configure this review to search for implementors in the entire image instead of searching only the visible classes. **ENVY/Packager** directives (associated with applications) can also be taken into account to prevent symbols being considered potential messages.

Advice

Implement the required method or change the message send. If the message is used only as a symbol, you can ignore it by using the **ENVY/Package** directive methods in your application.

4.12.23 Should Call Superclass

Warns if the checked method specializes selectors but does not call their superclass implementation. For example, when developers implement **#initialize**, they commonly call `super initialize`.

Advice

Review the method to ensure it does not need to call the **#initialize** method in its superclass.

4.12.24 Should Not Be Implemented

Warns if a method overrides a superclass method that sends **#shouldNotImplement**.

Advice

This warning indicates a potential problem with the design of the class hierarchy.

4.12.25 Should Use isEmpty

Warns if a method misuses **#size**; for example, as in:

```
object size > 0
```

instead of:

```
object notEmpty
```

Advice

Use **#isEmpty** and **#notEmpty**.

4.12.26 Too Many Consecutive Concatenations

Warns if the method contains statements with too many consecutive concatenations; for example, as in:

```
'hello ',self firstName,', ',self lastName,', . How are you ',self weather printString,' today? '
```

Advice

Too many consecutive concatenations create unnecessary intermediate collections, which could affect the performance of the code.

4.12.27 Too Many Consecutive Messages

Warns if the method contains statements with too many consecutive messages; for example, as in:

```
self values location positions maximum x
```

Advice

Too many consecutive messages indicate a design problem. This condition can indicate poor encapsulation of behavior or the need for additional API in other objects.

4.12.28 Unnecessary #isNil or #notNil

Warns if **#isNil** or **#notNil** is sent to the receiver by itself; for example, `self isNil` is typically unnecessary.

Advice

The receiver should know whether it is nil or not without sending a message to determine this. All objects, except the *UndefinedObject*, return false for the **#isNil** message, and return true for the **#notNil** message.

4.12.29 Unnecessary Parentheses

Warns if the method has unnecessary parentheses. The following are detected:

- parentheses around a unary message
- parentheses around a bracketed expression

Advice

Remove the additional parentheses unless they improve readability significantly.

4.12.30 Unsent Method

Warns if a method is implemented but not sent by any class in the method's application, in its subapplication, or by a class along the chain of dependent applications. You can configure this review to search for senders in the entire image instead of searching along the dependent application chain. **ENVY/Packager** directives (associated with applications) can also be taken into account to prevent symbols found in code being considered potential messages.

Advice

This warning indicates a potentially obsolete method. If a sender is found in the image but not along the dependency chain, a problem may exist with the prerequisites of the method's application.

4.12.31 Unused Arguments

Warns if a method has unused arguments.

Advice

This warning indicates potentially obsolete behavior. However, a method can support arguments but ignore them in order to provide a common API among several objects.

4.13 Customizing the Reviews

Each review has properties that affect the way the review checks code elements. Some properties are common to all reviews and others are specific to a review.

4.13.1 Common Properties

All reviews have the property *severity level*. Severity level is a number (greater than or equal to one) that indicates the severity of results found by the review. All reviews have a default severity level that you can modify to match your project's guidelines.

The following table summarizes the default severity levels for each review. (Severity level 1 is the most severe.)

Review	Default severity
Compiler warnings	2
Could be cascaded	3
Could use self	2
Defeats compiler optimization	3
Direct state variable access	2
Duplicate pool dictionaries	3
Extends base class	2
Identical to inherited method	1
Inefficient convenience method	1
Magic values	3
Missing #yourself	3
Missing class comment	3
Missing config comment/notes	3
Missing dependent method	2
Missing method comment	3
Missing primitive fail code	1
Missing wasRemovedCode	2
Not categorized methods	3
Not implemented in superclass	3

Review	Default severity
Poorly named method	2
Poorly named state variables	2
Poorly named variables	2
Public/private inconsistency	3
References development classes	2
References global variables	2
References outside prereq chain	2
References own class	2
Reimplements system method	1
Sends system method	1
Sent but not implemented	1
Should call superclass	1
Should not be implemented	2
Should use isEmpty	3
Subclass responsibility	1
Subclasses base class	2
Too many consecutive concatenations	2
Too many consecutive messages	2
Unnecessary #isNil or #notNil	3
Unnecessary parenthesis	3
Unreferenced class	1
Unsent method	2
Unused arguments	3
Unused pool dictionaries	3
Unused state variables	1

Table 5 — Default severity levels.

4.13.2 Specific Properties

Some reviews have properties specific to that review. The following table summarizes the properties for those reviews and the default value for the property. You can customize any of these property values using the **Advanced Settings** dialog. The default values may be different depending on your Smalltalk vendor; consult the **Advanced Settings** dialog.

Review	Property	Description	Default value
Defeats compiler optimization	Optimized methods	methods that are optimized by the compiler; the format is: ifTrue: ifFalse:	#and: #ifFalse:...
Direct state variable access	Ignore same name accessors	if the property is set to true, the review ignores any selectors that have the same name as the state variable (You can use the property to ignore selectors that perform lazy initialization.)	false
Extends base class	Base classes	base classes	#AdditiveSequenceableCollec...
Inefficient convenience method	Inefficient convenience selectors	the names of inefficient convenience methods	#ifNil: #ifNotNil:
Magic values	Ignore values	common values that are not considered to be hard-coded values	-1 0 1 2
Missing #yourself	Instance creation methods	methods that are optimized for instance creation	#basicNew #basicNew: #new...
Missing dependent method	Dependent methods	a list whose elements indicate a method and its dependent method; for example: (= -> #hash)	= -> #hash #loaded ->...
Poorly named method	Poor names	the patterns for poor method names	"0" "1" "2" "3" "4"...
Poorly named state variables	Poor names	the patterns for poor state variable names	"0" "1" "2" "3" "4"...
Poorly named variables	Poor names	the patterns for poor variable names	"0" "1" "2" "3" "4"...
References development classes	Development classes	the names of development classes that should not be referenced	#CompilerError #EsAnd...
References global variables	Acceptable global variables	the names of global variables that are commonly used and should not be flagged as a problem	#Compiler #Processor...
Reimplements system method	System (low-level) selectors	the names of low-level system methods	#allClassVarNames...
Sends system method	System (low-level) selectors	the names of low-level system methods	#allClassVarNames...

Review	Property	Description	Default value
Sent but not implemented	Ignore capitalized literals	indicates if literals (symbols) beginning with an uppercase letter will not be considered as potential messages (This is useful if your application uses symbols for state and by convention your symbols start with an uppercase letter.)	true
	Scan entire image	indicates whether the entire image should be scanned instead of only the visible classes scanned	false
	Use packager directives	indicates whether the ENVY/Packager directives are taken into account (These directives can help ignore symbols that might be mistaken for potential selectors.)	true
Should call superclass	Methods	the names of methods that should call super	#initialize
Subclasses base class	Base classes	base classes	#AdditiveSequenceableCollec...
Too many consecutive concatenations	Maximum allowed consecutive concatenations	the maximum number of allowed concatenations	4
	Maximum allowed consecutive messages	the maximum number of allowed consecutive messages	4
Too many consecutive messages	Maximum allowed consecutive messages	the maximum number of allowed consecutive messages	4
Unreferenced class	Ignore superclasses	if the class has subclasses, it is ignored	False
Unsent method	Scan entire image	indicates whether the entire image should be scanned instead of only the visible classes scanned	False
	Use packager directives	indicates whether the ENVY/Packager directives are taken into account (These directives can help ignore symbols that might be mistaken for potential selectors.)	True
Unused arguments	Methods to ignore	the patterns for selectors whose arguments are often not used	"clientData.callData:"

Table 6 — Specific review properties.

4.13.3 Modifying Properties

To modify the default value of review properties:

- from a development browser, select **Tool** → **Review** and click **Settings...**
- from the Transcript, select **Tool Settings** → **Review...**

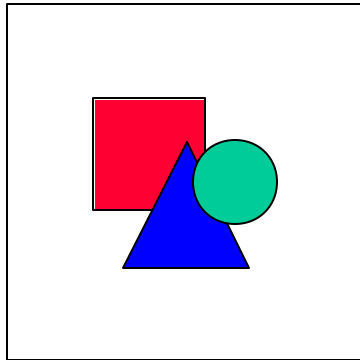



Figure 3 — Code Critic Advanced Settings dialog.

When the dialog opens, proceed as follows:

1. Click **Advanced...** to open the **Advanced Settings** (see Figure 3).
2. Select a review in the **Reviews** list.

The review's properties appear in the **Preferences** list.

 To access a review easily, you can sort the reviews in the **Reviews** list either alphabetically or by code element type, using **By Label** or **By Type**.

3. Select a property name in the **Preferences** list.

Its current value appears in the **Preference Value** text area.

4. Modify the text area to the desired value of the property according to the following table.

Property	Description
single-value property	must be Boolean, Integer, String, or Symbol, depending on the expected type of the property
multi-value property	elements are separated by spaces
values for class property	class names must be valid class names

Table 7 — Code Critic property values and their descriptions.

5. Click **Save Value** to save the new value of the property.
6. When you have made all modifications, click **OK** to accept the changes or **Cancel** to abort the changes.

4.14 Saving and Loading Your Settings

The **Code Critic Settings** dialog lets you save and load settings. To save the currently enabled reviews and their property values:

1. Open the **Code Critic Settings** dialog from any **Tool** → **Review** menu item or select **Tool Settings** → **Review...** from the Transcript.
2. Click **Save To File...**
3. Enter the name of the file to which to save the settings.

To load settings from a file:

1. Open the **Code Critic Settings** dialog from any **Tool** → **Review** menu item or select **Tool Settings** → **Review...** from the Transcript.
2. Click **Load From File...**

3. Enter the name of the file containing your settings. The loaded settings override any previous settings.

4.15 Using the Code Critic Results Browser

The **Code Critic Results Browser** lets you:

- view results
- view the source of the code element or the review description
- view all results or omit ignored results
- fix problems highlighted by the review
- save and load results from a file
- export the results as text or spreadsheet
- print the results

4.15.1 Layout of the Code Critic Results Browser

The following figure shows the **Code Critic Results Browser**.

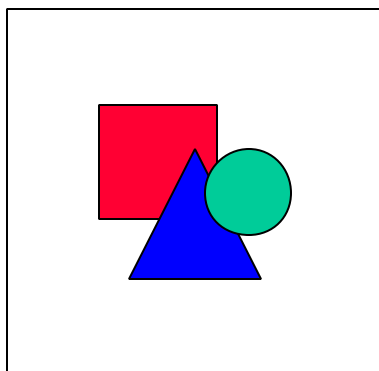


Figure 4— Code Critic Results Browser.

The following table describes the elements of the **Code Critic Results Browser**.

Element	Description
menu bar	provides access to browsing, results management, and viewing
Use Ignore Set...	ignores any results in the browser that match results in the specified file
Save Ignore Set...	saves the currently ignored results to a file
Applications Browser	launches or brings to the front the Applications Browser associated with the browser (You can use this browser to select the same code element as in the results browser.)
left list	displays the list of reviews for which problems were found

Element	Description
right list	displays specific problems for a review selected in the left list
Sort By Type	sorts reviews by their type (config map, application, class, method)
Sort By Severity	sorts reviews by their severity; severity 1 is the most severe
Code	shows the source code of the code element
Description	shows the description of the selected review
label	located directly above the source code, this area displays additional information about the problem
text area	displays the text of the code element, a summary of the results, or the description of a review
Next	advances to the next problem (You can also select the method again to advance to the next problem.)
Previous	returns to the previous problem

Table 8 — Elements of the Code Critic Results Browser.

4.15.2 Launching the Code Critic Results Browser

The **Code Critic Results Browser** lets you view the results of running Code Critic or view results that have been previously saved to a file.

To view the results of running Code Critic:

1. Select a code element from a development browser.
2. Select **Tool** → **Review**.
3. Run the review.

To view results saved to a file:

1. From the Transcript, select **Tool** → **Review...**
2. Enter a name of the results file to load.

4.15.3 Sorting by Type

To sort reviews according to the review type, select **Sort By Type**.

4.15.4 Sorting by Severity

To sort reviews according to problem severity, select **Sort By Severity**. Severity 1 is the most severe type of problem.

4.15.5 Viewing a Problem

Select a review from the left list and an item in the right list. The text area displays the problem and the status area displays any additional information about the problem.

To advance to the next problem, select the method again or click **Next**. To return to the previous, click **Previous**.

4.15.6 Ignoring Result

Some results are not relevant and you can ignore them in subsequent reviews.

To ignore specific review results:

1. In the right list, select the review results to be ignored.
2. Select **Result** → **Ignore/Restore**.

 This menu item toggles the ignore state of results.

Results that have been ignored can be made visible or hidden in the browser.

To show ignored results, select **Result** → **Show All Results**.

To hide ignored results, select **Result** → **Hide Ignored Results**.

 If you have selected **Show All Results**, ignored results appear with a minus sign (-) on the left to distinguish them from results that are not ignored.

4.15.7 Removing Results

You can remove specific results and entire review types from the set of results.

To remove specific review results:

1. In the right list, select the reviews to remove.
2. Select **Result** → **Remove From List**.

To remove all results of a given type:

1. In the left list, select a review type.
2. Select **Review** → **Remove From List**.

To remove all results of a specific review type and disable running this review type again:

1. In the left list, select a review type.
2. Select **Review** → **Turn Review Off**.

4.15.8 Refreshing Results

If you modify code elements in the **Code Critic Results Browser**, the selected code element is automatically refreshed; if the problem has been fixed, the review result disappears from the browser.

However, if you make modifications outside the **Code Critic Results Browser**, you may need to refresh results explicitly to reflect the current state of the code elements, rather than their state when Code Critic was run.

To refresh results:

1. In the right list, select the results to be refreshed.
2. Select **Result** → **Refresh**.

4.15.9 Saving and Loading Results

You can save to a file results collected during a code review and you can review them later.

To save results to a file:

1. Select **Review** → **Save...**
2. Enter the name of the file into which to save the results.

To load previously saved results into the browser:

1. Select **Review** → **Load...**
2. Enter the name of the results file from which to load. The loaded results override the current set of results.

4.15.10 Saving and Loading Ignore Sets

You can save ignored results to a file and use the file to ignore results in subsequent code reviews.

To save ignored results to a file:

1. Click **Save Ignore Set...**
2. Enter the name of the file into which to save the ignored results.

To ignore results previously saved to a file:

1. Click **Use Ignore Set...**
2. Enter the name of the file from which to load the results. The loaded ignore set overrides the current ignore set.

4.15.11 Reporting Results

Code Critic results can be printed, exported as text, or exported as spreadsheet format. You can customize the contents of a report to produce either an in-depth report or a quick summary.

You can customize which parts of the report you want included. The parts of a report comprise:

- header—user name, date and time, and comment
- list of reviews—the list of reviews that detected problems
- results—the list of results
- summary of results

4.15.12 Exporting Results to Text


To export results to a text file:

1. Select **Review** → **Export Report** → **As Text...**
2. Choose the sections to include (see Section 4.15.11) and click **OK**.
3. Enter the name of the text file to which to export the results.

