



# **goanna *central***

---

USER GUIDE – VERSION 3.3.2  
LINUX/WINDOWS EDITION

September 5, 2014

© 2008-2014 Red Lizard Software

---

Copyright © 2008-2014 Red Lizard Software  
All rights reserved.

---

This document, as well as the software described in it, is provided under license and may only be used or copied in accordance with the terms of such license. The information contained herein is the property of NICTA and is made available under license to Red Lizard Software. It is confidential information as between the Recipient and Red Lizard Software and remains the exclusive property of NICTA. No part of this documentation may be copied, translated, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior written permission of NICTA.  
NICTA does not warrant that this document is error-free.

---

Red Lizard Software  
Australian Technology Park  
Level 5, 13 Garden Street  
Eveleigh NSW 2015  
Australia

Web: <http://www.redlizards.com>  
Support: [support@redlizards.com](mailto:support@redlizards.com)

---

# Contents

<b>1</b>	<b>System Requirements</b>	<b>6</b>
1.1	Operating Systems	6
1.1.1	Microsoft Windows	6
1.1.2	Linux	6
1.1.3	Other Requirements	6
1.2	Hardware Requirements	8
1.3	Supported Compilers	8
1.3.1	A Word On C99 and C++11 Support	8
1.3.2	A Word On Compiler-Specific Syntax Extensions	9
1.3.3	Analog Devices CrossCore C/C++ Compiler (cc21x Dialect)	9
1.3.4	ARM C/C++ Compiler (armcc Dialect)	9
1.3.5	Cosmic Software C Cross Compiler (cosmic Dialect)	9
1.3.6	Cygwin GCC (cygwin Dialect)	9
1.3.7	Freescale (metrowerks Dialect)	10
1.3.8	GNU C/C++ Compiler (GCC) (gnu Dialect)	10
1.3.9	IAR Toolchain for 8051, ARM and MSP430 (iar-8051, iar-arm and iar-msp430 Dialects)	10
1.3.10	Keil Cx51 and C166 Optimizing C Compiler (c51 and c166 Dialects)	10
1.3.11	Microsoft Visual C++ (microsoft Dialect)	10
1.3.12	QNX QCC (qnx Dialect)	11
1.3.13	Renesas H8S, H8/300 Series C/C++ Compiler (renesas-h8 Dialect)	11
1.3.14	Renesas RXC Toolchain (renesas-rx Dialect)	11
1.3.15	Tasking VX-toolset for C166/ST10 (tasking-c166 Dialect)	11
1.3.16	TI Build Tools (ti-cl16x, ti-cl2000, ti-cl430, ti-cl470, ti-cl500 and ti-cl55 Dialects)	12
1.3.17	Wind River Diab Compiler (diab Dialect)	12
1.4	Supported Build Systems	12
1.4.1	GNU Make, QNX Make and Microsoft NMake	13
1.4.2	SCons	13
1.4.3	CMake	13
1.4.4	IAR Embedded Workbench <sup>®</sup> (IarBuild.exe)	13
1.4.5	Keil <sup>™</sup> $\mu$ Vision <sup>®</sup> (UV4.exe)	13
<b>2</b>	<b>Getting Started</b>	<b>14</b>
2.1	License Agreement	14
2.2	Installation (Linux)	14
2.3	Installation (Windows)	15
2.4	License Activation	15
2.4.1	Activating Node-locked License	15
2.4.2	Using Network (Floating) License	16
2.5	Next Steps	16

<b>3</b>	<b>Setting Up Projects for Goanna Analysis</b>	<b>18</b>
3.1	Introduction	18
3.2	Setting Up GNU Make, QNX Make and Microsoft NMake Projects with goannamake	19
3.2.1	Preparing Makefile for Goanna Integration	19
3.2.2	Using goannamake To Capture The Build Settings	20
3.3	Setting Up SCons Projects with goannascons	21
3.3.1	Preparing SConstruct file for Goanna Integration	21
3.3.2	Using goannascons To Capture The Build Settings	22
3.4	Setting Up CMake Projects with goannacmake-conv	23
3.4.1	Using goannacmake-conv To Generate The Build Settings	23
3.5	Setting Up IAR Embedded Workbench® Projects with goannaiarbuild	24
3.6	Setting Up Keil™ μVision® Projects with gotrace	25
<b>4</b>	<b>Running Goanna Analysis</b>	<b>26</b>
4.1	Introduction	26
4.2	Running Goanna Analysis On Non-Keil Projects With goanna	27
4.3	Running Goanna Analysis On Keil™ μVision® Projects With gokeil	27
4.4	Reading Analysis Results	28
4.4.1	Goanna Output On The Console	28
4.4.2	HTML Report of Analysis Results	29
4.4.3	Analysis Results In XML File	30
4.4.4	Using Goanna Dashboard Web Interface To Interact With Analysis Results	30
4.4.5	Using SonarQube To Interact With Analysis Results	30
<b>5</b>	<b>Configuring Goanna Analysis</b>	<b>31</b>
5.1	Setting Checks	31
5.1.1	Introduction	31
5.1.2	Selecting Checks With Command Line Options	31
5.1.3	Using Checks File To Select Multiple Checks	32
5.1.4	Enabling All Available Checks	32
5.2	Checks Packages	33
5.2.1	Listing Checks Packages	33
5.2.2	Enabling Available Checks Package	33
5.2.3	Disabling Installed Checks Package	34
5.2.4	Installing Custom Checks Package	34
5.3	Including Headers Into Analysis	34
5.4	Excluding Certain Files From Analysis	34
5.5	Setting Analysis Timeouts	34
5.6	Ignoring Certain Warnings ("Warning Suppression")	35
5.7	Other Configuration Options	35
<b>6</b>	<b>Running Goanna Analysis From Within IDEs</b>	<b>36</b>
6.1	Running Goanna Analysis From IAR Embedded Workbench®	36
6.2	Running Goanna Analysis From Keil™ μVision®	37

<b>7</b>	<b>Getting the Best Results from Goanna</b>	<b>40</b>
7.1	Interprocedural Analysis	40
7.2	A Word on False Positives	40
7.3	Using the <code>_GOANNA</code> Preprocessor Symbol	41
7.4	Using the <code>assert</code> macro	41
7.5	Sample Code	41
<b>8</b>	<b>Using the Goanna Dashboard</b>	<b>42</b>
8.1	Getting to the Goanna Dashboard	42
8.2	Bug Statuses	42
8.3	Severity	42
8.4	Dashboard Views	42
8.4.1	Project Page	42
8.4.2	Report Page	43
8.4.3	Directory Browser	44
8.4.4	Warnings Browser	45
8.4.5	Code Browser	46
8.5	Database Upgrades	46
8.6	Project Settings (Advanced)	48
8.6.1	External Code Browser Support	48
8.6.2	Code Browser Character Encodings	48
<b>9</b>	<b>Using Goanna With SonarQube Code Quality Platform</b>	<b>49</b>
9.1	Introduction	49
9.2	Setting Up Goanna To Integrate With SonarQube Installation	49
9.3	Running Goanna Analysis With SonarQube Publish	49
<b>10</b>	<b>Advanced Features, Concepts and Configurations</b>	<b>51</b>
10.1	Proceed With Caution	51
10.2	Manually Running Analysis On Source Files	51
10.3	Manually Running Link Time Analysis On Object Files	52
10.4	Build And Run Analysis On A Project At The Same Time	52
10.5	Using Embedded Build Information To Perform Analysis	53
10.6	Append New Build Information Into Existing Build Specification	53
10.7	The Project Database	55
10.8	How Goanna's Compiler Support Work	56
10.9	Suppressing Warnings Manually Using <code>goannacc</code>	57
10.9.1	Suppressing Warnings	57
10.9.2	Un-suppressing Warnings	57
10.9.3	Displaying Warnings Status	57
10.10	Using Goanna Central Bundled SonarQube Installation	58
10.10.1	Running Goanna Central Bundled SonarQube On Linux	58
10.10.2	Running Goanna Central Bundled SonarQube On Windows	58
10.10.3	Browsing Goanna Central Bundled SonarQube Dashboard	58
10.10.4	Running Goanna Analysis With Goanna Central Bundled SonarQube	59
10.11	Using CppCheck And CppNcss With Goanna	60

<b>11 Goanna Central Utility Reference</b>	<b>61</b>
11.1 goanna – Analyze C/C++ Projects . . . . .	61
11.2 goannamake – Analyze Makefile Projects . . . . .	63
11.3 goannascons – Analyze SCons Projects . . . . .	65
11.4 goannacmake-conv – Convert CMake Compilation Database To Goanna Build Specification . . . . .	67
11.5 goannaiarbuild – Analyze IAR Embedded Workbench Projects . . . . .	69
11.6 gotrace and gokeil – Analyze Keil™ $\mu$ Vision® Projects . . . . .	71
11.7 goannacc and goannac++ – Analyze C/C++ Source Files . . . . .	73
11.8 goannaId – C/C++ Link Time Analysis . . . . .	78
11.9 goreporter – Goanna Dashboard Server and Administration Tool, and Publish Analysis Results . . . . .	81

Index5

# 1 System Requirements

Before using Goanna, please check that your system and project meets the system requirements.

## 1.1 Operating Systems

### 1.1.1 Microsoft Windows

Goanna supports the following versions of Windows:

- Windows XP (Service Pack 2 or higher)
- Windows Vista
- Windows 7
- Windows 8
- Windows 8.1
- Windows Server 2003 (Service Pack 1 or higher)
- Windows Server 2008
- Windows Server 2008 R2
- Windows Server 2012
- Windows Server 2012 R2

Both 32-bit (x86) and 64-bit (x86-64/AMD64) versions of Windows are supported (except Windows XP, which we only support 32-bit version).

### Required Software

Before installing Goanna, you will need to install the following:

- Microsoft Visual C++ 2008 Redistributable (Download from <http://www.microsoft.com/en-us/download/details.aspx?id=5582>)
- .NET Framework 2.0 or higher (Goanna Central installer automatically installs .NET Framework 4.0 if not installed already)

### 1.1.2 Linux

Goanna supports all major distributions of Linux with glibc (GNU C Library) 2.4 or higher installed.

Both 32-bit (x86) and 64-bit (x86-64/AMD64) versions of Linux are supported.

Using Goanna with SELinux enabled is not recommended.

### 1.1.3 Other Requirements

Some features of Goanna may require additional software or packages.

### Requirements for Goanna Dashboard and HTML Report

Goanna Dashboard (see 8) and HTML Report (see 4.4.2) requires a web browser. The following web browsers are supported:

- Internet Explorer 9 or higher
- Mozilla Firefox - currently supported versions by Mozilla
- Google Chrome - currently supported versions by Google

We also support Internet Explorer 7 and 8, however you may experience slow performance on these browsers; using Goanna Central with these browsers is not recommended.

## Requirements for LM-X License Manager

Goanna uses LM-X License Manager 4.4.2 from X-Formation for licensing.

If you wish to use web-based UI of the License Manager (**Important:** Red Lizard Software does not provide full support for the web-based UI), the following software must be installed:

- A modern web browser
- Oracle Java Runtime Environment 1.6 or higher
- Adobe Flash Player

Please refer to X-Formation website (<http://docs.x-formation.com/display/GEN/System+requirements+for+web-based+UIs>) for more information.

## Requirements for CppCheck

**Important:** CppCheck is a third party component bundled with Goanna Central. Red Lizard Software does not provide full support for CppCheck.

Goanna Central bundles CppCheck 1.56 (for more information, see <http://cppcheck.sourceforge.net/>).

If you wish to use CppCheck on Linux, then libstdc++ 3.4.14 or higher is required. This means that on some Linux distributions (such as Ubuntu 10.04 LTS), CppCheck may not run even when the rest of Goanna analysis was performed successfully.

No additional software or packages are required to use CppCheck on Windows.

Because CppCheck uses different C/C++ parser from the rest of Goanna, some C/C++ source files may not be correctly understood by CppCheck even when the rest of Goanna analysis understands these files.

## Requirements for CppNcss

**Important:** CppNcss is a third party component bundled with Goanna Central. Red Lizard Software does not provide full support for CppNcss.

Goanna Central bundles CppNcss 1.0.3 (for more information, see <http://cppncss.sourceforge.net/>).

No additional software or packages are required to use CppNcss.

Because CppNcss uses different C/C++ parser from the rest of Goanna, some C/C++ source files may not be correctly understood by CppNcss even when the rest of Goanna analysis understands these files.

## Requirements for SonarQube Integration

**Important:** SonarQube is a third party component bundled with Goanna Central. Red Lizard Software does not provide full support for SonarQube.

Goanna Central comes with an ability to publish the analysis result to SonarQube (previously called "Sonar"). For more details, see <http://www.sonarqube.org/>.

Goanna Central supports SonarQube integration with any existing installations of SonarQube 3.2.1 and higher, except for versions 3.5 and 3.5.1.

SonarQube integration requires no additional software or packages to be installed, however SonarQube itself may have additional requirements. Please refer to SonarQube website for requirements if you wish to install your own instance of SonarQube (<http://docs.codehaus.org/display/SONAR/Requirements>).

Goanna Central also ships with the Goanna customized version of SonarQube 3.2.1.



## 1.2 Hardware Requirements

Goanna requires, at the minimum, the following hardware:

- Processor: Intel Pentium 4 or higher
- Memory: 1 GB or more
- Storage: Minimum 1 GB of free disk space

For optimal analysis performance, we recommend at least the following:

- Processor: Intel Core 2 Duo or later CPU with minimum speed 2 GHz. Multi core CPUs are recommended.
- Memory: 4 GB or more
- Storage: 5 GB or more of free disk space

For large projects, Goanna may require more RAM and disk space than the ones shown here.