4.15.13 Exporting to Spreadsheet

To export results to a spreadsheet file:

1. Select **Review** → **Export Report** → **As Spreadsheet...**
2. Choose the sections to include (see Section 4.15.11) and the type of column delimiter to use. Click **OK**.
3. Enter the name of the spreadsheet file to which to export the results.

 Each field in the export file is separated by a comma or a tab. You can modify the default setting in the **Report Setup** dialog that appears when you create a report.

4.15.14 Printing Results

To print results:

1. Select **Review** → **Print Report...**
2. Choose the sections to include (see Section 4.15.11) and click **OK**.

4.15.15 Adding a Comment to the Results

In some cases, it is useful to annotate your Code Critic results with a comment, especially if you are saving the results to examine later.

To add a comment to a code review:

1. Select **Review** → **Summary Info...**

The **Summary Info** dialog opens.

2. Enter a comment in the **Comment** text area.
3. Click **OK** to accept the comment or **Cancel** to abort.

4.15.16 Opening an Applications Browser

In some cases, you may want to switch quickly to a development browser, where you can make more complex modifications to the code. To facilitate the switch, the **Code Critic Results Browser** may have an associated **Applications Browser**.

When you select a result in the right list, the associated **Applications Browser** selects the code element corresponding to the result. You can open an associated **Applications Browser** by:

- clicking the **Applications Browser** button
- or
- double-clicking the selected review result in the right list

4.15.17 Opening Other Browsers

The **Code Critic Results Browser** provides the following options for opening development browsers:

- **Result** → **Browse Senders**
- **Result** → **Browse Implementors**
- **Result** → **Browse Messages**
- **Result** → **Browse Referenced Class...**
- **Result** → **Browse Class**
- **Result** → **Browse Hierarchy**

The above browsing functions operate on the code element of the selected review result. They function similarly to other development browsers.

4.15.18 Menus

The following sections describe the menus of the **Code Critic Results Browser**.

4.15.18.1 File Menu

This menu is the same as existing development browsers.

4.15.18.2 Edit Menu

This menu is the same as existing development browsers.

4.15.18.3 Review Menu

Save...

Saves the current results to a file.

Load...

Loads results from a file.

Summary Info...

Displays summary information (user's name, time and date, and comment) for the review results.

Export Report

Exports the results to a file.

As Text...

Exports the results to a text file.

As Spreadsheet...

Exports the results to a spreadsheet file; fields are separated by comma or tab.

Print Report...

Prints the results.

Remove From List

Removes all results of the selected review type.

Turn Review Off

Removes all results of the selected review type and turns off the review for subsequent code reviews.

Settings...

Opens the **Code Critic Settings** dialog.

4.15.18.4 Result Menu

Remove From List

Removes the selected review results.

Refresh

Refreshes the selected review results to reflect the current state of the code.

Browse Senders

Browses the senders of a selected method.

This is enabled only when the selected code element is a method.

Browse Implementors

Browses the implementors of a selected method.

This is enabled only when the selected code element is a method.

Browse Messages

Browses messages sent by the receiver.

Senders

Browses the senders of a message in a selected method.

Implementors

Browses the implementors of a message in a selected method.

This is enabled only when the selected code element is a method.

Browse Referenced Classes...

Browses a class that is referenced directly in a selected method.

This is enabled only when the selected code element is a method.

Browse Class

Browses the selected class.

This is enabled only when the selected code element is a class or method.

Browse Hierarchy

Browses the class hierarchy of a selected class.

This is enabled only when the selected code element is a class or method.

Delete

Deletes the selected method from the image.

This is enabled only when a menu measure result is selected.

Ignore/Restore

Ignores or restores the selected review results.

This toggles the state of reviewed results.

Hide Ignored Results

Does not show the ignored results.

Show All Results

Shows all the results including the ignored results.

Ignored results have a minus sign (-) beside them to distinguish them from results that are not ignored.

Change Public/Private

Changes the public or private status of the selected method.

4.15.18.5 Info Menu

This menu is the same as existing development browsers.

4.16 Advanced Concepts

Code Critic provides an extensible framework for writing your own project-specific reviews. This section describes how you can add your own code reviews.

4.16.1 Framework Configuration Maps

The **QA Code Critic** configuration map contains the hierarchy of reviews, the engine that runs the reviews (**CcFramework** application), and the user interface including the **Code Critic Results Browser** (**CcUserInterface** application).

To load the framework separately (without the UI) and run specific dolts directly, load only the **CcFramework** application.

4.16.2 Code Critic Framework Classes

You can add a review through subclassing. The location of your class in the hierarchy depends on what your review does and on what code element it operates on. Use the following table to decide which framework class to subclass.

Code element	Must be loaded	Subclass
configuration map	Y N	CtConfigurationMapMeasure
application	Y	CtLoadedApplicationMeasure
application	N	CtApplicationMeasure
subapplication	Y	CtLoadedSubApplicationMeasure
subapplication	N	CtSubApplicationMeasure
entire class	Y	CtEntireClassMeasure
class	Y	CtLoadedClassMeasure
class extension	Y	CtClassExtensionMeasure
class definition	Y	CtLoadedClassDefinitionMeasure
class definition	N	CtClassDefinitionMeasure
method	Y	CtLoadedMethodMeasure
method	N	CtMethodMeasure

Table 9 — Subclassing CtMeasure.

4.16.3 Implementing Your Review Subclass

The work of a review is performed in the method **#measure**. This method takes the code element as a parameter and returns nil (if the review found no problems) or an instance of *CcComplaint*.

4.16.3.1 Mandatory Methods

CtMeasure defines a set of methods categorized as *override mandatory*. These methods must be overridden in your subclass.

4.16.3.2 Optional Methods

CtMeasure defines a set of methods categorized as *override optional*. These methods will often be overridden in your subclass; however, failure to override them does not result in an error.

4.16.4 Example

The following example shows how to add a review that raises a warning when classes are too deep in the hierarchy.

CcDeepHierarchy class

```
CtLoadedClassDefinitionMeasure subclass: #CcDeepHierarchy
  instanceVariableNames: ""
  classVariableNames: ""
  poolDictionaries: ""
```

CcDeepHierarchy public class methods

```

defaultProperties
    "Answer the receiver's default properties.
    PARAMETERS
        None

    RETURN VALUE
        <OrderedCollection of SqaProperty>"

^super defaultProperties
    add:(
        SqaIntegerProperty new
            label: 'Allowed number of superclasses';
            name: #acceptableSuperclasses;
            comment: 'The maximum number of allowed super classes';
            value: 5;
            yourself);
    yourself

group
    "Answer the functional group the receiver is in.
    In addition to being grouped by the type of object they
    apply to (Application, Class, etc.) measures are also organized
    into functional groups. A measure belongs to one group.
    Group names are also used when generating the user interface.

    PARAMETERS
        None

    RETURN VALUE
        <String>"

    ^'Inheritance'

descriptionText
    "Answer the description text for the receiver.

    PARAMETERS
        None

    RETURN VALUE
        <String>"

    ^ ' Warn if a class is nested too deep in a class hierarchy.'

adviceText
    "Answer the advice text for the receiver.

    PARAMETERS
        None

```

RETURN VALUE

<String>"

^' A class that has too many superclasses may be difficult to maintain.

Review your class structure and check whether the class actually needs to inherit behavior from its superclasses.'

isReview

"Answer true if the receiver is a review otherwise answer false.

PARAMETERS

None

RETURN VALUE

<Boolean>"

^true

label

"Answer a single line label briefly describing the receiver.

PARAMETERS

None

RETURN VALUE

<String>"

^'Too deep in hierarchy'

loaded

"When the receiver has been loaded, execute the appropriate initialization.

PARAMETERS

None

RETURN VALUE

None"

CcDeepHierarchy resetProperties

CcDeepHierarchy public instance methods

measure: aClass

"Measure aClass for the number of superclasses.

Warn if the number of superclasses is higher than the #acceptableSuperclasses property value.

RETURN VALUE

<nil | CcComplaint>"

| propertyValue |

(propertyValue := self propertyValueAt: #acceptableSuperclasses) isNil
ifTrue:[^nil].

aClass allSuperclasses size > propertyValue

ifTrue:[

^self

defaultReviewOf: aClass

```
result: (  
  CcComplaint new  
    message: 'Too deep in hierarchy';  
    yourself)  
ifFalse:[^nil]
```

5 Code Metrics

5.1 Overview

Code Metrics lets you compute a set of static metrics for your code. It is integrated fully with the existing development browsers and provides an ideal environment for integrating software metrics directly into the development process.

Code Metrics provides an extensible set of metrics. A metric is a specific type of measure that executes over code elements and returns a numerical result. A metric has an upper and a lower threshold. Results between these thresholds are in range; other results are out of range and need to be examined in more detail. Out-of-range results do not necessarily indicate a problem. However, it is important to understand why results are out of range.

Code Metrics results are viewed using the **Code Metrics Results Browser**. You can modify code elements from this browser.

5.2 Who Should Use This Tool

Developers can use Code Metrics to:

- isolate areas of the system that are highly coupled
- identify and correct common problems

- focus on potential areas for detailed code inspections
- estimate the complexity of a component

Project managers can use Code Metrics to:

- determine whether the project is following the estimated effort
- improve project estimation skills
- check whether guidelines are followed consistently

Release engineers can use Code Metrics to:

- estimate component footprint
- identify areas that need more testing

5.3 Loading Code Metrics

Code Metrics is installed automatically as part of the complete installation of **ENVY/QA** (see Chapter 3).

If you want to use only Code Metrics:

1. Open the **Configuration Maps Browser**.
2. Load the most recent edition of the **QA Code Metrics** configuration map with required maps.

When Code Metrics is loaded:

- the system menu has new submenus:
 - **Tool → Metrics...**
 - **Tool Settings → Metrics...**
- newly opened development browsers have a **Tool → Metrics** menu

You are now ready to use Code Metrics.

5.4 Unloading Code Metrics

To unload Code Metrics:

1. Open the **Configurations Maps Browser**.
2. Unload the loaded edition of the **QA Code Metrics** configuration map.

✘ Before unloading Code Metrics, make sure that all **Code Metrics Results Browsers** are closed.

3. If Code Metrics is the only **ENVY/QA** tool loaded, you can also unload (in order) the following configuration maps:
 - (a) **QA Code Tools (CC/CM) Framework**
 - (b) **QA Framework**

5.5 Guided Tour

In this section, we guide you through the process of:

- creating a sample class
- running metrics
- browsing the results

5.5.1 Creating a Class to Measure

Create the following class and methods.

Person class

```
Object subclass: #Person
  instanceVariableNames: 'name age '
  classVariableNames: "
  poolDictionaries: "
```

Person public class methods

```
named: aString age: anInteger
  "Answer a new instance of the receiver whose name is aString
  and age is anInteger."

  ^self new
    name: aString;
    age: anInteger
```

Person public instance methods

```
age
  ^age

happyBirthday
  "The receiver has had a birthday.
  Display a Happy Birthday message, and increment his/her age."

  age := age + 1.
  System message: ('Happy ', self age printString, ' Birthday, ', self name, '!').

name
  "Answer the name (String) of the receiver."

  ^name
```

Person private instance methods

age: anInteger

"Set the age (Integer) of the receiver to anInteger."

age := anInteger

name: aString

"Set the name (String) of the receiver to aString."

name := aString

5.5.2 Running the Metrics

After you have added the *Person* class and its methods, run Code Metrics as follows:

1. Open a **Classes Browser** and select the *Person* class.
2. Select **Classes** → **Tool** → **Metrics** → **Class...**
3. When the **Code Metrics Options** dialog opens, click **OK**.

Code Metrics runs all metrics on the *Person* class.

5.5.3 Browsing the Results

After the metrics run, the **Code Metrics Results Browser** displays the results. To browse the results:

1. In the left list, select metric **Lines of code**.

The right list shows the results of that metric type. The text area shows summary information for all methods.

2. Select **Result** → **Hide In-Range Results**.

Results inside the normal thresholds are hidden, letting you focus on the problem results.

3. In the right list, select the method **Person>>#happyBirthday**.

Code Metrics displays the number of lines of code for this method.

5.6 Code Metrics and the Development Browsers

Code Metrics is integrated fully with the development browsers. You can measure code elements using **Tool** → **Metrics** in applicable development browsers.

Code Metrics measures your code's compliance to project guidelines and standard practices. You can measure the following types of code elements:

- method
- class
- class fragment
- class hierarchy
- application
- application hierarchy
- configuration map

5.6.1 Measuring Configuration Maps

To measure a configuration map:

1. Open the **Configuration Maps Browser**.
2. Select a configuration map edition.
3. Select **Editions** → **Tool** → **Metrics**.
4. In the **Code Metrics Options** dialog, click **OK** to start the measure.

5.6.2 Measuring Applications

You can measure applications from the following browsers:

- **Configuration Maps Browser**
- **Application Manager**
- **Applications Browser**
- **Classes Browser**
- **Class Hierarchy Browser**
- **Visual Age Organizer**

To measure an application:

1. Open one of the browsers mentioned above.
2. Select the applications to be reviewed.

3. Select a menu option:

Item to measure	Menu option
applications	Applications → Tool → Metrics → Application...
applications and their loaded subapplications	Applications → Tool → Metrics → Include All Subapplications...

Table 10 — Measuring applications.

4. In the **Code Metrics Options** dialog, click **OK** to start the measure.

5.6.3 Measuring Classes

You can measure classes from the following browsers:

- **Application Manager**
- **Application Browser**
- **Applications Browser**
- **Classes Browser**
- **Class Hierarchy Browser**
- **Visual Age Organizer**

To measure a class:

1. Open one of the browsers mentioned above.
2. Select the classes to be reviewed.
3. Select a menu option:

Item to measure	Menu option
classes	Classes → Tool → Metrics → Class...
classes and all their subclasses	Classes → Tool → Metrics → Include All Subclasses...
class fragments	Classes → Tool → Metrics → Class Fragment...

Table 11 — Measuring classes.

4. In the **Code Metrics Options** dialog, click **OK** to start the measure.

5.6.4 Measuring Methods

You can measure methods from the following browsers:

- **Application Browser**
- **Applications Browser**
- **Classes Browser**
- **Class Browser**

- **Class Hierarchy Browser**
- **Methods Browser**

To measure a method:

1. Open one of the browsers mentioned above.
2. Select the methods to be reviewed.
3. Select **Methods** → **Tool** → **Metrics**.
4. In the **Code Metrics Options** dialog, click **OK** to start the measure.

5.7 Code Metrics Options Dialog

When you select a **Tool** → **Metric** menu option, the **Code Metrics Options** dialog opens. From this dialog you can run either all metrics (by selecting **All**), or a specific set of metrics (by selecting **Settings...**).

Click **OK** to start the review.