## 1.3 Supported Compilers

Goanna currently supports the following compilers:

Compiler Name	Goanna Dialect Name	Common Compiler Executables
Analog Devices CrossCore C/C++ Compiler	cc21x	cc21k
ARM C/C++ Compiler	armcc	armcc, armlink
Cosmic Software C Cross Compiler	cosmic	cx6808, cx6812, cx6816, cxs12x, cxstm8
Cygwin GCC	cygwin	gcc, g++, ld
Freescale (formerly Metrowerks) GNU	metrowerks gnu	mwccarm, mwccmcf gcc, g++, ld
IAR Toolchain for 8051 - 6.x	iar-8051	icc8051
IAR Toolchain for ARM - 6.x	iar-arm	iccarm
IAR Toolchain for MSP430 - 6.x	iar-msp430	icc430
Keil Cx51 Optimizing C Compiler	c51	c51, cx51, bl51, lx51
Keil C166 Optimizing C Compiler	c166	c166, l166
Microsoft Visual C++	microsoft	cl, link
QNX QCC	qnx	qcc
Renesas H8S, H8/300 Series C/C++ Compiler	renesas-h8	ch38
Renesas RXC Toolchain	renesas-rx	ccrx
Tasking VX-toolset for C166	tasking-c166	cc166
TI Build Tools - CL16X	ti-cl16x	cl6x
TI Build Tools - CL2000	ti-cl2000	cl2000
TI Build Tools - MSP430	ti-cl430	cl430
TI Build Tools - CL470	ti-cl470	cl470
TI Build Tools - CL500	ti-cl500	cl500
TI Build Tools - CL55	ti-cl55	cl55
Wind River Diab Compiler	diab	dcc

### Notes

- Green Hills compiler is no longer supported since Goanna 3.1.0.
- Goanna also ships with `gnu-4.4.4` and `ti` dialects. However, these dialects exist only for backward compatibility. We strongly recommend that you do not use these dialects.

### 1.3.1 A Word On C99 and C++11 Support

Goanna strives to support most C99 and C++11 features as long as the compilers used in your projects also accept them. However, please note that, for C++11 extensions, the analysis engine generally does not make use of these extensions or any information derived from usage of these extensions. For example, Goanna does not perform any pointer or memory use related analysis on C++11 `std::shared_ptr`.

Also note that CppCheck or CppNcss use different C/C++ parsers, and may not support some C99 and C++11 features (even including those supported by Goanna analysis engine). Notably, CppNcss does not support most C++11 extensions (such as range-based for loops).

### 1.3.2 A Word On Compiler-Specific Syntax Extensions

Goanna strives to support most compiler-specific C/C++ syntax extensions for supported compilers. However, please note that even in cases where Goanna supports compiler-specific C/C++ syntax extensions, the analysis engine will generally not make use of these extensions or any information derived from usage of these extensions. For example, Goanna does not take Keil Cx51 Memory Models or Memory Types into account during analysis, even though Goanna supports relevant syntax extensions.

Also note that CppCheck or CppNess use different C/C++ parsers, and do not support any compiler-specific C/C++ syntax extensions.

### 1.3.3 Analog Devices CrossCore C/C++ Compiler (cc21x Dialect)

Goanna supports Analog Devices CrossCore C/C++ Compiler 1.0 or higher.

### 1.3.4 ARM C/C++ Compiler (armcc Dialect)

Goanna supports the following versions of ARM C/C++ Compiler:

- RealView Development Suite (RVDS) versions 2.0 to 4.1 (inclusive)
- DS-5 Development Studio
- ARM Compiler versions 4.1 and 5.0 (including those shipped with Keil MDK-ARM versions 4 and 5)

RVDS 1.2 and older, ARM Developer Suite (ADS) versions 1.2 and older, and ARM Compiler 6 (armclang) are not supported.

### Known Limitations

- Some compiler arguments, such as `--cpp` (for `--cpp` use `--c++` instead), `--kandr_include`, `--strict`, `-wchar`, `--no_wchar`, `--wchar16` and `--wchar32` are not supported. Goanna will ignore these arguments.
- Goanna will always include `RVCT<version>INC`, `ARMCC<version>INC` and `ARMINC` environment variables, and `<installation-path-of-compiler>\..\include` into the system include directories, even when `-J` option is used.

### 1.3.5 Cosmic Software C Cross Compiler (cosmic Dialect)

Goanna supports all recent versions of:

- CX6808 Compiler (part of Cosmic S08 and HC08 Cross Development Tools)
- CX6812 Compiler (part of Cosmic 68HC12 and HCS12 Cross Development Tools)
- CX6816 Compiler (part of Cosmic 68HC16 Cross Development Tools)
- CXS12X Compiler (part of Cosmic S12X and XGATE Cross Development Tools)
- CXSTM8 Compiler (part of Cosmic STM8 Cross Development Tools)

**Note:** CXXGATE Compiler (part of Cosmic S12X and XGATE Cross Development Tools) is not supported.

### 1.3.6 Cygwin GCC (cygwin Dialect)

Goanna supports most versions of Cygwin GCC (GNU C/C++ Compiler).

Goanna does not automatically detect usage of Cygwin GCC. You need to manually specify `cygwin` dialect to ensure all source files are properly understood by Goanna.

Any limitations that apply to GNU C/C++ Compiler generally (with `gnu` dialect) also apply to Cygwin GCC.

## Known Limitations

- Before using Goanna, you need to make sure that either:
  - The path to the Cygwin installation directory is in the PATH environment variable, or
  - Cygwin is installed to the 32-bit default location of C:\Cygwin.

Goanna assumes that the compiler is located in bin subdirectory of the above path. Parse errors may occur if the compiler is installed to any other location.

### 1.3.7 Freescale (metrowerks Dialect)

Goanna supports mwccarm compiler shipped with Freescale CodeWarrior Development Studio for Microcontrollers (CW MCU) version 10.2 *only*.

**Note:** Goanna supports only this particular version of CW MCU. In particular, CW MCU versions 10.3 and higher are not supported.

- Goanna assumes the compiler is located at:
    - C:\Freescale\CW MCU v10.2\MCU\ARM\_Tools\Command\_Line\_Tools.
- Parse errors may occur if the compiler is installed to any other location.

### 1.3.8 GNU C/C++ Compiler (GCC) (gnu Dialect)

Goanna supports most versions of GNU C/C++ Compiler (GCC).

You need to use cygwin dialect if your project uses Cygwin version of GCC.

## Known Limitations

- CPATH, C\_INCLUDE\_PATH and CPLUS\_INCLUDE\_PATH environment variables are not recognized by Goanna.
- --sysroot option is not supported.

### 1.3.9 IAR Toolchain for 8051, ARM and MSP430 (iar-8051, iar-arm and iar-msp430 Dialects)

Goanna supports IAR Toolchain for 8051, ARM and MSP430 shipped with IAR Embedded Workbench 5.40 or higher.

### 1.3.10 Keil Cx51 and C166 Optimizing C Compiler (c51 and c166 Dialects)

Goanna supports all recent versions of Keil Cx51 and C166 Optimizing C Compiler.

### 1.3.11 Microsoft Visual C++ (microsoft Dialect)

Goanna supports Microsoft Visual C++ compiler shipped with Microsoft Visual Studio 6.0 or higher.

## Known Limitations

- Goanna assumes the system include path to be *<installation-path-of-compiler>\include*. If your project uses additional system include directories (such as when using MFC or ATL headers), you will need to manually specify these directories.
- Goanna does not recognize INCLUDE environment variable.
- Managed C++, C++/CLI syntax extensions, and CLR (Common Language Runtime) related compiler options are not supported.

### 1.3.12 QNX QCC (qnx Dialect)

Goanna supports QNX QCC shipped with QNX Momentics 4.7.0 or higher.