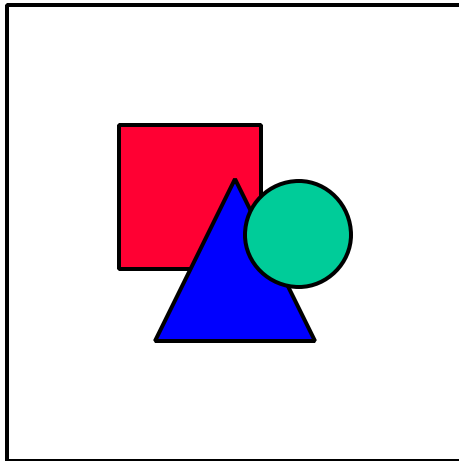


Figure 5 — Code Metrics Options dialog.

5.8 Selecting Which Metrics to Run

When you are familiar with the different types of metrics, you may find that some metrics are more useful than others at different stages of development. In such cases, you may want to run only some of the metrics.

You can select specific metrics in the **Code Metrics Settings** dialog shown below.

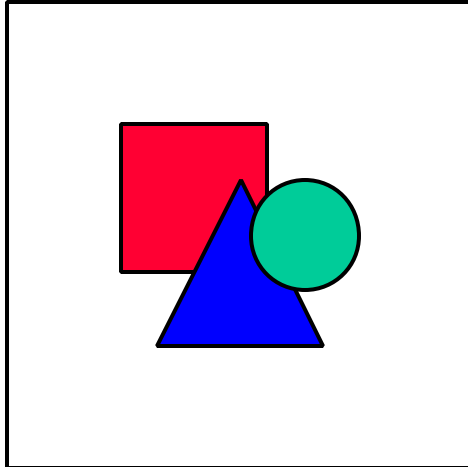


Figure 6 — Code Metrics Settings dialog.

The **Code Metrics Settings** dialog groups metrics into the categories shown in the following table.

Category	Explanation
Complexity	overall complexity of a code element—such as, cyclomatic complexity and Lorenz method complexity
Coupling	coupling and dependencies between code elements—such as, class coupling and the number of dependent applications
Decomposition	components of a code element—such as, the defined or extended classes of an application
Inheritance	class inheritance—such as, the total number of methods and depth of hierarchy
Interface	public interface of a class—such as, the ratio of public to private methods
Size	size of code elements—such as, the number of statements in a method and the memory size of a class

Table 12 — Metrics groupings.

To select specific metrics:

1. In the **Code Metrics Options** dialog, select **Settings...**, or from the Transcript, select **Tool Settings** → **Metrics...**

2. Select a category in the left list.

The list on the right shows check boxes for each metric in that category.

3. Do one of the following:

Desired result	Action
enable all metrics in a category	click All
disable all metrics in a category	click None
enable or disable specific metrics	click individual check boxes
reset to their default values all metrics in all categories	click Reset All

Table 13 — Enabling metrics.

4. Click **OK** to accept the selected metrics or **Cancel** to abort the selection.

5.9 Application Metrics

This section describes metrics that you can run on applications.

5.9.1 All Defined Classes

Measures the number of defined classes in an application and all its subapplications.

Advice

This metric helps measure the work required to develop an application and its subapplications.

5.9.2 All Dependent Applications

Measures the total number of applications that are dependent on the application.

Advice

This metric helps evaluate the dependence of other applications on the application.

5.9.3 All Extended Classes

Measures the number of extended classes in an application and all its subapplications.

Advice

This metric helps measure the work required to develop an application and its subapplications, as well as its dependence on its prerequisites.

5.9.4 All Prerequisites

Measures the total number of prerequisites of the application.

Advice

This metric helps evaluate an application's dependence on other applications.

5.9.5 All Subapplications

Measures the total number of subapplications of an application.

Advice

This metric helps evaluate the decomposition of an application.

5.9.6 Defined Classes

Measures the number of defined classes in an application.

Advice

This metric helps measure the work required to develop an application. It also helps evaluate the size of an application.

5.9.7 Dependent Applications

Measures the number of applications that have the application as an immediate prerequisite.

Advice

This metric helps evaluate the dependence of other applications on the application.

5.9.8 Extended Classes

Measures the number of extended classes in an application.

Advice

This metric helps measure the work required to develop an application.

5.9.9 Memory Size (Including Subapplications)

Measures the amount of memory used by the application and all its subapplications. This is the sum of the memory size of:

- the application
- the subapplications of the application
- the classes contained in the application
- the classes contained in each subapplication of the application

Advice

This metric helps measure the work required to develop an application.

5.9.10 Memory Size for Applications

Measures the amount of memory used by an application. This is the sum of the memory size of:

- the application
- the classes contained in the application

Advice

This metric helps measure the work required to develop an application.

5.9.11 Prerequisites

Measures the number of immediate prerequisites of an application.

Advice

This metric helps evaluate an application's dependence on other applications.

5.9.12 Subapplications

Measures the number of immediate subapplications of an application.

Advice

This metric helps evaluate the decomposition of an application.

5.10 Class Metrics

This section described metrics that you can run on classes.

5.10.1 Accessors

Measures the number of accessor methods of a class. An accessor method contains only a get or set operation on a state variable.

Advice

Typically there are at most two accessors per state variable. A larger number of accessors may indicate duplicate behavior. A smaller number of accessors may indicate a large number of direct accesses, which in turn can cause increased maintenance or limit the ability of a subclass to extend or refine the class's behavior.

5.10.2 All Class Methods

Measures the number of class methods defined by a class and its superclasses.

Advice

This metric helps evaluate the complexity of the class. The number of class methods is usually small compared to the number of instance methods. A large number of class methods may indicate that many services are handled by the class instead of by individual instances.

5.10.3 All Instance Methods

Measures the number of instance methods defined by a class and its superclasses.

Advice

This metric helps evaluate the complexity of the class's instances. Large classes sometimes attempt too much work themselves, instead of transferring responsibility to where it belongs. A large number of instance methods may indicate an overloaded class.

5.10.4 All Instance Variables

Measures the number of instance variables declared by a class (including those defined by its superclasses).

Advice

This metric helps evaluate the size of an instance of the class.

5.10.5 All Subclasses

Measures the total number of all subclasses of a class.

Advice

This metric helps evaluate the complexity of a class's hierarchy. If a class has a large number of subclasses, the methods that are inherited may require extra testing.

5.10.6 Class Coupling

Measures the number of classes with which the class is coupled. Class A is coupled with class B if A calls methods implemented in B.

Advice

A large coupling factor may indicate a design problem. A lower coupling indicates a more independent object, which in turn means the object may be easier to use.

5.10.7 Class Methods

Measures the number of class methods defined by a class.

Advice

This metric helps evaluate the complexity of the class. The number of class methods is usually small compared to the number of instance methods. A large number of class methods may indicate many services are handled by the class instead of by individual instances. The number of new methods decreases when you move down the inheritance tree. Therefore, the deeper the class is in the class hierarchy, the lower should be the ratio of class methods to all class methods. If this is not the case, there may be design problems.

5.10.8 Class Response

Measures the number of messages sent by methods of a class.

Advice

A large response factor indicates a larger set of methods can be sent in response to communications with the object. That is, when the object receives a message, it has the potential to call many other methods in response to the received message. This may in turn indicate an overly complex class, requiring more testing.

5.10.9 Class Variables

Measures the number of class variables declared by a class.

Advice

A large number of class variables may indicate that the class does too much work, instead of delegating work to other classes.

5.10.10 Cyclomatic Complexity

Measures the cyclomatic complexity of a class. Cyclomatic complexity is defined by the formula:

$$\text{sum-over-methods}((\text{number of exit points}) - 1) + 2$$

Advice

A large value may indicate complex code with too much branching.

5.10.11 Depth of Hierarchy

Measures the depth of the class hierarchy of a class. *Object* has a depth of 0.

Advice

This metric helps evaluate the complexity of a class hierarchy. A large

depth may indicate a complex inheritance tree and a potential design problem.

5.10.12 Direct Variable Accesses

Measures the number of methods that reference an instance, class, or class instance variable (excluding basic accessors).

Advice

This metric helps determine how variables are used in a class. Accessors can help preserve the encapsulation principle of object-oriented programming. Direct accesses can also cause increased maintenance or limit the ability of a subclass to extend or refine the behavior of the class.

5.10.13 Global/Pool References

Measures the number of references to pool variables or global variables from a class's methods.

Advice

If a class has too many references, the class may rely too much on shared information.

5.10.14 Instance Methods

Measures the number of instance methods defined by a class.

Advice

This metric helps evaluate the complexity of the class's instances. Large classes sometimes attempt too much work themselves instead of transferring responsibility to where it belongs. A large number of instance methods may indicate an overloaded class. The number of new methods typically decreases when you move down the inheritance tree. Therefore, the deeper the class is in the class hierarchy, the lower the ratio. If this is not the case, there may be design problems.

5.10.15 Memory Size for Classes

Measures the amount of memory (in bytes) used by a class and its methods.

Advice

This metric helps measure the work required to develop a class.

5.10.16 New Methods

Measures the number of new methods defined by a class. New methods are those not inherited from a superclass.

Advice

The number of new methods should decrease when you move down the class hierarchy.

5.10.17 Pool Dictionaries

Measures the number of pool dictionaries declared by a class.

Advice

A class with too many pool dictionaries may be too tightly coupled to other classes.

5.10.18 Ratio API/Internal

Measures the ratio of a class's API methods to internal methods. This is based on a method's category and not on its public or private status.

Advice

A low ratio (many internal methods and few API methods) may indicate that the class is simple to use but its internal processing may be complex (hard to maintain). A high ratio (many API methods and few internal methods) may indicate that the class is doing too much work, because API methods represent services that other classes can use.

5.10.19 Ratio Public/Private

Measures the ratio of a class's public methods to private methods.

Advice

A low ratio (few public methods and many private methods) may indicate that the class is simple to use but its internal processing may be complex (hard to maintain). A high ratio (many public methods and few private methods) may indicate that the class is doing too much work, because public methods represent services that other classes can use.

5.10.20 Refined Methods

Measures the number of refined methods defined by a class. A refined method is a method that overrides a subclass implementation.

Advice

A large number of refined methods may indicate a design problem. As a specialization of its superclass, a subclass should primarily extend the services of its superclasses. Therefore, a large number of refined methods should occur only when the superclass is an abstract class.

5.10.21 Specialization Index

Measures the ratio:

refined instance methods * superclasses / all instance methods

Advice

This metric helps evaluate the quality of a subclass. A good subclass is usually an extension of the capabilities of its superclasses. An undesirable subclass is one that overrides the methods of its superclasses if the superclasses are not abstract classes. A high ratio may indicate a poor subclass and design problems.

5.10.22 Subclasses

Measures the number of immediate subclasses of a class.

Advice

This metric helps evaluate the complexity of a class hierarchy. If a class has a large number of subclasses, the methods that are inherited may require extra testing.

5.11 Method Metrics

This section describes metrics that you can run on methods.

5.11.1 Lines of Code

Measures the number of lines of code (LOC) in a method. Comments at the top of the method and blank lines are ignored.

Advice

Larger values may indicate overly complex methods. You should refactor the method.

5.11.2 Lorenz Complexity

Measures the Lorenz complexity² of a method. The Lorenz complexity adds weighted attributes of each method.

Advice

A larger value may indicate an overly complex method.

5.11.3 Memory Size for Methods

Measures the amount of memory used by a method.

Advice

This metric helps measure the work required to develop a method.

² Mark Lorenz and Jeff Kidd, *Object-Oriented Software Metrics*, Prentice Hall, 1994.

5.11.4 Method Density

Measures the density of a method. Method density is the number of statements (as defined by the compiler) divided by the lines of code (LOC). Comments at the top of the method and blank lines are ignored.

Advice

Large values may indicate less-readable methods.

5.11.5 Statements

Measures the number of statements. A statement is defined as any sequence of expressions that ends in a period.

Advice

Larger values may indicate overly complex methods. You should refactor the method.

5.12 Customizing the Metrics

Each metric has properties that affect the way the metric checks code elements. Some properties are common to all metrics, while others are specific to a metric.

5.12.1 Common Properties

All metrics have the properties *lower threshold* and *upper threshold*. These properties define the acceptable range of values for the metric. You should examine carefully results that fall outside the acceptable range. You can modify the default thresholds to match your project's guidelines.

The following table summarizes the default thresholds for each metric.

Metric	Default lower threshold	Default upper threshold
Accessors	0	12
All class methods	0	45
All defined classes	1	40
All dependent applications	0	20
All extended classes	0	26
All instance methods	0	250
All instance variables	0	8
All prerequisites	1	20
All subapplications	0	4
All subclasses	0	12
Class coupling	0	15

Metric	Default lower threshold	Default upper threshold
Class methods	0	6
Class response	0	60
Class variables	0	2
Cyclomatic complexity	0	6
Defined classes	1	20
Dependent applications	0	4
Depth of hierarchy	1	4
Direct variable accesses	0	0
Extended classes	0	15
Global/Pool references	0	7
Instance methods	0	25
Lines of code	1	10
Lorenz complexity	0	65
Memory size (incl. subapps)	0	75000
Memory size for applications	0	5500
Memory size for classes	0	5500
Memory size for methods	68	400
Method Density	1	8
New methods	0	15
Pool dictionaries	0	2
Prerequisites	1	3
Ratio API/internal	0	2
Ratio public/private	0	2
Refined methods	0	5
Specialization index	0	0.1
Statements	1	16
Subapplications	0	4
Subclasses	0	3

Table 14 — Default lower and upper thresholds.

5.12.2 Specific Properties

Some metrics have properties specific to that metric. The following table summarizes the properties for those metrics and the default value for the property. You can customize any of these property values.

Metric	Property	Description	Default value
Class coupling	Classes to ignore	classes that should not be considered as coupled	#AdditiveSequenceableCollec...
Lorenz complexity	API calls weight	the weight given to the number of API calls (API calls are calls to platform functions.)	5.0
	Assignments weight	the weight given to the number of assignments	0.5
	Binary expressions weight	the weight given to the number of binary expressions	2.0
	Keyword expressions weight	the weight given to the number of keyword expressions	3.0
	Nested expressions weight	the weight given to the number of nested expressions	0.5
	Parameters weight	the weight given to the number of method arguments	0.3
	Primitive calls weight	the weight given to the number of primitive calls (Primitive calls are methods that call virtual-machine primitives.)	7.0
	Temporaries weight	the weight given to the number of temporary variables	0.5
Ratio API/internal	Unary expressions weight	the weight given to the number of unary expressions	1.0
	API categories	the patterns of API categories	**API**
	Internal categories	the patterns of internal categories	**Internal**

Table 15 — Specific metric properties.

5.12.3 Modifying Properties

To modify the default value of metric properties:

- from a development browser, select **Tool** → **Metrics** and click **Settings...**

or

- from the Transcript, select **Tool Settings** → **Metrics...**

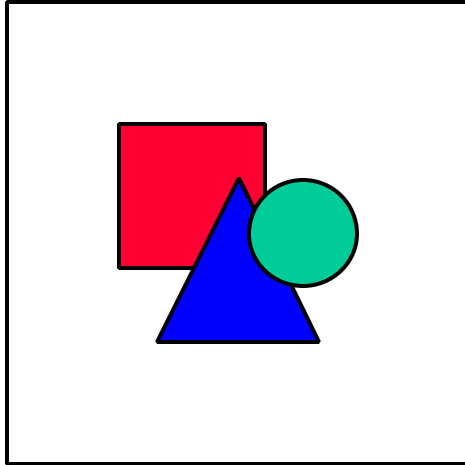



Figure 7 — Code Metrics Advanced Settings dialog.