### 1.3.13 Renesas H8S, H8/300 Series C/C++ Compiler (renesas-h8 Dialect)

Goanna supports ch38 compiler shipped with Renesas C/C++ Compiler Package for H8SX, H8S, H8 Family (also called "H8S, H8/300 Series C/C++ Compiler") version 7.00 Release 00 *only*.

**Note:** Goanna supports only this particular version of ch38 compiler. In particular, ch38 compiler version 6 and older are not supported.

#### Known Limitations

- Goanna assumes the compiler is located at:
  - C:\Program Files (x86)\Renesas\Hew\Tools\Renesas\H8\7\_0\_0\bin.Parse errors may occur if the compiler is installed to any other location.

### 1.3.14 Renesas RXC Toolchain (renesas-rx Dialect)

Goanna supports CC-RX compiler shipped with Renesas C/C++ Compiler Package for RX Family version 1.02.01 *only*.

**Note:** Goanna supports only this particular version of CC-RX compiler. In particular, CC-RX compiler version 2 is not supported.

#### Known Limitations

- Goanna assumes the compiler is located at:
  - C:\Renesas\e2studio\Tools\Renesas\RX\1\_2\_1\bin.Parse errors may occur if the compiler is installed to any other location.

### 1.3.15 Tasking VX-toolset for C166/ST10 (tasking-c166 Dialect)

Goanna supports Tasking VX-toolset for C166/ST10 version 3.1r1 *only*.

**Note:** Goanna supports only this particular version of VX-toolset. In particular, VX-toolset for C166/ST10 version 3.1r2 is not supported.

#### Known Limitations

- Goanna assumes the compiler is located at:
  - C:\Program Files (x86)\TASKING\C166-VX v3.1r1\bin.Parse errors may occur if the compiler is installed to any other location.

### 1.3.16 TI Build Tools (ti-cl16x, ti-cl2000, ti-cl430, ti-cl470, ti-cl500 and ti-cl55 Dialects)

Goanna supports the following Texas Instruments compilers:

- CL16X compiler shipped with Texas Instruments Code Composer Studio versions 2.2, 3 and 5,  
**Note:** CL16X compiler shipped with Texas Instruments Code Composer Studio version 4 is not supported.
- CL2000 compiler shipped with Texas Instruments Code Composer Studio versions 4 and 5,
- CL430 compiler shipped with Texas Instruments Code Composer Studio versions 4 and 5,
- CL470 compiler shipped with Texas Instruments Code Composer Studio version 5,
- CL500 compiler shipped with Texas Instruments Code Composer Studio version 5, and
- CL55 compiler shipped with Texas Instruments Code Composer Studio version 5.

**Note:** Compilers shipped with Texas Instruments Code Composer Studio version 6 are not supported.

#### Known Limitations

- Goanna assumes the compiler is located at:
  - C:\ti\ccsv5\tools\compiler\c6000\_7.3.4\bin,
  - C:\ti\ccsv5\tools\compiler\c2000\_6.1.0\bin,
  - C:\ti\ccsv5\tools\compiler\c5400\_4.2.0\bin,
  - C:\ti\ccsv5\tools\compiler\c5500\_4.4.1\bin,
  - C:\ti\ccsv5\tools\compiler\msp430\_4.1.0\bin,
  - C:\ti\ccsv5\tools\compiler\tms470\_4.9.1\bin,
  - C:\ti\ccsv4\tools\compiler\c2000\bin,
  - C:\ti\ccsv4\tools\compiler\msp430\bin,
  - C:\CCStudio\_v3.3\C6000\cgtools\bin, or
  - C:\ti\c6000\cgtools\bin.Parse errors may occur if the compiler is installed to any other location.
- Goanna recognizes compiler intrinsics for CL2000 compiler only (ti-cl2000 dialect).  
**Note:** For CL2000 compiler, Goanna ignores all compiler intrinsics; they are not used for analysis.

### 1.3.17 Wind River Diab Compiler (diab Dialect)

Goanna supports Wind River Diab Compiler shipped with Wind River Workbench 3.3 or higher.

#### Known Limitations

- Goanna assumes the compiler is located at:
  - C:\WindRiver\diab\5.9.0.0\WIN32\bin, or
  - C:\WindRiver\diab\5.8.0.0\WIN32\bin.Parse errors may occur if the compiler is installed to any other location.

## 1.4 Supported Build Systems

Goanna currently supports the following build systems:

Build System Name	Goanna Utility Name
GNU Make	goannamake
QNX Make	goannamake
Microsoft NMake	goannamake
SCons	goannascons
CMake	goannacmake-conv
IAR Embedded Workbench (IarBuild.exe)	goannaiarbuild
Keil $\mu$ Vision (UV4.exe)	gotrace

#### 1.4.1 GNU Make, QNX Make and Microsoft NMake

Goanna supports most versions of GNU Make, QNX Make and Microsoft NMake.

Goanna requires your `Makefile` to be written in specific style, and manual modification may be necessary to integrate Goanna into your build. For instructions of how to do so, see [3.2](#).

#### 1.4.2 SCons

Goanna supports most versions of SCons.

Goanna requires your `SConstruct` to be written in specific style, and manual modification may be necessary to integrate Goanna into your build. For instructions of how to do so, see [3.3](#).

#### Known Limitations

- Link time analysis will not be performed if the build calls linker directly, rather than performing linking through the compiler.

#### 1.4.3 CMake

Goanna supports the following versions of CMake:

- CMake 2.8.5 or higher for Unix Makefile generator projects
- CMake 2.8.9 or higher for Ninja generator projects

**Note:** CMake projects with any other generators (such as Visual Studio) are not supported.

Goanna does not support direct integration with CMake build. However, Goanna provides a utility to read the build information from the compilation database generated by CMake.

#### Known Limitations

- CMake projects with MinGW GCC compiler are not supported.

#### 1.4.4 IAR Embedded Workbench® (IarBuild.exe)

Goanna supports command line build of IAR Embedded Workbench projects using `IarBuild.exe` shipped with IAR Embedded Workbench version 5.40 or higher.

Goanna also provides a mechanism to run analysis from within the IDE. For more details, see [6.1](#).

#### 1.4.5 Keil™ μVision® (UV4.exe)

Goanna supports command line build of Keil μVision projects with Keil μVision 4 or higher.

Goanna also provides a mechanism to run analysis from within the IDE. For more details, see [6.2](#).

#### Important Notes

- Support for Keil μVision is provided in a separate Keil μVision Support Package. Contact your distributor to obtain the package.
- Old versions of Keil μVision (version 3 and older) are not supported.

## 2 Getting Started

### 2.1 License Agreement

Before installing Goanna Central, ensure you read the Goanna license agreement.

For evaluation (trial) licenses, please refer to:

<http://redlizards.com/license-term/evaluation-license-agreement/>

For registered (paid) licenses, please refer to:

<http://redlizards.com/license-term/>

### 2.2 Installation (Linux)

To install Goanna Central for Linux:

1. Download the Goanna Central for Linux tarball.
2. Unpack the downloaded tarball:

```
tar -zxvf goanna-central-linux-release-3.3.2.tgz
```

This should extract all files needed for installation in a separate directory.

3. Navigate to the directory just created:

```
cd goanna-central-linux-release-3.3.2
```

4. Run the install script (`install-goanna`) to start the installation process.  
The installation process can be run either with `sudo` (i.e. under root):

```
sudo ./install-goanna
```

or without `sudo` (i.e. under your user account):

```
./install-goanna
```

We recommend that you install Goanna with `sudo`. The following features will not be installed if you do not install Goanna with `sudo`:

- The Goanna Dashboard daemon; if the daemon is not installed, then to access the Goanna Dashboard, you will need to start the server manually. See 8 for details.
5. Follow the instructions of the install script to complete installation.

By default, Goanna Central will be installed to:

- `/usr/local/goanna` if installed with `sudo`, or
- `$HOME/goanna` if installed without `sudo`.

If you wish to install Goanna Central to other location, provide a desired location when the install script asks for the installation path.

6. To use Goanna Central from the command line, you should set your `PATH` environment variable to include Goanna's `bin` directory. The install script will show you how to do this at the end of installation.

For example, if you installed Goanna in `/usr/local/goanna`, you can add:

```
export PATH=$PATH:/usr/local/goanna/bin
```

to your `~/.profile` file.

## 2.3 Installation (Windows)

To install Goanna Central for Windows:

1. Download the Goanna Central for Windows installer.
2. Double click the installer .exe file.
3. **Important:** If you wish to use one of the following features:
  - SonarQube integration support (see 9)
  - Goanna Central bundled SonarQube 3.2.1 (see 10.10)
  - CppCheck (see 10.11)
  - CppNcss (see 10.11)

Then you need to click Options at this screen, and check “Codehaus Sonar” option to ensure all necessary components are installed.

Additionally, you should also click Options here if you wish to change the installation directory.

4. Click Install.
5. Follow the instructions of the installer to complete installation.

**Important:** If you wish to use Goanna Central with Keil  $\mu$ Vision, you will also need to install Keil  $\mu$ Vision Support Package. Contact your distributor to obtain the installer.

## 2.4 License Activation

Whether you are just evaluating Goanna or have purchased the full version, you must activate your license before you can use Goanna.

### 2.4.1 Activating Node-locked License

If you have obtained node-locked licenses, you should have received an email containing your license information from Red Lizard Software. This email will contain an Order Number which is required to complete the activation process.

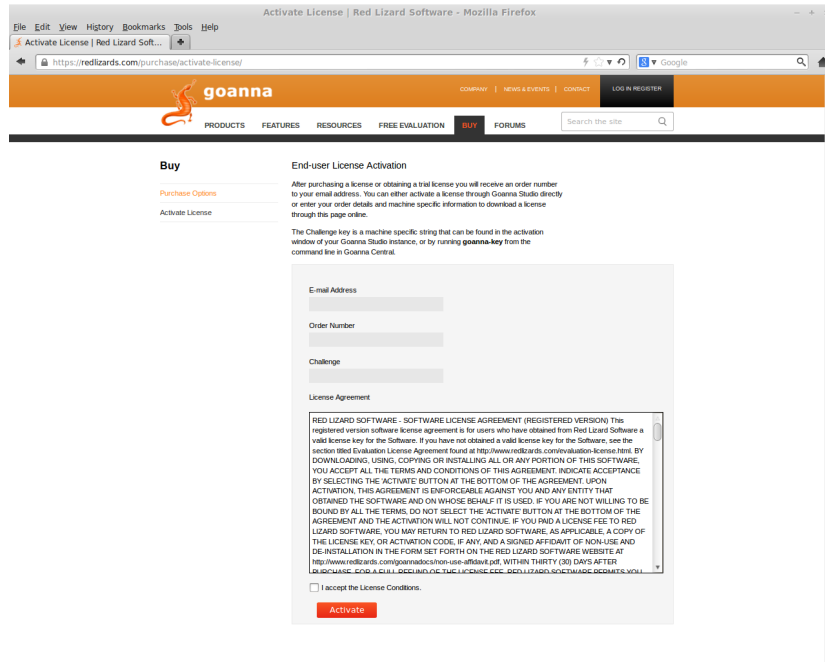
Before activating your node-locked license, you will need a *challenge key* for your computer. To obtain the challenge key, run:

**goanna-key**

Now, to activate your node-locked license, follow these steps:

1. Go to the activation page at: <http://www.redlizards.com/purchase/activate-license/>  
Enter the following details:
  - Email Address:** The email address you provided when purchasing Goanna.
  - Order Number:** The order number provided in the Goanna purchase confirmation email.
  - Challenge:** The challenge key of the computer you wish to activate the license for.
2. Read the license agreement.
3. Check “I accept the License Conditions”, and click “Activate” to accept the license agreement and activate your license.
4. The resulting license file (called `goanna_license.lic`) will be sent to the email address you provided. You will also be taken to a page where you can download the license file by clicking on Download your license.
5. Save the `goanna_license.lic` file to the following location:
  - Linux** \$HOME (Home directory of the user running Goanna), or `/etc/goanna`
  - Windows (XP and Server 2003)** `C:\Documents and Settings\\Local Settings\Application Data\RedLizards\Goanna Central`
  - Windows (Vista, Server 2008 and later)** `C:\Users\\AppData\Local\RedLizards\Goanna Central`  
(where `<username>` is the name of the user account who have installed Goanna Central; usually it is your Windows user name).





## 2.4.2 Using Network (Floating) License

If you have obtained network (floating) licenses, then you will need to perform the following steps to set up Goanna to use your license:

1. Set up a license server somewhere on your network with the license file supplied.  
The license server for Goanna is shipped in a separate installation package. Contact your distributor to obtain the installer.  
For instructions on how to set up the license server, please refer to the separate “License Server User Guide”.
2. Always use **--license-server** option when running Goanna analysis.

Once the license server is up and running, to use the license server, pass the **--license-server=<server>** option to goanna or gokeil utility when running analysis (read on the rest of this user guide for instructions on how to run analysis).

### Borrowing a Network (Floating) license

When you run the Goanna analysis with a network (floating) license, you can optionally specify a borrow duration (in the range of 1 to 24 hours). This is enabled by using the **--license-borrow-hours=<hours>** option in combination with the **--license-server** option, and will force Goanna to reserve a single license seat for you for this amount of time. In addition this technique can be used to improve license validation time, and will allow you to perform Goanna analysis while disconnected from the license server.

## 2.5 Next Steps

Performing analysis on your C/C++ project with Goanna Central is a three-step process:

1. Setting up your C/C++ project, and running a full build on the project using Goanna build integration utility to capture settings of your build.
2. Using this information from full build to run analysis.
3. Reading and interacting with the analysis results.

The rest of this documentation explains this process in detail:

1. The next section, section 3, explains how to set up your C/C++ projects to be used with Goanna, and to capture settings of your build.
2. The section 4 explains how to use this captured settings to run analysis, and to read the analysis results.
3. The section 5 explains what options are available to control the analysis, and how to do so.
4. The section 6 explains ways to set up IAR Embedded Workbench or Keil  $\mu$ Vision IDEs to run analysis directly from within IDE.
5. The section 8 explains how to use the Goanna Dashboard, a web-based interface to navigate and interact with analysis results and issues found within your projects.
6. The section 9 explains how to use Goanna Central with SonarQube quality platform.