When the dialog opens, proceed as follows:

1. Click **Advanced...** to open the **Advanced Settings** dialog (see Figure 7).
2. Select a metric in the **Metrics** list.

The metric's properties appear in the **Preferences** list.

 To access a metric easily, you can sort the metrics in the **Metrics** list either alphabetically or by code element type, using **By Label** or **By Type**.

3. Select a property name in the **Preferences** list.

Its current value appears in the **Preference Value** text area.

4. Modify the text area to the desired value of the property, according to the following table.

Property	Description
single-value property	must be Boolean, Integer, String, or Symbol, depending on the expected type of the property
multi-value property	elements are separated by spaces
values for class property	class names must be valid class names

Table 16 — Code Metrics property values and their descriptions.

5. Click **Save Value** to save the new value of the property.
6. When you have made all modifications, click **OK** to accept the changes or **Cancel** to abort the changes.

5.13 Saving and Loading Your Settings

The **Code Metrics Settings** dialog lets you save and load settings. To save the currently enabled metrics and their property values:

1. Open the **Code Metrics Settings** dialog from any **Tool** → **Metrics** menu item, or select **Tool Settings** → **Metrics...** from the Transcript.
2. Click **Save To File...**
3. Enter the name of the file to which to save the settings.

To load settings from a file:

1. Open the **Code Metrics Settings Dialog** from any **Tool** → **Metrics** menu item or select **Tool Settings** → **Metrics...** from the Transcript.
2. Click **Load From File...**
3. Enter the name of the file containing your settings. The loaded settings override any previous settings.

5.14 Using the Code Metrics Results Browser

The **Code Metrics Results Browser** lets you:

- view results
- view the source of the code element or the metric description
- view all results or focus on those out of range
- fix problems highlighted by the review
- export the results as text or spreadsheet
- print the results

5.14.1 Layout of the Code Metrics Results Browser

The following figure shows the **Code Metrics Results Browser**.

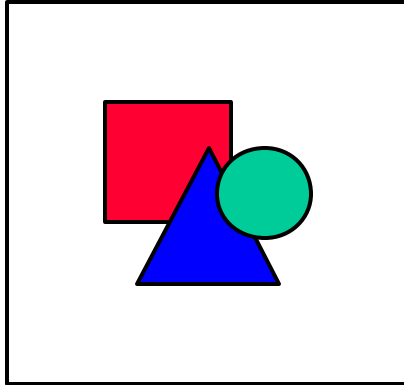


Figure 8 — Code Metrics Results Browser.

The following table describes the elements of the **Code Metrics Results Browser**.

Element	Description
menu bar	provides access to browsing, results management, and viewing
Applications Browser	launches or brings to the front the Applications Browser associated with the browser (You can use the Applications Browser to select the same code element as in the results browser.)
left list	display by metric mode—displays the metric labels that have some results display by object mode—displays the hierarchy of code elements that have results
right list	displays specific metric results
Result	shows the source code of the code element
Description	shows the description of the selected metric type
label	located directly above the source code, this area displays additional information about the result
text area	displays the text of the code element, a summary of the results, or the description of a metric

Table 17 — Elements of the Code Metrics Results Browser.

5.14.2 Launching the Code Metrics Results Browser

The **Code Metrics Results Browser** lets you view the results of running Code Metrics or view results that have been previously saved to a file.

To view the results of running Code Metrics:

1. Select a code element from a development browser.
2. Select **Tool** → **Metrics**.
3. Run the metrics.

To view results saved to a file:

1. From the Transcript, select **Tool** → **Metrics...**
2. Enter the name of a results file to load.

5.14.3 Hiding In-Range Results

To focus on abnormal results, you can hide results that are within the normal threshold values.

To hide in-range results, select **Result** → **Hide In-Range Results**.

To show all results, select **Result** → **Show All Results**.

Metrics results above the upper threshold are displayed with a plus sign (+); metrics results below the lower threshold are displayed with a minus sign (-).

5.14.4 Removing Results

You can remove specific results and entire metric types from the set of results.

To remove specific metrics results:

1. In the right list, select the results to remove.
2. Select **Result** → **Remove From List**.

To remove all results of a given type:

1. In the left list, select a metric type.
2. Select **Metric** → **Remove From List**.

To remove all results of a specific metric type and disable running this metric type again:

1. In the left list, select a metric type.
2. Select **Metric** → **Turn Metric Off**.

5.14.5 Refreshing Results

If you modify code elements in the **Code Metrics Results Browser**, the selected code element is refreshed automatically. If the problem has been fixed, the metrics result disappears from the browser.

However, if you make modifications outside the **Code Metrics Results Browser**, you may need to refresh results explicitly to reflect the current state of the code elements, rather than their state when the metrics were collected.

To refresh results:

1. In the right list, select the results to be refreshed.

2. Select **Result** → **Refresh**.

5.14.6 Saving and Loading Results

You can save to a file results collected by Code Metrics and review the results later.

To save results to a file:

1. Select **Metric** → **Save...**
2. Enter the name of the file to which to save the results.

To load previously saved results into the browser:

1. Select **Metric** → **Load...**
2. Enter the name of the results file from which to load. The loaded results override the current set of results.

5.14.7 Reporting Results

Code Metrics results can be printed, exported as text, or exported as spreadsheet format. You can customize the contents of a report to produce either an in-depth report or a quick summary.

You can customize which parts of the report you want included. The parts of a report comprise:

- header—user name, date and time, and comment
- list of metrics—the list of all metrics that collected results
- results—the list of results
- summary of results

5.14.8 Exporting Reports to Text

To export results to a text file:

1. Select **Metric** → **Export Report** → **As Text...**
2. Select the sections to include (see Section 5.14.7) and click **OK**.
3. Enter the name of the file to which to export the results.

5.14.9 Exporting to Spreadsheet

To export results to a spreadsheet file:

1. Select **Metric** → **Export Report** → **As Spreadsheet...**
2. Select the sections to include (see Section 5.14.7) and the type of column delimiter to use. Click **OK**.

3. Enter the name of the file to which to export the results.

☞ Each field in the export file is separated by a comma or a tab. You can modify the default setting in the **Report Setup** dialog that appears when you create a report.

5.14.10 Printing Results

To print results:

1. Select **Metric** → **Print Report...**
2. Select the sections to include (see Section 5.14.7) and click **OK**.

5.14.11 Adding a Comment to the Results

In some cases, it is useful to annotate your Code Metrics results with a comment, especially if you are saving the results to examine later.

To add a comment to your Code Metrics results:

1. Select **Metric** → **Summary Info...**

The **Summary Info** dialog opens.

2. Enter a comment in the **Comment** text area.
3. Click **OK** to accept the comment or **Cancel** to abort.

5.14.12 Opening an Applications Browser

In some cases you may want to switch quickly to a development browser, where you can make more complex modifications to the code. To facilitate the switch, the **Code Metrics Results Browser** may have an associated **Applications Browser**.

When you select a result in the right list, the associated **Applications Browser** selects the code element corresponding to the result. You can open an associated **Applications Browser** by:

- clicking the **Applications Browser** button
- or
- double-clicking the selected metrics result in the right list

5.14.13 Opening Other Browsers

The **Code Metrics Results Browser** provides the following options for opening development browsers:

- **Result** → **Browse Senders**
- **Result** → **Browse Implementors**

- **Result** → **Browse Messages**
- **Result** → **Browse Referenced Class...**
- **Result** → **Browse Class**
- **Result** → **Browse Hierarchy**

The above browsing functions operate on the code element of the selected metric result. They function similarly to other development browsers.

5.14.14 Menus

This section describes the menus of the **Code Critic Results Browser**.

5.14.14.1 File Menu

This menu is the same as existing development browsers.

5.14.14.2 Edit Menu

This menu is the same as existing development browsers.

5.14.14.3 Metric Menu

Save...

Saves the current results to a file.

Load...

Loads results from a file.

Summary Info...

Displays summary information (user's name, time and date, and comment) for the metrics results.

Export Report

Exports the results to a file.

As Text...

Exports the results to a text file.

As Spreadsheet...

Exports the results to a spreadsheet file; fields are separated by comma or tab.

Print Report...

Prints the results.

Remove From List

Removes all results of the selected metric type.

Turn Metric Off

Removes all results of the selected metric type and turns off the metric for subsequent code metrics.

Settings...

Opens the **Code Critic Settings** dialog.

5.14.14.4 Result Menu**Remove From List**

Removes the selected metric results.

Refresh

Refreshes the selected metric results to reflect the current state of the code.

Browse Senders

Browses the senders of a selected method.

This is enabled only when the selected code element is a method.

Browse Implementors

Browses the implementors of a selected method.

This is enabled only when the selected code element is a method.

Browse Messages

Browses messages sent by the receiver.

Senders

Browses the senders of a message in a selected method.

Implementors

Browses the implementors of a message in a selected method.

This is enabled only when the selected code element is a method.

Browse References Classes...

Browses a class that is referenced directly in a selected method.

This is enabled only when the selected code element is a method.

Browse Class

Browses the selected class.

This is enabled only when the selected code element is a class or method.

Browse Hierarchy

Browses the class hierarchy of a selected class.

This is enabled only when the selected code element is a class or method.

Delete

Deletes the selected method from the image.

This is enabled only when a menu measure result is selected.

Hide In-Range results

Does not show the results that are within the normal range.

Show All Results

Shows all the results.

Results below the lower threshold have a minus sign (-) beside them. Results above the upper threshold have a plus sign (+) beside them.

5.14.14.5 Info Menu

This menu is the same as existing development browsers.

5.15 Advanced Concepts

Code Metrics provides an extensible framework for writing your own project-specific metrics. This section describes how you can add your own code metrics.

5.15.1 Framework Configuration Maps

The **QA Code Metrics** configuration map contains the hierarchy of metrics, the engine that runs them (**CmFramework** application), and the user interface including the **Code Metrics Results Browser** (**CmUserInterface** application).

To load the framework separately (without the UI) and run specific dolts directly, load only the **CmFramework** application.

5.15.2 Code Metrics Framework Classes

You add a metric through subclassing. The location of your class in the hierarchy depends on what your metric does and on what code element it operates on. Use the following table to decide which framework class to subclass.

Code element	Must be loaded	Subclass
configuration map	Y N	CtConfigurationMapMeasure
application	Y	CtLoadedApplicationMeasure
application	N	CtApplicationMeasure
subapplication	Y	CtLoadedSubApplicationMeasure
subapplication	N	CtSubApplicationMeasure
entire class	Y	CtEntireClassMeasure
class	Y	CtLoadedClassMeasure
class extension	Y	CtClassExtensionMeasure
class definition	Y	CtLoadedClassDefinitionMeasure

Code element	Must be loaded	Subclass
class definition	N	CtClassDefinitionMeasure
method	Y	CtLoadedMethodMeasure
method	N	CtMethodMeasure

Table 18 — Subclassing *CtMeasure*.

5.15.3 Implementing Your Metric Subclass

The work of a metric is performed in the method **#measure:**. This method takes the code element as a parameter and returns a *CtMeasurement* with a numerical value.

5.15.3.1 Mandatory Methods

CtMeasure defines a set of methods categorized as *override mandatory*. These methods must be overridden in your subclass.

5.15.3.2 Optional Methods

CtMeasure defines a set of methods categorized as *override optional*. These methods will often be overridden in your subclass; however, failure to override them does not result in an error.

5.15.4 Example

The following example shows how to add a metric that measures the number of applications in a configuration map edition.

CmApplications class

CtConfigurationMapMeasure subclass: #CmApplications

instanceVariableNames: "

classVariableNames: "

poolDictionaries: "

CmApplications public class methods

defaultUpperThreshold

"Answer the default upper threshold for the receiver.

The thresholds are affected by many factors (ex. prototype, first release, UI...) and the user has to tailor the thresholds to his own needs and experience.

PARAMETERS

None

RETURN VALUE

<Number>"

^10

group

"Answer the functional group the receiver is in.

In addition to being grouped by the type of object they apply to (Application, Class etc.) measures are also organized into functional groups. A measure belongs to one group. Group names are also used when generating the user interface.

PARAMETERS

None

RETURN VALUE

<String>"

^'Decomposition'

descriptionText

"Answer the information text about the receiver.

PARAMETERS

None

RETURN VALUE

<String>"

^' Measure the number of applications contained in a configuration map'

adviceText

"Answer the advice text about the receiver..

PARAMETERS

None

RETURN VALUE

<String>"

^' Helps estimate the packaging effort.

isMetric

"Answer true if the receiver is a metric
otherwise answer false.

PARAMETERS

None

RETURN VALUE

<Boolean>"

^true

label

"Answer a single line label briefly describing the receiver.

PARAMETERS

None

RETURN VALUE

<String>"

^'Applications'

CmApplications public instance methods

measure: aConfigurationMap

"Measure aConfigurationMap for the number of applications
it contains.

PARAMETERS

aConfigurationMap <EmConfigurationMap>

RETURN VALUE

<CtMeasurement>"

^self

defaultMeasurementOf: aConfigurationMap
result:aConfigurationMap applicationNames size

6 Code Coverage

6.1 Overview

Code Coverage helps you determine whether test cases that were developed to test your applications provide you with complete test coverage. It is integrated fully with the existing development browsers and lets you evaluate your test cases as you develop the software.

Code Coverage lets you focus on testing your applications by presenting a view of what is left to be tested of your application. It does this by hiding components as they become tested, showing the remaining untested components.

6.2 Who Should Use This Tool

Developers can use Code Coverage to:

- design test cases that maximize the test coverage of their applications
- find deficiencies in application test suites
- set up reusable test coverage configurations

Project managers and release engineers can use Code Coverage to:

- verify the amount of coverage obtained by regression test suites

- produce summary reports detailing test coverage statistics

6.3 Loading Code Coverage

Code Coverage is installed automatically as part of the complete installation of **ENVY/QA** (see Chapter 3).

If you want to use only Code Coverage:

1. Open the **Configuration Maps Browser**.
2. Load the most recent edition of the **QA Code Coverage** configuration map with required maps.

When Code Coverage is loaded:

- the system menu has a new submenu **Tool → Coverage**
- newly opened development browsers have a **Tool → Coverage** menu

You are now ready to use Code Coverage.

6.4 Unloading Code Coverage

To unload Code Coverage:

1. Open the **Configurations Maps Browser**.
2. Unload the loaded edition of the **QA Code Coverage** configuration map.

✘ Before unloading Code Coverage, make sure that all **Code Coverage Browsers** are closed.

3. If Code Coverage is the only **ENVY/QA** tool loaded, you can also unload **QA Framework**.

6.5 Guided Tour

In this section, we guide you through the process of:

- creating an application that you will *watch* using Code Coverage
- selecting the application to watch
- ignoring methods that are not to be tested

6.5.1 Creating an Application

Create the application **SampleApplication** and add the following class and methods.