## 3 Setting Up Projects for Goanna Analysis

### 3.1 Introduction

The most basic way to perform Goanna analysis on the source code is *project-wide analysis*<sup>1</sup>. Project-wide analysis scans and identifies potential issues within the whole project, based on information about how the project is built.

To perform project-wide analysis, the following steps should be taken:

1. First, you need to ensure that the project compiles successfully with no syntax errors. Goanna analysis engine relies on the source code being syntactically correct.
2. Depending on the build system, you may need to modify your build settings to ensure Goanna detects all of source files. For details, see:
  - 3.2 for GNU Make, QNX Make and Microsoft NMake projects, or
  - 3.3 for SCons projects.
3. In addition, the path to the compiler should be added into PATH environment variable. This is to ensure that Goanna can detect the compiler and its configuration. If the build program or the compiler requires other environment variables to be set appropriately for the build to succeed, these variables should also be set in the same way you would do when you are building the project normally.
4. Clean your build to force a full build. This is to ensure Goanna detects all of source files during build.
5. Run a Goanna build integration utility (different depending on the build system). This utility runs and monitors a full build of the project, and captures all necessary information for Goanna to understand the source files. This process is called *build recording*, and the result of this process is stored in a file called *build specification*.
6. Run the project analysis utility with the generated build specification to perform the analysis.

The generated build specification file can be re-used for future Goanna analysis, so that you do not have to re-run full build before every analysis. However, if the structure or settings of the project changes, such as:

- When a new source file has been added to the project,
- When an existing source file has been removed from the project, or
- When the project build settings themselves change, such as adding new include paths or predefined macros, or upgrading to a newer version of compiler,

then the full build, and build recording process should be performed again to ensure the build specification reflects the changes made to the project.

The working directory where you run Goanna build integration utility from is treated as the *project root directory* by all Goanna utilities.

#### Use The Same Machine For Build And Analysis

To ensure that Goanna has an accurate understanding of the source files, Goanna needs access to not only the source files, but other extra build-specific information such as:

- All used header files
- Configuration of the compiler used

This means that Goanna requires that the machine used to analyze the project also has:

- The same version of the compiler used,
- The same versions of all used libraries, and
- The same version of the build program used.

It is therefore recommended to run analysis on the same machine used to build the project.

---

<sup>1</sup>Goanna also provides other methods to perform analysis. See 10 for details.

## 3.2 Setting Up GNU Make, QNX Make and Microsoft NMake Projects with goannamake

### 3.2.1 Preparing Makefile for Goanna Integration

In order for Goanna to detect source files and the compiler settings, Goanna injects into Make variables to replace all calls to compiler and linker. Because of this mechanism, your Makefile must call compilers and linkers via Make variables, rather than calling them directly.

For example, Goanna will *not* be able to inject the following Makefile target:

```
example.o: example.c
gcc -o $@ $<
```

However, by replacing direct call to gcc with CC Make variable, Goanna will be able to inject this Makefile target:

```
CC=gcc
example.o: example.c
$(CC) -o $@ $<
```

By default, Goanna will replace compiler and linker calls in the following Make variables:

- For C compiler: CC, HOST\_CC and TARGET\_CC
- For C++ compiler: CXX, HOST\_CXX, TARGET\_CXX and CCC
- For linker: LD

If your Makefile uses Make variables to call compilers and linkers, but the variables names are different, then either of the following needs to be done:

- Rename the variables to one of the default variable names listed above, or
- Set GOANNA\_MAKE\_CC, GOANNA\_MAKE\_CXX and/or GOANNA\_MAKE\_LD variables to tell Goanna that the different naming convention is used; see below.

#### GOANNA\_MAKE\_CC, GOANNA\_MAKE\_CXX and GOANNA\_MAKE\_LD Environment Variables

Goanna accepts GOANNA\_MAKE\_CC, GOANNA\_MAKE\_CXX and GOANNA\_MAKE\_LD environment variables to specify which Make variables Goanna should monitor for compiler and linker calls.

These environment variables should contain a list of Make variable names used for compiler or linker calls, separated by a colon. If one or more of these environment variables are not set, Goanna assumes the following default values:

Environment Variable	Default Value
GOANNA_MAKE_CC	CC:HOST_CC:TARGET_CC
GOANNA_MAKE_CXX	CXX:HOST_CXX:TARGET_CXX:CCC
GOANNA_MAKE_LD	LD

#### Notes On Microsoft NMake Projects (Windows Only)

With Microsoft NMake, the build program is generally called nmake. In order for Goanna to detect NMake, you need to either:

- Pass **--microsoft** option to **goannamake**. Using this option causes Goanna to look for nmake program, instead of make; or,
- Set MAKE environment variable to ensure Goanna looks for nmake program; see below.

## Using MAKE Environment Variable To Specify Make Program Name

By default, Goanna assumes that the "Make" program is called `make` (unless `--microsoft` option is given). If this is not the case, then you need to set MAKE environment variable to specify which program Goanna should refer to.

MAKE environment variable should contain a name of the program, or an absolute path, to GNU Make, QNX Make or Microsoft NMake program.

### 3.2.2 Using goannamake To Capture The Build Settings

Once your `Makefile` is ready, you can run the build and capture the build settings; `goannamake` utility is used to perform this process.

The steps to run the build and capture the build settings are as follows:

1. Run `make clean`, or other suitable command to "clean" the project.
2. For GNU Make and QNX Make projects, run:

```
goannamake --record=<build-specification-file> <arguments-to-make>
```

For Microsoft NMake projects, run:

```
goannamake --record=<build-specification-file> --microsoft <arguments-to-make>
```

This command performs and monitors the full build; once the build is complete, `goannamake` will output the build settings into `<build-specification-file>`. This file is called *build specification file*.

For example, if your project is GNU Make project, and the build command is `make all`, then you can run the following to save a build specification to `myproject.goannabuild`:

```
goannamake --record=myproject.goannabuild all
```

3. You can now use `goanna` utility to run analysis on the project; see [4.2](#) for more details.

## 3.3 Setting Up SCons Projects with goannascons

### 3.3.1 Preparing SConstruct file for Goanna Integration

In order for Goanna to detect source files and the compiler settings, Goanna injects into SCons build variables to replace all calls to compiler and linker. In addition, Goanna also relies on the system environment variables (such as USERPROFILE on Windows) being set during build recording.

This means that, in most cases, your SConstruct file needs to be modified, since by default:

- SCons projects do not read build variables to override compiler and linker calls, and
- SCons projects do not read system environment during the build.

The following is an example of Python code that can be added to SConstruct file to:

- allow Goanna to override compiler and linker calls, and
- allow Goanna to read system environment variables during build.