Person class

Object subclass: #Person
instanceVariableNames: 'name age '
classVariableNames: "
poolDictionaries: "

Person public class methods

named: aString age: anInteger
"Answer a new instance of the receiver whose name is aString
and age is anInteger."
^self new
name: aString;
age: anInteger

initialize
"Initialize the receiver. This should only be done when it is first loaded
into the image."
Transcript cr; show: 'Initializing the receiver.'

Person public instance methods

happyBirthday
"The receiver has had a birthday.
Display a Happy Birthday message."
System message: ('Happy ', self age printString, ' Birthday, ', self name, '!').

name
"Answer the name (String) of the receiver."

^name

age
"Answer the age (Integer) of the receiver."
^age

Person private instance methods

age: anInteger
"Set the age (Integer) of the receiver to anInteger."
age := anInteger

name: aString
"Set the name (String) of the receiver to aString."
name := aString

6.5.2 Selecting the Applications to Watch

After you have added the application **SampleApplication**, run Code Coverage as follows:

1. Open an **Application Manager** and select the application **SampleApplication**.
2. Select **Applications** → **Tool** → **Coverage** → **Application**.

The **Code Coverage Browser** opens, containing your application. At the top of the browser, the status line shows the number of remaining untested methods.

3. Select your application (**SampleApplication**).

Classes and methods that remain to be tested are shown in the right two lists.³

6.5.3 Watching Your Application

In the **Code Coverage Browser**, click **watch**. Your application is now being watched, and Code Coverage detects automatically which methods you execute. Proceed as follows:

1. From the Transcript, evaluate:
 Person named: 'Joe' age: 30.
2. Return to the **Code Coverage Browser** and select your application.

The **#named:age:** method has been executed (tested) and automatically disappears from the **Code Coverage Browser**.

3. To see methods hidden by Code Coverage, select **Methods** → **Show All**. (This is the default. The toggle for this is **Methods** → **Hide Ignored**.)

6.5.4 Ignoring Methods

In some situations, there are methods that you do not intend your test suites to execute. To ignore these methods:

1. Select the class method **#initialize**.
2. Select **Methods** → **Ignore/Restore**.

³ On VisualAge for Smalltalk Professional, accessor and primitive methods are not shown. These methods are handled specially by the virtual machine and cannot be monitored by Code Coverage.

The method disappears and the status line at the top of the **Code Coverage Browser** is updated.

6.6 Code Coverage and the Development Browsers

Code Coverage is integrated fully with the development browsers. You can run Code Coverage using **Tool** → **Coverage** in applicable development browsers and in the Transcript.

6.6.1 Coverage on Configuration Maps

To run Code Coverage on configuration maps:

1. Open the **Configuration Maps Browser**.
2. Select a configuration map edition.
3. Select **Editions** → **Tools** → **Coverage**.

6.6.2 Coverage on Applications

You can run Code Coverage on applications from the following browsers:

- **Configuration Maps Browser**
- **Application Manager**
- **Applications Browser**
- **Classes Browser**
- **Class Hierarchy Browser**
- **Visual Age Organizer**

To run Code Coverage on an application:

1. Open one of the browsers mentioned above.
2. Select the applications to be watched.
3. Select a menu option:

Item to watch	Menu option
applications	Applications → Tool → Coverage → Application...
applications and their loaded subapplications	Applications → Tool → Coverage → Include Subapplications... ⁴

Table 19 — Watching applications.

⁴ In the **Configuration Maps Browser**, select **Tools** → **Coverage**.

6.7 Using the Code Coverage Browser

The **Code Coverage Browser** lets you:

- add components to be observed for coverage testing
- ignore components to focus coverage testing
- start, stop, and pause coverage testing
- view the percentage completion of testing
- export the results as text or as spreadsheet
- print the results

6.7.1 Layout of the Code Coverage Browser

The following figure shows the **Code Coverage Browser**.

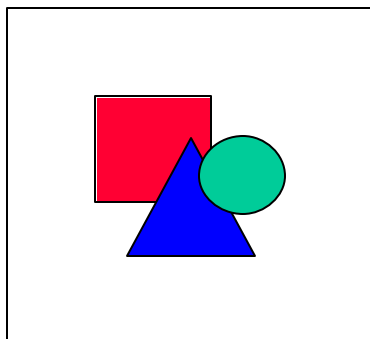


Figure 9 — Code Coverage Browser.

The following table describes the elements of the **Code Coverage Browser**.

Element	Description
menu bar	provides access to component management and allows applications to be added and selected components to be ignored
control buttons	controls whether coverage testing is one of: <ul style="list-style-type: none"> — active (watch) — inactive (stop) — active, but momentarily paused (pause)
status area	shows the number of tested methods in the selected applications as a: <ul style="list-style-type: none"> — fraction (tested over total) — percentage
applications list	displays the applications and subapplications added for coverage testing
classes list	displays classes of the selected application

Element	Description
methods list	displays methods of the selected class
text area	displays the text of the code element
information label	displays the status of the selected method

Table 20 — Elements of the Code Coverage Browser.

6.7.2 Launching the Code Coverage Browser

The **Code Coverage Browser** lets you watch a set of applications while they are being tested. To open the **Code Coverage Browser**, select **Tool** → **Coverage** from a development browser or from the Transcript.

6.7.3 Status Area

The status area gives a brief summary of the number of methods that are still untested in the selected application. The status area shows the:

- number of tested methods
- number of untested methods
- percentage of tested methods for the selected applications

For example:

17/435 (3.91%)

indicates that out of a total of 435 testable methods, 17 have executed, which represents 3.91% of the total.

The total testable methods do not include ignored methods or methods that cannot be monitored by Code Coverage.⁵ If the selected application is displayed with points of ellipsis (...), the status area displays the totals for the entire application hierarchy. If the points of ellipsis are not present, the totals are shown for only the specific applications. When an application is fully tested, the status label indicates **No methods**.

6.7.4 The Control Buttons

The buttons **watch**, **stop**, and **pause** control the operation of the **Code Coverage Browser**.

⁵ On VisualAge for Smalltalk Professional, accessor and primitive methods cannot be monitored by Code Coverage.

watch

Sets the **Code Coverage Browser** to watching mode.

When a method executes, the **Code Coverage Browser** records the fact that the method has been tested. Click **watch** before you start your test suites.

stop

Causes the browser to stop watching your methods.

Click **stop** only when you have completed running your test suites.

pause

Temporarily suspends the **Code Coverage Browser** watching the methods.

If you click **pause**, the button label changes to **resume**. If you click the button again, the browser continues to watch the execution of your methods.

6.7.5 Refreshing the Browser

As you test your application, you may need to explicitly refresh the **Code Coverage Browser** to reflect the current state of the applications being tested. Select **Applications** → **Refresh** to refresh the browser.

6.7.6 Ignoring Components

It is possible that some of your methods are never executed during the course of running your test suites. For example, test cases validating API for adding and removing from a collection might not test the initialization of the class itself. Class initialization might be tested as part of a test suite to verify the correct installation and removal of a class from the image.

The **Code Coverage Browser** lets you ignore methods, classes, or applications that your test cases are not intended to test. This lets you reduce the amount of information presented in the browser and lets you focus on the main problem of identifying the incomplete areas of your test suites.

To ignore the methods in a component, select the component (application, class, or method) and select a menu option:

Ignore all methods in	Menu option
an application	ensure the application's hierarchy is expanded (that is, ... does not follow the application name) and select: Applications → Ignore Application
an application and its subapplications	ensure the application's hierarchy is collapsed (that is, ... follows the application name) and select: Applications → Ignore Application...

Ignore all methods in	Menu option
a class	Classes → Ignore → Class...
a class hierarchy	Classes → Ignore → Class Hierarchy...
methods	Methods → Ignore/Restore
all implementors of a method	Methods → Ignore All Implementors...

Table 21 — Ignoring methods in a component.

To view all methods (including those that have been ignored), select **Methods** → **Show All**. Ignored methods are prefixed with a minus sign (-).

To view only those methods that need to be tested, select **Methods** → **Hide Ignored**. Ignored methods are hidden. Classes and applications that do not have any remaining methods are also removed, unless they have subclasses or subapplications that still contain untested methods.

6.7.7 Restoring Components

To remove the ignore status from your methods:

1. Select the component (application, class, or method) whose ignore status you want to reverse. If the component is not visible, you must first select **Methods** → **Show All**.
2. Select a menu option:

Restore all methods in	Menu option
an application	ensure the application's hierarchy is expanded (that is, ... does not follow the application name) and select: Applications → Restore Application...
applications and its subapplications	ensure the application's hierarchy is collapsed (that is, ... follows the application name) and select: Applications → Restore Application...
a class	Classes → Restore → Class...
a class hierarchy	Classes → Restore → Class Hierarchy...
methods	Methods → Ignore/Restore
all implementors of a method	Methods → Restore All Implementors...

Table 22 — Restoring all methods in a component.

6.7.8 Saving and Loading Your Coverage Setup

You can save to a file the set of applications and ignored methods. You can later use the file to retest your application with minimal setup time.

To save the Code Coverage setup to a file:

1. Select **Applications** → **Save Setup...** or **Applications** → **Save Setup As...**
2. Enter the name of the file to which to save the setup.

To load the setup information from a file:

1. Select **Applications** → **Load Setup...**
2. Enter the name of the file containing your setup. The loaded setup overrides any applications previously added to the **Code Coverage Browser**.

6.7.9 Reporting Results

Code Coverage results can be printed, exported as text, or exported as spreadsheet format. You can customize the contents of a report to produce either an in-depth report or a quick summary.

You can customize which parts of the report you want included. The parts of a report comprise:

- header—user's name, date and time, and comment
- summary of all applications—a summary of the number of testable, untestable, and ignored methods in all applications
- summary by application—a summary of the number of testable, untestable, and ignored methods for each application
- method details by application—a detailed summary of the testable, untestable, and ignored methods in all applications, grouped by application

6.7.10 Exporting Reports to Text

To export results to a text file:

1. Select **Applications** → **Export Report** → **As Text...**
2. Select the sections to include (see Section 6.7.9) and click **OK**.
3. Enter the name of the text file to which to export the results.

6.7.11 Exporting to Spreadsheet

To export results to a spreadsheet file:

1. Select **Applications** → **Export Report** → **As Spreadsheet...**
2. Select the sections to include (see Section 6.7.9) and the type of column delimiter. Click **OK**.

3. Enter the name of the spreadsheet file to which to export the results.

☞ Each field in the export file is separated by a comma or a tab. You can modify the default setting in the **Report Setup** dialog that appears when you create a report.

6.7.12 Printing Results

To print results:

1. Select **Applications** → **Print Report...**
2. Select the sections to include (see Section 6.7.9) and click **OK**.

6.7.13 Adding a Comment to the Results

In some cases, it is useful to annotate your Code Coverage report with comments, especially if you are saving the report to examine later.

To add a comment for a coverage report:

1. Select **Applications** → **Summary Info...**

The **Summary Info** dialog opens.

2. Enter a comment in the **Comment** text area.
3. Click **OK** to accept the comment or **Cancel** to abort.

6.7.14 Menus

This section describes the menus of the **Code Coverage Browser**.

6.7.14.1 File Menu

This menu is the same as existing development browsers.

6.7.14.2 Edit Menu

This menu is the same as existing development browsers.

6.7.14.3 Applications Menu

Save Setup...

Saves to the previous file the set of loaded applications and the set of ignored methods. If this is the first save operation, this option prompts for a filename.

This option is enabled when changes made to the browser have not been saved.

Save Setup As...

Saves to a new file the set of loaded applications and the set of ignored methods.

Load Setup...

Loads a previously saved Code Coverage setup file.

Summary Info...

Displays summary information (user's name, time and date, and comment) for the coverage results.

Export Report

Exports the results to a file.

As Text...

Exports the results to a text file.

As Spreadsheet...

Exports the results to a spreadsheet file; fields are separated by comma or tab.

Print Report...

Prints the results.

Browse Application

Opens an **Application Browser** on the selected application.

Browse Methods

Displays a submenu with the following options. After one of the options is selected, a **Methods Browser** opens that contains a filtered set (see below) of all methods from all applications loaded in the **Code Coverage Browser**.

Tested

Opens a **Methods Browser** with all methods that have been executed (tested).

Untested

Opens a **Methods Browser** with all methods that have not been executed (untested).

Ignored

Opens a **Methods Browser** with all methods that have been ignored.

Add

Displays a submenu of options to add components.

Application...

Adds one or more applications to the **Code Coverage Browser**.

Application Hierarchy...

Adds one or more applications and their subapplications to the **Code Coverage Browser**.

SubApplication...

Adds one or more subapplications to the **Code Coverage Browser**.

Configuration Map...

Adds to the **Code Coverage Browser** the applications of a chosen configuration map.

Remove

Displays a submenu with options to remove selected applications.

Application...

Removes the selected applications and their subapplications from the **Code Coverage Browser**.

Configuration Map...

Removes from the **Code Coverage Browser** the applications and subapplications of the selected configuration map.

Ignore Application...

Ignores all methods in the selected applications.

Restore Application...

Restores all ignored methods in the selected applications.

Reset

Displays a submenu with options to reset methods to the untested state.

Application...

Resets to the untested state all methods in the selected applications.

All Applications...

Resets to the untested state all methods in all applications.

Verbose Dialogs

Toggles whether dialogs are verbose or terse. You should use verbose dialogs initially; when you become familiar with Code Coverage, you can switch to terse dialogs.

Refresh

Updates the browser to reflect changes made to its applications, classes, and methods.

6.7.14.4 Classes Menu

Find**Browse References****Browse Messages****Browse Class****Browse Hierarchy**

These menu options are the same as existing development browsers.

Ignore

Ignores methods in the selected classes.

Class...

Ignores all methods in the selected classes.

Class Hierarchy...

Ignores all methods in the selected classes and their subclasses.

Restore

Displays a submenu with the options to restore methods to the untested state.

Class...

Restores to the untested state all methods in the selected classes.

Class Hierarchy...

Restores to the untested state all methods in the selected classes and their subclasses.

Reset

Displays a submenu with options to reset methods.

Class...

Resets all methods in the selected classes.

Class Hierarchy...

Resets all methods in the selected classes and their subclasses.

6.7.14.5 Methods Menu

Browse Senders**Browse Implementors****Browse Messages****Browse References Classes...**

These menu options are the same as existing development browsers.

Ignore/Restore

Toggles the selected methods between ignored and restored.

Ignore All Implementors...

Ignores all implementors of the selected methods in all applications being viewed by the **Code Coverage Browser**.

Restore All Implementors...

Restores all implementors of the selected methods in all applications being viewed by the **Code Coverage Browser**.

Hide Ignored

Hides ignored methods.

If all methods of a class are ignored, the class is hidden. If all methods in an application are ignored, the application is hidden.

Show All

Shows all methods including ignored methods.

Ignored methods are prefixed with a minus sign (-).

6.7.14.6 Info Menu

This menu is the same as existing development browsers.