Note that this is only an example; depending on how your SConstruct file is written, modifications may need to be applied in a different way.

```
# Start of SConstruct file
import os
# Also import any extra modules here if required

# Read system environment block
my_env = os.environ

# If your build needs to override PATH or any other environment variables,
# we recommend to do so here
#
# my_env['<variable-name>'] = '...'

# Set up build environment with system environment included.
env = Environment(ENV = my_env)

# If your build needs to specify additional environment settings, then the
# above line can be written as:
#
# env = Environment(ENV = my_env,
#                  ...,
#                  ...)

# Read CC argument from command line, and if it's set, override C compiler.
cc = ARGUMENTS.get('CC', "")
if cc != "":
    env['CC'] = cc

# Read CXX argument from command line, and if it's set, override C++
# compiler.
cxx = ARGUMENTS.get('CXX', "")
if cxx != "":
    env['CXX'] = cxx

# ... - rest of your SConstruct file
```

Goanna requires that your SConstruct file accepts a build variable called CC to override C compiler, and CXX to override C++ compiler. No other variable names are accepted.

**Important Note:** Goanna currently does not support link time analysis if your SConstruct file calls linker directly. This is because Goanna does not inject linker calls for SCons projects.

### 3.3.2 Using goannascons To Capture The Build Settings

Once your SConstruct file is ready, you can run the build and capture the build settings; goannascons utility is used to perform this process.

The steps to run the build and capture the build settings are as follows:

1. Run **scons -c** to "clean" the project.
2. Run:

```
goannascons --record=<build-specification-file> <arguments-to-scons>
```

This command performs and monitors the full build; once the build is complete, goannascons will output the build settings into *<build-specification-file>*. This file is called *build specification file*.

For example, you can run the following to save a build specification to `myproject.goannabuild`:

```
goannascons --record=myproject.goannabuild
```

3. You can now use goanna utility to run analysis on the project; see [4.2](#) for more details.

## 3.4 Setting Up CMake Projects with `goannacmake-conv`

### 3.4.1 Using `goannacmake-conv` To Generate The Build Settings

For CMake projects, Goanna does not provide a utility to directly monitor the build process. Instead, Goanna relies on CMake's ability to generate a compilation database for the build, which then can be used by Goanna to generate a build specification (see <http://clang.llvm.org/docs/JSONCompilationDatabase.html> for more details).

The steps to run the build and capture the build settings are as follows:

1. Run `make clean, ninja -t clean`, or other suitable command to "clean" the project.
2. Perform a CMake build, with a special option `CMAKE_EXPORT_COMPILE_COMMANDS` set to `ON` to generate a compilation database. For example, for Unix Makefile projects, you can run the commands like:

```
cmake -G "Unix Makefiles" -DCMAKE_EXPORT_COMPILE_COMMANDS=ON .  
make
```

or for Ninja projects:

```
cmake -G "Ninja" -DCMAKE_EXPORT_COMPILE_COMMANDS=ON .  
ninja
```

Once the build is finished, you should see a file called `compile_commands.json` in the current working directory. This is the compilation database file.

3. Run:

```
goannacmake-conv --record=<build-specification-file>
```

This command reads the compilation database file, and converts it to Goanna build specification file.

4. You can now use `goanna` utility to run analysis on the project; see [4.2](#) for more details.

**Important:** Goanna does not support CMake projects with generators other than Unix Makefiles and Ninja. Using `goannacmake-conv` on such projects (for example, MinGW Makefiles projects, Visual Studio projects and Eclipse CDT projects) may result in inaccurate analysis results or parse errors.



### 3.5 Setting Up IAR Embedded Workbench® Projects with goannaiarbuild

There are two ways to run Goanna analysis with IAR Embedded Workbench projects:

- If you use IAR's `IarBuild.exe` utility to perform build from command line, you can use `goannaiarbuild` utility to record your build and run Goanna analysis. See the following sections for more details.  
**Note:** Please refer to IAR website (<http://supp.iar.com/Support/?Note=47884>) and IAR Embedded Workbench manual for instructions to configure command line build.
- Alternatively, Goanna Central can be configured to run analysis directly from within IAR Embedded Workbench IDE. See 6.1 for more details.

#### Using goannaiarbuild To Capture The Build Settings

For IAR Embedded Workbench command line projects (with file extension `.ewp`), `goannaiarbuild` can be used to run the build and capture the build settings.

The steps to run the build and capture the build settings are as follows:

1. Run `iarbuild <project-file> -clean <target-name>` to "clean" the project.
2. Run:

```
goannaiarbuild --record=<build-specification-file> <project-file> -make <target-name>
```

This command performs and monitors the full build; once the build is complete, `goannaiarbuild` will output the build settings into `<build-specification-file>`. This file is called *build specification file*.

For example, if the project file is `myproject.ewp` and the target name is `Debug`, you can run the following to save a build specification to `myproject.goannabuild`:

```
goannaiarbuild --record=myproject.goannabuild myproject.ewp -make Debug
```

**Note:** If your build requires a different set of options to `IarBuild.exe`, then you need to pass these options to `goannaiarbuild` as well. The format of options to `goannaiarbuild` is:

```
goannaiarbuild <arguments-to-goannaiarbuild> <arguments-to-IarBuild.exe>
```

3. You can now use `goanna` utility to run analysis on the project; see 4.2 for more details.

### 3.6 Setting Up Keil™ $\mu$ Vision® Projects with gotrace

There are two ways to run Goanna analysis with Keil  $\mu$ Vision projects:

- If you use Keil  $\mu$ Vision (UV4.exe) directly to perform build from command line, you can use gotrace utility to record your build and run Goanna analysis. See the following sections for more details.  
**Note:** Please refer to Keil website ([http://www.keil.com/support/man/docs/uv4/uv4\\_commandline.htm](http://www.keil.com/support/man/docs/uv4/uv4_commandline.htm)) and Keil  $\mu$ Vision manual for instructions to configure command line build.
- Alternatively, Goanna Central can be configured to run analysis directly from within Keil  $\mu$ Vision IDE. See 6.2 for more details.

#### Using gotrace To Capture The Build Settings

For Keil  $\mu$ Vision command line projects (with file extension .uvproj), gotrace can be used to run the build and capture the build settings.

The steps to run the build and capture the build settings are as follows:

1. Run:

```
gotrace <directory-of- $\mu$ vision-installtion>\UV4.exe -j0 -r <project-file>.uvproj
```

This command cleans the build, and then performs and monitors the full build for the last target used. Once the build is complete, gotrace will output the build settings into *<build-specification-file>*. This file is called *build specification file*.

For example, if the project file is myproject.uvproj, and if Keil  $\mu$ Vision is installed to C:\Keil\_v5\UV4, then you can run the following to save a build specification to myproject.goannaspec:

```
gotrace --output-file=myproject.goannaspec "C:\Keil_v5\UV4\UV4.exe" -j0 -r myproject.uvproj
```

**Note:** If your build requires a different set of options to Keil  $\mu$ Vision (UV4.exe), then you need to pass these options to gotrace as well. The format of options to gotrace is:

```
gotrace <arguments-to-gotrace> <arguments-to-Keil- $\mu$ Vision>
```

2. You can now use gokeil utility to run analysis on the project; see 4.3 for more details.

#### Using gotrace With Multi-Project Workspace

If your project uses a multi-project workspace (with file extension .uvmpw), then you will need to specify the target (or use Keil  $\mu$ Vision's -z option for all targets). For example, to perform full rebuild and generate a build specification file for all targets of all projects with a workspace called myworkspace.uvmpw, run:

```
gotrace --output-file=myproject.goannaspec <directory-of-uvision-installtion>\UV4.exe -z -j0 -r myworkspace.uvmpw
```

**Important:** The format of build specification files generated by gotrace is different from those generated by goannamake, goannascons and goannaiarbuild. gotrace format build specification files can only be used with gokeil; goanna cannot read these files.