7 Code Publisher

7.1 Overview

Code Publisher produces typeset-quality manuals from applications, classes, and methods. The report structure is highly customizable. You can easily create documents that include only the API methods and their comments. You can produce in-depth manuals containing code, cross-reference tables, and quick look-up indexes to be used during code reviews.

Code Publisher can export to these formats:

- LaTeX
- RTF
- MIF
- HTML
- OTIML

HTML manuals are hyperlinked internally to let you navigate easily on line. To further improve readability, embedded GIFs are included in the HTML output.

7.2 Who Should Use This Tool

Developers can use Code Publisher to create:

- code documents for use in code inspections

Project managers can use Code Publisher to create:

- API documentation for delivery to customers
- HTML documentation suitable for the World Wide Web

Release engineers can use Code Publisher to create:

- high-level summaries of their software components

7.3 Loading Code Publisher

Code Publisher is installed automatically as part of the complete installation of **ENVY/QA** (see Chapter 3).

If you want to use only Code Publisher:

1. Open the **Configuration Maps Browser**.
2. Load the most recent edition of the **QA Code Publisher** configuration map with required maps.
3. If you plan to use Code Publisher to publish Microsoft Word manuals, copy `OTIMLRTF.DOT` to the directory containing your Microsoft Word user templates. (For the appropriate directory, refer to the Microsoft Word documentation.)

 To determine the file location of the user templates, in Microsoft Word select **Tool** → **Options...**

4. If you plan to use Code Publisher to publish LaTeX manuals, copy `SYSPUB.STY` and `SYSPUBF.STY` to the directory containing your LaTeX style files. (For appropriate directory, refer to the LaTeX documentation.)

When Code Publisher is loaded:

- the system menu has a new submenu **Tool Settings** → **Publish...**
- newly opened development browsers have a **Tool** → **Publish** menu

You are now ready to use Code Publisher.

7.4 Unloading Code Publisher

To unload Code Publisher:

1. Open the **Configurations Maps Browser**.

2. Unload (in order) the loaded edition of the following configuration maps:
 - (a) **QA Code Publisher**
 - (b) **OTIML System Publisher**
 - (c) **OTIML Publishing Backend**
3. If Code Publisher is the only **ENVY/QA** tool loaded, you can also unload **QA Framework**.

7.5 Guided Tour

In this section, we guide you through the process of publishing the API of the application **SqaEtBrowserExtensions**.

1. Open an **Application Manager**.
2. Select the application **SqaEtBrowserExtensions**.
3. Select **Applications** → **Tool** → **Publish** → **Applications...**
4. When the **Code Publisher Output Options** dialog opens, change the **Document Title** to be *SQA API*.
5. Select the **RTF** radio button.
6. Select **Categories to Publish...** and ensure *SQA-API* is the only included category. Click **OK**.
7. Select **Settings...**
8. When the **Code Publisher Settings** dialog opens, select **Application** in the left list.
 - (a) Click **None**.
 - (b) Click the check boxes for **Application class** and **Public classes**.

The output contains the application class and the public classes of the application.
 - (c) Select **Class** in the left list.
 - (d) Click **None**.
 - (e) Click the check boxes for **Class comment** and **Public methods**.

The output contains class comments and the public methods of the published classes.
 - (f) Select **Method** in the left list.
 - (g) Click **None**.
 - (h) Click the check box for **Method header comment (from top of method)**.

The output contains only the comment from the start of each method. The code for each method is omitted.

(i) Click **OK**.

9. Click **OK**.

Code Publisher creates your document.

10. When the document is created, you can open it in Microsoft Word.

When the document opens, a macro automatically adjusts the tables, figures, and pagination.

7.5.1 Publishing Configuration Maps

To publish a configuration map:

1. Open the **Configuration Maps Browser**.
2. Select a configuration map edition.
3. Select **Editions** → **Tool** → **Publish**.
4. In the **Code Publisher Options** dialog, click **OK** to start publishing.

7.5.2 Publishing Applications

You can publish applications from the following browsers:

- **Configuration Maps Browser**
- **Application Manager**
- **Applications Browser**
- **Classes Browser**
- **Class Hierarchy Browser**
- **Visual Age Organizer**

To publish an application:

1. Open one of the browsers mentioned above.
2. Select the applications to be published.

3. Select a menu option:

Item to publish	Menu option
applications	Applications → Tool → Publish → Application...
applications and their loaded subapplications	Applications → Tool → Publish → Include All Subapplications... ⁶

Table 23 — Publishing applications.

4. In the **Code Publisher Options** dialog, click **OK** to start publishing.

7.5.3 Publishing Classes

You can publish classes from the following browsers:

- **Application Manager**
- **Application Browser**
- **Applications Browser**
- **Classes Browser**
- **Class Hierarchy Browser**

To publish a class:

1. Open one of the browsers mentioned above.
2. Select the classes to be published.
3. Select a menu option:

Item to publish	Menu option
classes	Classes → Tool → Publish → Class...
classes and all their subclasses	Classes → Tool → Publish → Include All Subapplications...
class fragments	Classes → Tool → Publish → Class Fragment...

Table 24 — Publishing classes.

4. In the **Code Publisher Output Options** dialog, click **OK** to start publishing.

A class fragment is a part of a class within a single application, rather than an entire class (which may span many class extensions as well as the class definition).

⁶ In the **Configuration Maps Browser**, select **Applications → Tool → Publish**.

7.5.4 Code Publisher Output Options Dialog

The following figure shows the **Code Publisher Output Options** dialog. You use this dialog to specify the name and format of your output file.

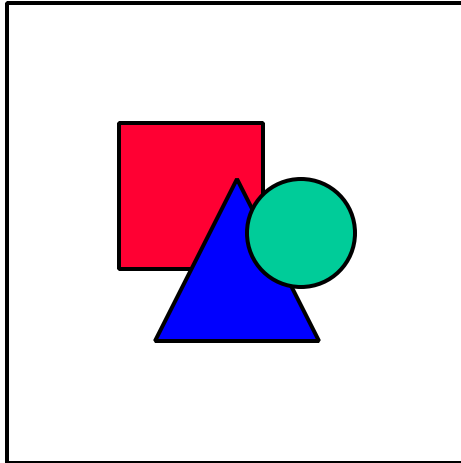


Figure 10 — Code Publisher Output Options dialog.

The following table describes the elements of the **Code Publisher Output Options** dialog.

Element	Description
Document Title	specifies the title for the document (The title appears on the front cover of the document.)
Output	specifies the name and format of the file to create
Browse...	lets you select a specific file and directory (Some output formats—for example, HTML—create multiple files, whose names are based on the entered name.)
Categories to Publish...	lets you include or exclude specific method categories from your document
Settings...	opens the Code Publisher Settings dialog

Table 25 — Elements of the Code Publisher Output Options dialog.

7.6 Output Formats

This section describes the output formats supported by Code Publisher. You can specify the output using the **Code Publisher Output Options** dialog.

- ✘ If you do not install the style files properly or do not correctly associate the template files specified for each output type, the pages will not format correctly. You will see problems, such as, table columns that are too narrow and incorrect font selections and sizes.

7.6.1 HTML—Hypertext Markup Language

HTML is the popular minimalist SGML⁷ markup language used to prepare Web pages on the Internet. HTML markup tags are continuously evolving.

Code Publisher creates several files for HTML output: one file for each entry in the table of contents. In the **Code Publisher Output Options** dialog, you can specify the name of the main document file. For example, if you enter `FILE`, Code Publisher creates `FILE.HTM`. Additional files are created by adding a unique number to your filename (for example, `FILE1.HTM`). If the filename is too long for the file system, the filename is modified.

To improve the readability of the HTML document, Code Publisher creates and uses the following two GIF files:

- `GOTO.GIF`
- `RED-BALL.GIF`

- ☞ Numerous small files are created instead of one large file, because small files improve Web-based navigation of your HTML document.

7.6.2 LaTeX—A Document Preparation System

LaTeX⁸ is a collection of macros that makes Donald Knuth's TeX⁹ typesetting system easy to use. Code Publisher creates one file for LaTeX output. The name of the file corresponds to the name you enter

⁷ Charles F. Goldfarb, *The SGML Handbook*, Oxford University Press, 1994.

⁸ Leslie Lamport, *LaTeX: A Document Preparation System*, Addison-Wesley Publishing, 1986.

⁹ Donald E. Knuth, *The TeXbook*, Addison-Wesley Publishing, 1986.

in the **Code Publisher Output Options** dialog. If you enter `FILE`, Code Publisher creates `FILE.TEX`.

The generated LaTeX file also requires two style files:

- `SYSPUB.STY`
- `SYSPUBF.STY`

7.6.3 MIF—Maker Interchange Format

The MIF file format is used by the FrameMaker electronic publishing product to provide an import and export mechanism. Code Publisher creates several files for MIF output. If you enter `FILE` in the **Code Publisher Output Options** dialog, Code Publisher creates the files shown in the following table.

Filename	Contents
<code>FILE.BK</code>	book file (the main document file)
<code>FILE.COV</code>	cover page
<code>FILE.TOC</code>	table of contents
<code>FILE.MIF</code>	document body
<code>FILE.IX</code>	index

Table 26 — MIF output files.

The book file is the main document and contains links to the other document files.

To view the document in FrameMaker, open the book file (`FILE.BK`). Create a new table of contents and index by updating the files:

- `FILE.TOC`
- `FILE.IX`

7.6.4 OTIML—OTI Markup Language

OTIML is an example SGML markup language that provides document styling elements similar to those provided with the LaTeX `*.STY` files described in Section 7.6.2. Code Publisher creates one file for OTIML output. The name of the file corresponds to the name you enter in the **Code Publisher Output Options** dialog. If you enter `FILE`, Code Publisher creates `FILE.OML`.

The `OTIML.DTD` uses standard SGML DTD specifications to define the structure of the created document.

7.6.5 RTF—Rich Text Format

The RTF file format provides an import and export facility for Microsoft Word documents. Code Publisher creates one file for RTF

output. The name of the file corresponds to the name you enter in the **Code Publisher Output Options** dialog. If you enter `FILE`, Code Publisher creates `FILE.RTF`.

The generated RTF file also requires the Word template file `OTIMLRTF.DOT`. The template automatically adjusts the RTF document as it is opened. To skip the adjustments, press `SHIFT` while the document is opening.

7.7 Code Publisher Settings Dialog

You can customize the contents of your document using the **Code Publisher Settings** dialog shown below.

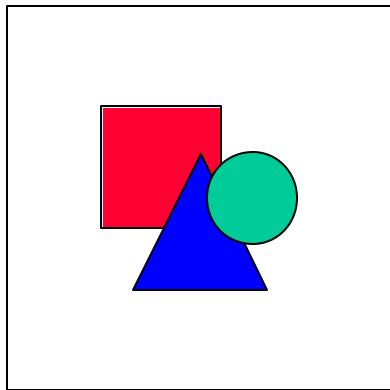


Figure 11 — Code Publisher Settings dialog.

The **Code Publisher Settings** dialog groups publishing options into the categories shown in the following table.

Category	Options defining
Application	which application elements to include
Class	which class elements to include
Method	which method elements to include
Cross reference	which cross-references to include
General	general options that apply to the entire document

Table 27 — Groupings of publisher options.

To select specific publishing options:

1. In the **Code Publisher Output Options** dialog, select **Settings...**, or from the Transcript, select **Tool Settings** → **Publish...**

2. Select a category in the left list.

The list on the right shows check boxes for each publishing option in that category.

3. Do one of the following:

Desired result	Action
enable all options in a category	click All
disable all options in a category	click None
enable/disable specific options	click individual check boxes
reset to their default values all options in all categories	click Reset All

Table 28 — Enabling publisher options.

4. Click **OK** to accept the selected publisher options or **Cancel** to abort the selection.

7.8 Application Publishing Options

The following table describes the application publishing options.

Property	Includes
Application all prerequisites list	the list of all prerequisite applications
Application class	the class of an application
Application class hierarchy	the class hierarchy for all published classes in the application
Application immediate prerequisites list	the list of immediate prerequisite applications
Extended classes	extended classes
Private classes	private classes
Public classes	public classes

Table 29 — Application publishing options.

7.9 Class Publishing Options

The following table describes the class publishing options.

Property	Includes
Class comment	the class comment field
Class definition	the class definition of each class
Class hierarchy	the class hierarchy at the start of each class's section (The hierarchy shows superclasses of the class.)
Class notes	the class notes field

Property	Includes
List of private inherited methods	the table that maps inherited private methods to the implementing superclass
List of public inherited methods	the table that maps inherited public methods to the implementing superclass
Private methods	private class and instance methods
Public methods	public class and instance methods

Table 30 — Class publishing options.

7.10 Method Publishing Options

The following table describes the method publishing options.

Property	Includes
Method application name	the name of the application in which the method is defined
Method categories list	the list of categories in which the method is defined
Method comment field	the method comment field
Method header comment (from top of method)	the comment that appears at the start of a method
Method notes field	the method notes field
Private method body	the code body of private methods
Public method body	the code body of public methods

Table 31 — Method publishing options.

7.11 Cross-Reference Publishing Options

The following table describes the cross-reference publishing options.

Property	Includes the table that maps
Class to application	classes to the applications containing them
Class to referencing selectors	classes to the methods referencing them
Global to referencing selectors	global variables to the methods referencing them
Selector to implementing classes	selector to the classes implementing it

Table 32 — Cross-reference publishing options.

7.12 General Publishing Options

The following table describes the general publishing options.

Property	Description
Empty classes	includes classes that would be empty in the published document
Index	includes an index of classes and methods
Skip empty sections	omits sections that would be empty in the published document

Table 33 — General publishing options.

7.13 Saving and Loading Your Settings

The **Code Publisher Settings** dialog lets you save and load settings.

To save the currently enabled publisher settings:

1. Open the **Code Publisher Settings** dialog from any **Tool** → **Publish** menu item or select **Tool Settings** → **Publish...** from the Transcript.
2. Click **Save To File...**
3. Enter the name of the file to which to save the settings.

To load settings from a file:

1. Open the **Code Publisher Settings** dialog from any **Tool** → **Publisher** menu item or select **Tool Settings** → **Publish...** from the Transcript.
2. Click **Load From File...**
3. Enter the name of the file containing your settings. The loaded settings override any previous settings.

8 Code Formatter

8.1 Overview

Code Formatter lets you format Smalltalk source code. You can format:

- entire classes
- class hierarchies
- applications
- configuration maps
- method source while you are editing it

Custom controls let you define your preferred formatting style and a preview mechanism lets you determine quickly how the code will look.

8.2 Who Should Use This Tool

Developers can use Code Formatter to:

- standardize coding styles and improve the consistency of code among team members
- quickly check the syntax of methods without saving the methods
- reduce time spent understanding the style of other developers

Release engineers can use Code Formatter to:

- standardize coding styles and improve the consistency of code among team members

8.3 Loading Code Formatter

Code Formatter is installed automatically as part of the complete installation of **ENVY/QA** (see Chapter 3).

If you want to use only Code Formatter:

1. Open the **Configuration Maps Browser**.
2. Load the most recent edition of the **QA Code Formatter** configuration map with required maps.

When Code Formatter is loaded:

- the system menu has a new submenu **Tool Settings** → **Format...**
- newly opened development browsers have a **Tool** → **Format** menu
- development browsers showing method source have these menu options:
 - **Edit** → **Format**
 - **Edit** → **Format Settings...**

You are now ready to use Code Formatter.

8.4 Unloading Code Formatter

To unload Code Formatter:

1. Open the **Configurations Maps Browser**.
2. Unload the loaded edition of the **QA Code Formatter** configuration map.
3. If Code Formatter is the only **ENVY/QA** tool loaded, you can also unload **QA Framework**.

8.5 Guided Tour

In this section, we guide you through the process of:

- setting up a style
- formatting a method

8.5.1 Creating a Class and Method

Create the following class and method.

Person class

```
Object subclass: #Person
  instanceVariableNames: 'name age '
  classVariableNames: "
  poolDictionaries: "
```

Person public class methods

```
named: aString age: anInteger
  "Answer a new instance of the receiver whose name is aString
  and age is anInteger."
  ^self new name: aString; age: anInteger
```

1. Open a **Class Browser** on the *Person* class.
2. Select the method **#named:age:**.
3. Select **Edit** → **Format Settings...**
4. When the **Code Formatter Settings** dialog opens, select **Keywords** from the left list.
5. In the right list, enable option **Always split keywords**.
6. Click **Preview**.

The code shown in the **Class Browser** is formatted.

7. Experiment with the other settings options by enabling or disabling them and clicking **Preview**.
8. When you have finished, either click **Cancel** to restore the text to its original form (prior to opening the dialog) or click **OK** to keep the changes.

8.6 Code Formatter and the Development Browsers

Code Formatter is integrated fully with the development browsers. You can format code elements using the **Tool** → **Format** or **Edit** → **Format** menu items in applicable development browsers.

Code Formatter lets you format the following types of code elements:

- source shown in a browser
- method
- class
- class fragment
- class hierarchy
- application
- application hierarchy

- loaded applications of a configuration map edition

8.6.1 Formatting Configuration Maps

To format a configuration map:

1. Open the **Configuration Maps Browser**.
2. Select a configuration map edition.
3. Select **Editions** → **Tool** → **Format**.
4. In the confirmation dialog, click **OK** to start formatting.

8.6.2 Formatting Applications

You can format applications from the following browsers:

- **Configuration Maps Browser**
- **Application Manager**
- **Applications Browser**
- **Classes Browser**
- **Class Hierarchy Browser**
- **Visual Age Organizer**

To format an application:

1. Open one of the browsers mentioned above.
2. Select the applications to be formatted.
3. Select a menu option:

Item to format	Menu option
applications	Applications → Tool → Format → Application...
applications and their loaded subapplications	Applications → Tool → Format → Include All Subapplications... ¹⁰

Table 34 — Formatting applications.

4. In the confirmation dialog, click **OK** to start formatting.

8.6.3 Formatting Classes

You can format classes from the following browsers:

- **Application Manager**

¹⁰ In the **Configuration Maps Browser**, select **Applications** → **Tool** → **Format**.

- **Application Browser**
- **Applications Browser**
- **Classes Browser**
- **Class Hierarchy Browser**
- **Visual Age Organizer**

To format a class:

1. Open one of the browsers mentioned above.
2. Select the classes to be formatted.
3. Select a menu option:

Item to format	Menu option
classes	Classes → Tool → Format → Class...
classes and all their subclasses	Classes → Tool → Format → Include All Subclasses...
class fragments	Classes → Tool → Format → Class Fragment...

Table 35 — Formatting classes.

4. In the confirmation dialog, click **OK** to start formatting.

8.6.4 Formatting Methods

You can format methods from the following browsers:

- **Application Browser**
- **Applications Browser**
- **Classes Browser**
- **Class Browser**
- **Class Hierarchy Browser**
- **Methods Browser**

To format a method:

1. Open one of the browsers mentioned above.
2. Select the methods to be formatted.
3. Select **Methods** → **Tool** → **Format**.
4. In the confirmation dialog, click **OK** to start formatting.

8.7 Formatting Source

You can format source code shown in the text area of the following browsers:

- **Application Browser**

- **Applications Browser**
- **Classes Browser**
- **Class Browser**
- **Class Hierarchy Browser**
- **Methods Browser**
- **Debugger**

To format source code from the text area:

1. Open one of the browsers mentioned above.
2. Select a method or begin a new method.
3. Select **Edit** → **Format** or press CTRL+W.

8.8 Code Formatter Settings Dialog

You can customize the way your source code is formatted by using the **Code Formatter Settings** dialog, shown below. To open the dialog:

- from the Transcript, select **Tool Settings** → **Formatter...**
- or
- from a development browser, select **Edit** → **Format Settings...**

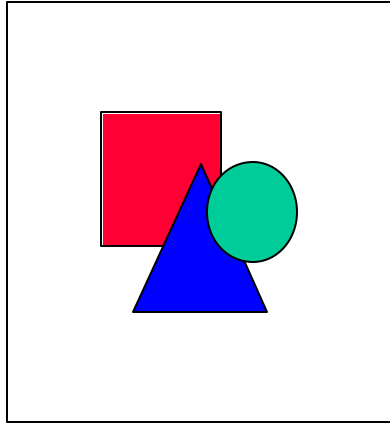


Figure 12 — Code Formatter Settings dialog.

The **Code Formatter Settings** dialog groups options into the categories shown in the following table.

Category	Options defining
Blocks	how blocks are formatted
Parentheses	the positioning of parentheses
Comments	how comments are formatted
Conditional Statements	how conditionals are formatted
Indent	indentation
Keywords	how keyword selectors are formatted
Line breaks	when to introduce line breaks
General	general formatting

Table 36 — Categories of formatter options.

You can use the formatting options currently selected in the **Formatter Settings Dialog** to preview the code that is displayed in the text pane from which the dialog was opened. To do this, click **Preview**. If you click **Cancel**, the dialog closes and the formatting is reset.

To select specific formatting options:

1. Select a category in the left list.

The list on the right shows check boxes representing each formatting option in the selected category.

2. Do one of the following:

Desired result	Action
enable all options in a category	click All
disable all options in a category	click None
enable or disable specific options	click individual check boxes
reset to their default values all options in all categories	click Reset All

Table 37 — Enabling formatter options.

3. Click **OK** to accept the selected formatter options or **Cancel** to abort the selection.

8.9 Formatting Blocks

This section describes the options for formatting blocks.

8.9.1 Block Arguments on the Start Line

If this option is enabled, block arguments are kept on the same line as the open square bracket.

OFF

(do not keep the arguments on the start line)

```
self verify: [
  :a |
  temp := a.
  a]
```

ON

(keep the arguments on the start line)

```
self verify: [:a |
  temp := a.
  a]
```

8.9.2 Block Variables on a New Line

If this option is enabled, block variables are placed on a new line.

OFF

(do not place the variables on a new line)

```
self verify: [| var1 |  
    var1 := 2.  
    var1 + 4]
```

ON

(place the variables on a new line)

```
self verify: [  
    | var1 |  
    var1 := 2.  
    var1 + 4]
```

8.9.3 Line up Brackets

If this option is enabled, brackets are lined up. They are lined up only if the expression does not fit on one line.

OFF

(do not line up brackets)

```
aBlock :=  
    [  
        self sizeOf: myCollection.  
        self enable]
```

ON

(line up brackets)

```
aBlock := [  
    self sizeOf: myCollection.  
    self enable  
]
```

8.9.4 Space between Square Brackets

If this option is enabled, a space is inserted after opening square brackets and before closing square brackets.

OFF
(do not insert spaces)

```
[self sizeOf: myCollection]
```

ON
(insert spaces)

```
[ self sizeOf: myCollection ]
```

8.9.5 Start Open Brackets on the Same Line

If this option is enabled, opening brackets follow the selector.

OFF
(brackets do not follow the selector)

```
self myCollection do:  
    [:element|  
    ...]
```

ON
(brackets follow the selector)

```
self myCollection do:[:element|  
    ...]
```

8.10 Formatting Comments

This section describes the options for formatting comments.

8.10.1 Format Comments in the Body of the Code

If this option is enabled, comments in the body of the code are formatted.

OFF
(do not format comments in the body of the code)

sample

```
self location: 2 "this is a lengthy comment that should be formatted"
```

ON
(format comments in the body of the code)

sample

```
self location: 2 "this is a lengthy comment that should be
formatted"
```

8.10.2 Format Method Header Comment

If this option is enabled, the main method comment is formatted.

OFF
(do not format the method header comment)

sample

```
"This is a very, very long comment about the sample method and it should be
reformatted."
```

```
^self
```

ON
(format the method header comment)

sample

```
"This is a very, very long comment about the sample method and it
should be reformatted."
```

```
^self
```

8.11 Formatting Conditional Statements

This section describes the options for formatting conditional statements.

8.11.1 Apply Rule (Line up #ifTrue:ifFalse:) to All

If this option is enabled, the **Line up #ifTrue:ifFalse:** selection is always applied to short or long statements.

If the **Line up #ifTrue:ifFalse:** option is on, **ifTrue:** is always on a new line.

If the **Line up #ifTrue:ifFalse:** option is off, **ifTrue:** is always on the same line.

Several related options help you obtain your preferred formatting of **#ifTrue:ifFalse:**. These include:

- line up **#ifTrue:ifFalse:** (in the Conditional Statements category)
- indent **#ifFalse:** (when not lined up) (in the Conditional Statements category)
- start square brackets on the same line (in the Brackets category)

OFF

(do not apply the lineup option to short statements)

```
^self isEnabled ifTrue: [self] ifFalse: [nil]
```

ON

(and the lineup option is ON)

```
^self isEnabled
  ifTrue: [self]
  ifFalse: [nil]
```

ON (and the lineup option is OFF)

```
^self isEnabled ifTrue: [
  self]
ifFalse: [
  nil]
```

8.11.2 Indent #ifFalse: (When Not Lined Up)

If this option is enabled and **#ifTrue:ifFalse:** is not lined up, **ifFalse:** is indented.

Several related options help you obtain your preferred formatting of **#ifTrue:ifFalse:**. These include:

- line up **#ifTrue:ifFalse:** (in the category Conditional Statements)
- apply rule (line up **#ifTrue:ifFalse:**) to all (in the Conditional Statements category)
- start square brackets on the same line (in the Brackets category)

OFF

(do not indent)

```
self isEnabled ifTrue: [
    self setPositionTo: 12 usingGlobalFactorOf: 15]
ifFalse: [
    self setPositionTo: 10 usingGlobalFactorOf: 10]
```

ON

(indent)

```
self isEnabled ifTrue: [
    self setPositionTo: 12 usingGlobalFactorOf: 15]
ifFalse: [
    self setPositionTo: 10 usingGlobalFactorOf: 10]
```

8.11.3 Line up #ifTrue:ifFalse:

If this option is enabled, **ifTrue:ifFalse:** is lined up.

Several related options that help you obtain your preferred formatting of **#ifTrue:ifFalse:**. These include:

- indent **#ifFalse:** (when not lined up) (in the Conditional Statements category)
- apply the rule (line up **#ifTrue:ifFalse:**) to all (in the Conditional Statements category)
- start square brackets on the same line (in the Brackets category)

OFF

(do not line up)

```
self isEnabled ifTrue: [
    self setPositionTo: 12 usingGlobalFactorOf: 15]
ifFalse: [
    self setPositionTo: 10 usingGlobalFactorOf: 10]
```

ON

(line up)

```
self isEnabled
    ifTrue: [self setPositionTo: 12 usingGlobalFactorOf: 15]
    ifFalse: [self setPositionTo: 10 usingGlobalFactorOf: 10]
```

8.12 Formatting Indentation

This section describes the options for formatting indentation.

8.12.1 Indent Long Keywords in Cascades

If this option is enabled, long keywords are indented if they are cascaded.

OFF

(do not indent long keywords in cascades)

```
self
  setPositionTo: 12
  usingGlobalFactorOf: 15
  refreshDisplayOnCompletion: true;
  setPositionTo: 13
  usingGlobalFactorOf: 15
  refreshDisplayOnCompletion: true
```

ON

(indent long keywords in cascades)

```
self
  setPositionTo: 12
    usingGlobalFactorOf: 15
    refreshDisplayOnCompletion: true;
  setPositionTo: 13
    usingGlobalFactorOf: 15
    refreshDisplayOnCompletion: true
```

8.12.2 Indent Nested Receivers

If this option is enabled, messages to nested receiver are indented.

OFF

(do not indent messages to nested receivers)

```
self sampleMethodWhichIsLong
  anotherMessage
  yetAnotherLengthyMessage
```

ON

(indent messages to nested receivers)

```
self sampleMethodWhichIsLong
  anotherMessage
    yetAnotherLengthyMessage
```

8.13 Formatting Keywords

This section describes the options for formatting keywords.

8.13.1 Always Put a Single Keyword on the Same Line

If this option is enabled, keywords with only one selector and one argument are kept on the same line as the receiver.

OFF

(do not keep on the same line)

```
theReceiverIsAVeryLongVariableName  
    setPositionTo: 12
```

ON

(keep on the same line)

```
theReceiverIsAVeryLongVariableName setPositionTo: 12
```

8.13.2 Always Split Keywords

If this option is enabled, keywords are always split into multiple lines.

OFF

(do not split keywords)

```
self setPositionTo: 12 usingGlobalFactorOf: 15
```

ON

(split keywords)

```
self  
    setPositionTo: 12  
    usingGlobalFactorOf: 15
```

8.13.3 First Selector of Long Keywords on the Same Line

If this option is enabled, the first part of a long keyword is kept on the same line as the receiver.

OFF
(do not keep on the same line)

```
self
  setPositionTo: 12
  usingGlobalFactorOf: 15
  refreshDisplayOnCompletion: true
```

ON
(keep on the same line)

```
self setPositionTo: 12
  usingGlobalFactorOf: 15
  refreshDisplayOnCompletion: true
```

8.13.4 Keep #to:do: on the Same Line

If this option is enabled, **#to:do:** is kept on the same line. This option overrides the **Always split keywords** option for **#to:do:**.

Several related options help you obtain your preferred formatting of **#to:do:**. These include:

- block arguments on the same line (in the Blocks category)
- start square brackets on the same line (in the Brackets category)

OFF
(do not keep on the same line)

```
1
  to: 30
  do: [:index |
    Transcript
      cr;
      show: myCollection at: index]
```

ON
(keep on the same line)

```
1 to: 30 do: [:index |
  Transcript
    cr;
    show: myCollection at: index]
```

8.13.5 Keep Right-Hand Side of an Assignment (:=) on the Same Line

If this option is enabled, the right-hand side of an assignment is kept on the same line as the assignment.

OFF
(do not keep on the same line)

```
a :=
  self
    setPositionTo: 12
    usingGlobalFactorOf: 15
    refreshDisplayOnCompletion: true
```

ON
(keep on the same line)

```
a := self
  setPositionTo: 12
  usingGlobalFactorOf: 15
  refreshDisplayOnCompletion: true
```

8.14 Formatting Line Breaks

This section describes the options for formatting line breaks.

8.14.1 Blank Line between Comments and Temps

If this option is enabled, a blank line is inserted between the method comment and the temporary variables.

OFF
(do not insert a blank line)

```
sample
  "A sample method"
  | sampleVariable |
  ^self size
```

ON
(insert a blank line)

```
sample
  "A sample method"

  | sampleVariable |
  ^self size
```

8.14.2 Blank Line between Selector and Comments

If this option is enabled, a blank line is inserted between the method selector and the main comment for the method.

OFF
 (do not insert a blank line)

```
sample
  "A sample method"
  | sampleVariable |
  ^self size
```

ON
 (insert a blank line)

```
sample

  "A sample method"
  | sampleVariable |
  ^self size
```

8.14.3 Blank Line between Temps and the Body of the Method

If this option is enabled, a blank line is inserted between the temporary variables and the method body.

OFF
 (do not insert a blank line)

```
sample
  "A sample method"
  | sampleVariable |
  ^self size
```

ON
 (insert a blank line)

```
sample

  "A sample method"
  | sampleVariable |

  ^self size
```

8.14.4 Retain Blank Lines

If this option is enabled, blank lines are retained.

OFF
(do not retain blank lines)

```
sample
  self location: 2.
  ^self size
```

ON
(retain blank lines)

```
sample
  self location: 2.

  ^self size
```

8.15 Formatting Parentheses

This section describes the options for formatting parentheses.

8.15.1 Line up Array Parentheses

If this option is enabled, array parentheses are lined up. They are lined up only if the array does not fit on one line.

OFF
(do not line up parentheses)

```
self testArray:
  (AAAAA BBBB CCCCC DDDD
  EEEE FFFF)
```

ON
(line up parentheses)

```
self testArray:
  (AAAAA BBBB CCCCC DDDD
  EEEE FFFF
  )
```

8.15.2 Line up Parentheses

If this option is enabled, parentheses are lined up. They are lined up only if the expression does not fit on one line.

OFF
(do not line up parentheses)

```
^self location:
  (self
    sizeOf: myCollection
    afterRemoving: otherCollection
    includingDuplicates: true)
```

ON
(line up parentheses)

```
^self location:
  (self
    sizeOf: myCollection
    afterRemoving: otherCollection
    includingDuplicates: true
  )
```

8.15.3 Space between Parentheses

If this option is enabled, a space is inserted after an opening parenthesis and before a closing parenthesis.

OFF
(do not insert spaces)

```
2 + (self sizeOf: myCollection)
```

ON
(insert spaces)

```
2 + ( self sizeOf: myCollection )
```

8.15.4 Start Open Array Parentheses on the Same Line

If this option is enabled, opening array parentheses are placed on the same line as the selector.

OFF
(parenthesis does not follow the selector)

```
self testArray:  
    (AAAAA BBBB CCCCC DDDD  
     EEEEE)
```

ON
(parenthesis follows the selector)

```
self testArray: (AAAAA BBBB CCCCC DDDD  
                EEEEE)
```

8.15.5 Start Open Parentheses on the Same Line

If this option is enabled, opening parentheses follow the selector.

OFF
(parenthesis does not follow the selector)

```
^self location:  
    (self  
     sizeof: myCollection  
     afterRemoving: otherCollection  
     includingDuplicates: true)
```

ON
(parenthesis follows the selector)

```
^self location: (  
    self  
    sizeof: myCollection  
    afterRemoving: otherCollection  
    includingDuplicates: true)
```

8.16 General Formatting

This section describes the options for the formatting general category.

8.16.1 Add Optional Periods

If this option is enabled, optional periods are added.

OFF
(do not add optional periods)

```
^[a | a + self size]
```

ON
(add optional periods)

```
^[a | a + self size.]
```

8.16.2 Add a Space after Return (^)

If this option is enabled, a space is inserted after return (^)

OFF
(do not insert a space)

```
sample
  ^self
```

ON
(insert a space)

```
sample
  ^ self
```

8.16.3 Use the Maximum Width (Ignore the Window Width)

If this option is enabled, the maximum width is used and the width of the current window is ignored. The following examples use a very narrow window.

OFF
(use window width)

```
sample
  ^Transcript
    cr;
  show:
    self location
    printString
```

ON
(use maximum width)

```
sample
  ^Transcript
    cr;
  show: self location printString
```


8.17 Advanced Options

Code Formatter provides advanced options that let you control the maximum line width and the amount of space used when indenting code and comments. To change the values of these advanced options, click **Advanced...** from the **Code Formatter Settings** dialog.

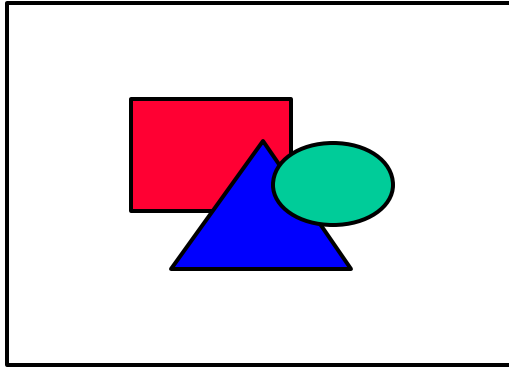


Figure 13 — Formatter Advanced Settings dialog.

When the dialog opens, proceed as follows:

1. Select a group name in the left list and a property from the right list.

The current value of the property appears in the **Preference Value** text area.

2. Modify the text area to the desired value of the property, according to the following table.

Property	Description
single-value property	must be Boolean, Integer, String, or Symbol, depending on the expected type of the property
multi-value property	elements are separated by spaces
values for class property	class names must be valid class names

Table 38 — Property values and their descriptions.

3. Click **Save Value** to save the new value of the property.
4. When you have made all modifications, click **OK** to accept the changes or **Cancel** to abort the changes.

8.17.1 Formatting Code Indentation

This section describes the options for formatting code indentation.

8.17.1.1 Code

This option specifies the number of spaces to indent code in addition to the current indentation level. If it is possible, spaces are automatically replaced by the appropriate number of tabs.

VALUE= 4

```
^(self
  setPositionTo: 12
  usingGlobalFactorOf: 15
  refreshDisplayOnCompletion: true) + 2
```

VALUE= 8

```
^(self
  setPositionTo: 12
  usingGlobalFactorOf: 15
  refreshDisplayOnCompletion: true) + 2
```

8.17.1.2 Code Continuation

This option specifies the number of spaces to indent a code continuation in addition to the current indentation level. If it is possible, spaces are automatically replaced by the appropriate number of tabs.

VALUE= 0

```
^(self
  setPositionTo: 12
  usingGlobalFactorOf: 15
  refreshDisplayOnCompletion: true) + 2
```

VALUE= 4

```
^(self
  setPositionTo: 12
  usingGlobalFactorOf: 15
  refreshDisplayOnCompletion: true) + 2
```

8.17.2 Formatting Comment Indentation

This section describes the options for formatting comment indentation.

8.17.2.1 Beside Line of Code

This option specifies the number of spaces to indent a comment beside a line of code. The number of spaces is in addition to the current indentation level. If it is possible, spaces are automatically replaced by the appropriate number of tabs.

VALUE= 0

sample

```
^self"A sample method comment that
    is not very lengthy but does not
    say anything."
```

VALUE = 4

sample

```
^self  "A sample method comment that
        is not very lengthy but does not
        say anything."
```

8.17.2.2 Comment

This option specifies the number of spaces to indent a comment in addition to the current indentation level. If it is possible, spaces are automatically replaced by the appropriate number of tabs.

VALUE= 0

sample

```
"A sample method comment that
is not very lengthy but does not
say anything."
^self
```

VALUE = 4

sample

```
"A sample method comment that
is not very lengthy but does not
say anything."
^self
```

8.17.2.3 Comment Continuation

This option specifies the number of spaces to indent a comment continuation in addition to the current indentation level. If it is possible, spaces are automatically replaced by the appropriate number of tabs.

VALUE= 1

sample

```
"A sample method comment that is not very lengthy but does
not say anything . It does however show indentation."
```

```
^self
```

VALUE = 4

sample

```
"A sample method comment that is not very lengthy but does
not say anything . It does however show indentation."
```

```
^self
```

8.17.3 Formatting Margins

This section describes the options for formatting margins.

8.17.3.1 Maximum Width

This option specifies the maximum width (in characters) of a line. See also the related option **Use maximum width** in the General category.

8.17.3.2 Minimum Width for Comments

This option specifies the minimum width (in characters) to print comments at the end of a line of code. If necessary, the margin (maximum width) is extended to fit the given number of characters of comments.

VALUE= 10

(Maximum width = 20)

sample

```
^self  "A sample
comment
at the
end of a
line of
code
that is
not very
lengthy
but does
not say
anything."
```

```
VALUE = 20  
(Maximum width = 20)
```

sample

```
^self  "A sample comment at  
       the end of a line  
       of code that is  
       not very lengthy  
       but does not say  
       anything."
```

8.18 Saving and Loading Your Settings

The **Code Formatter Settings** dialog lets you save and load settings.

To save the currently enabled formatter settings:

1. Open the **Code Formatter Settings** dialog from any **Edit** → **Format Settings...** menu item or select **Tool Settings** → **Formatter...** from the Transcript.
2. Click **Save To File...**
3. Enter the name of the file to which to save the settings.

To load settings from a file:

1. Open the **Code Formatter Settings** dialog from any **Edit** → **Format Settings...** menu item or select **Tool Settings** → **Formatter...** from the Transcript.
2. Click **Load From File...**
3. Enter the name of the file that contains your settings. The loaded settings override any previous settings.

9 Troubleshooting

This chapter describes solutions to some of the problems you may encounter when you are using **ENVY/QA**.

Published RTF documents are incorrectly formatted (for example, tables are too narrow)

Verify that OTIMLRTF.DOT is installed in your Microsoft Word user templates directory. (To find the file location of the user templates, from Microsoft Word select **Tool** → **Options**.)

Some anti-virus software for the Microsoft Word Concept virus overrides the auto-open macros. For the purpose of publishing your code, you may need to use a NORMAL.DOT that does not have the anti-virus software installed.

Formatting rules do not always seem to work

Some formatting rules are applied only when lines become too wide.

Some of the browsers do not have a Tool menu

Only newly opened browsers will have the menu option.

Code Coverage report percentages do not add up to 100%

The numbers shown in the output are approximations, and some round-off may occur.

Code Critic or Code Metric cannot be unloaded

Close all open results browsers.

Code Coverage cannot be unloaded

Close all open coverage browsers.

Bibliography

Charles F. Goldfarb, *The SGML Handbook*, Oxford University Press, 1994.

Donald E. Knuth, *The TeXbook*, Addison-Wesley Publishing, 1986.

Leslie Lamport, *LaTeX: A Document Preparation System*, Addison-Wesley Publishing, 1986.

Mark Lorenz and Jeff Kidd, *Object-Oriented Software Metrics*, Prentice Hall, 1994.

Suzanne Skublics, Edward J. Klimas, and David A. Thomas, *Smalltalk with Style*, Prentice Hall, 1996.

Index

A

- adding a comment
 - Code Coverage, 89
 - Code Critic, 37
 - Code Metrics, 71
- adding metrics, 74
- adding reviews, 41
- advanced settings dialog, 66
- application metrics, 63
- Applications Browser, 38, 71
- applications menu, 89

C

- classes menu, 92
- Code Coverage browser, 82, 84, 85, 86, 89
- code coverage percentages, 134
- Code Critic advanced settings dialog, 29
- Code Critic options dialog, 12, 16
- Code Critic results browser, 12, 33, 36, 38
- Code Critic settings dialog, 16, 32
- Code Formatter settings dialog, 109, 112, 131
- Code Metrics options dialog, 50, 52, 53
- Code Metrics results browser, 50, 67, 68, 71
- Code Metrics settings dialog, 53
- Code Publisher output options dialog, 99, 103
- Code Publisher settings dialog, 102
- coverage menu, 83, 85
- coverage menus, 89
- coverage setup, 88
- customizing metrics, 63
- customizing reviews, 28

E

- export results to spreadsheet
 - Code Coverage, 88
- exporting results to text
 - Code Coverage, 88
- exporting to spreadsheet
 - Code Critic, 37
 - Code Metrics, 71
- exporting to text

- Code Critic, 37
- Code Metrics, 70

F

- format menu, 109
- formatting
 - applications, 110
 - classes, 110
 - methods, 111
 - source, 111
- formatting options, 127

H

- hide ignored, 87
- hiding in-range results, 69
- HTML, 100, 101

I

- ignore sets, 16, 35, 36
- ignoring components, 86
- ignoring results, 35, 40

L

- LaTeX, 100, 101
- loading
 - Code Coverage, 80
 - Code Critic, 10
 - Code Formatter, 108
 - Code Metrics, 48
 - Code Publisher, 96
- loading coverage setup, 88
- loading ignore sets, 36
- loading results
 - Code Critic, 36
 - Code Metrics, 70
- loading settings
 - Code Critic, 32
 - Code Formatter, 131
 - Code Metrics, 67
 - Code Publisher, 106
- lower threshold, 63

M

- measuring
 - applications, 51
 - classes, 52
 - configuration maps, 51
 - methods, 52
- menus
 - Code Critic, 38
- methods menu, 92
- metrics menu, 50, 72
- metrics menus, 72
- metrics properties, 64
- MIF, 100, 101
- missing tool menu, 133
- modifying properties, 31, 65

O

- OTIML, 100, 102
- output formats, 100

P

- pause button, 86
- printing results
 - Code Coverage, 89
 - Code Critic, 37
- problems unloading, 134
- problems with tool menu, 133
- publishing
 - applications, 98
 - classes, 99
 - configuration maps, 98
- publishing options, 104, 105

R

- refreshing coverage browser, 86
- refreshing results, 36, 69
- reporting results, 37, 70, 88
- restoring components, 87
- result menu
 - Code Critic, 39
 - Code Metrics, 73
- review menu, 39
- review properties, 29
- reviewing
 - applications, 14
 - classes, 15

- configuration maps, 14
- methods, 15

- RTF, 100, 102, 133
- RTF table problems, 133
- running reviews, 12, 16

S

- save value, 32, 66, 127
- saving coverage setup, 88
- saving ignore sets, 36
- saving results
 - Code Critic, 36
 - Code Metrics, 70
- saving settings
 - Code Critic, 32
 - Code Formatter, 131
 - Code Metrics, 67
 - Code Publisher, 106
- selecting applications to watch, 81
- severity level, 28
- SGML, 100
- show all, 87
- status area, 85
- summary info dialog
 - Code Coverage, 90
 - Code Critic, 38
- system menu, 10, 34, 48, 80, 96, 108

T

- tool menu problems, 133
- troubleshooting, 133

U

- unloading
 - Code Coverage, 80
 - Code Critic, 10
 - Code Formatter, 108
 - Code Metrics, 48
 - Code Publisher, 96
- upper threshold, 63

W

- watch button, 86
- watching
 - applications, 83
 - configuration maps, 83