## 4 Running Goanna Analysis

### 4.1 Introduction

Goanna Central provides two utilities to perform project-wide analysis. Deciding which one to use is based on which build system your project uses:

- If your project is a Keil  $\mu$ Vision project; in other words, if the build specification file was generated by `gotrace`, then use `gokeil`.
- Otherwise (i.e. if the build specification file was generated by `goannamake`, `goannascons`, `goannacmake-conv` or `goannaiarbuild` utility), then use `goanna`.

The following table summarises this difference:

Record Build With...	Utility To Run Analysis
<code>goannamake</code>	<code>goanna</code>
<code>goannascons</code>	<code>goanna</code>
<code>goannacmake-conv</code>	<code>goanna</code>
<code>goannaiarbuild</code>	<code>goanna</code>
<code>gotrace</code>	<code>gokeil</code>

**Note:** It is not possible to use a build specification generated by `goannamake`, `goannascons`, `goannacmake-conv` or `goannaiarbuild` with `gokeil`. Similarly, it is not possible to use a build specification generated by `gotrace` with `goanna`.

Both `goanna` and `gokeil` operate in a similar manner under the hood. `goanna` and `gokeil` generally perform the following steps to run project-wide analysis:

1. `goanna` or `gokeil` reads the build specification file; this file contains essential information about your build, such as the list of source files, compiler options and environment variables used during the build, and the list of produced object files and binaries.
2. `goanna` or `gokeil` calls `goannacc` (for C files) and `goannac++` (for C++ files) to perform analysis on individual source files. During analysis, Goanna uses a database file (called *Project Database* or *Project DB*; see 10.7) to store information about the analysis and any intermediate data required to perform *interprocedural analysis* (see 7.1).  
Once `goannacc` or `goannac++` find any issues in the source file(s), it will report these to the console.
3. Once this is finished, then `goanna ld` is called to perform *link time analysis*. With link time analysis, Goanna can find more issues that may span across the whole project.  
Like `goannacc` and `goannac++`, all issues found by `goanna ld` during link time analysis goes to the console.
4. If enabled, `goanna` or `gokeil` then invokes `CppCheck` and/or `CppNcss` at this point (see 10.11).
5. Finally, `goanna` or `gokeil` publishes the analysis results to the Goanna Dashboard (see Section 8), and optionally to HTML report files (see 4.4.2), a XML file (see 4.4.3), and/or SonarQube (see Section 9).

Goanna provides various configuration options to fine tune Goanna analysis run, such as:

- Selecting what *check(s)* to be enabled.  
A *check* is a Goanna term for a rule that describes a single type of potential issues that may be found within a project. For example, `ARR-inv-index` is a Goanna check that detects attempts to read a value from an array with an invalid (out-of-range) index.  
By default Goanna uses built-in "default" set of checks which contains rules for many common issues. Using the checks selection options allows you to choose exactly which checks, or rules, that Goanna should look for. See 5.1 for more details.
- Whether to include system and/or user header files into the analysis.

For details of configuration options, see 5.

## 4.2 Running Goanna Analysis On Non-Keil Projects With goanna

To run project-wide analysis with goanna, you will need:

- A build specification file generated by goannamake, goannascons, goannacmake-conv or goannaiaar-build.
- Source files and header files used. They must be located in the same location as when the build specification was generated.
- All used compilers (of the same version) and libraries (again the same version) should also be installed to the same location as well.

To run the analysis, follow these steps:

1. Add the path to the compiler should be added into PATH environment variable, and set all other build-related environment variables appropriately (i.e. exactly the same value as during the build recording).

This is needed because Goanna needs to be able to find the compiler, and detect its configuration, during the analysis time to configure Goanna's C/C++ parser to ensure it understands all the source files and header files.

2. Run:

```
goanna --analyze=<build-specification-file>
```

to start the analysis.

It's important to note that the analysis may take a long time; if the project contains a large number of files (generally 100 or more), and/or large files, then the analysis may possibly take hours to complete.

To pass additional Goanna analysis options, add at the end of the command. For example:

```
goanna --analyze=<build-specification-file> <additional-analysis-options>
```

## 4.3 Running Goanna Analysis On Keil™ $\mu$ Vision® Projects With gokeil

To run project-wide analysis with gokeil, you will need:

- A build specification file generated by got race,
- Source files and header files used. They must be located in the same location as when the build specification was generated.
- All used compilers (of the same version) and libraries (again the same version) should also be installed to the same location as well.

To run the analysis, follow these steps:

1. Run:

```
gokeil --buildspec=<build-specification-file>
```

to start the analysis.

**Note:** Unlike goanna, there is no need to (re)set environment variables before analysis when using gokeil.

It's important to note that the analysis may take a long time; if the project contains a large number of files (generally 100 or more), and/or large files, then the analysis may possibly take hours to complete.

To pass additional Goanna analysis options, add at the end of the command. For example:

```
gokeil --buildspec=<build-specification-file> <additional-analysis-options>
```

## 4.4 Reading Analysis Results

The results of Goanna analysis will be made available in several places:

- During analysis, Goanna will show the name of the file(s) being analyzed, and any issues found within that file to the console; see 4.4.1 for details.
- Once the analysis is complete, Goanna automatically publishes the result of the analysis to the Goanna Dashboard.  
Goanna Dashboard is a web-based interactive interface that allows you to see the history of all analysis runs on particular project(s). See 8 for details.
- Goanna can optionally output the analysis result to HTML report files (see 4.4.2), or a XML file (see 4.4.3).
- Goanna can also be configured to publish the analysis result to SonarQube (see 9).

### 4.4.1 Goanna Output On The Console

The simplest way to see the result of the analysis is the output message on the console (stderr by default). The following is an example of the output message from Goanna:

```
Goanna - analyzing file foo.c
Number of functions: 20
foo.c:200 warning: Goanna[MEM-free-no-alloc] Severity-Medium, Pointer 'tmp'
    is freed without being allocated memory.
foo.c:211 warning: Goanna[ATH-cmp-float] Severity-Low, Comparison with a
    float using == or !=.
    CERT-FLP06-C, CERT-FLP35-CPP, MISRAC2004-13.3, MISRAC++2008-6-2-2
foo.c:222 warning: Goanna[ATH-cmp-float] Severity-Low, Comparison with a
    float using == or !=.
    CERT-FLP06-C, CERT-FLP35-CPP, MISRAC2004-13.3, MISRAC++2008-6-2-2
```

Any issues in the project found by Goanna are called *warnings* (or *Goanna warnings*). In this example, Goanna shows three warnings, one from MEM-free-no-alloc check and two from ATH-cmp-float check.

By default, Goanna shows the following information for each warning:

- File name, and line number of the warning.
- Name of the check which generated this warning.
- Severity of the warning, which is one of: Low, Medium and High. Severity is determined on per-check basis and is not affected by the actual code analyzed.
- Warning message, explaining why this warning is shown.
- A list of Goanna recognized C/C++ standard rules that may be violated with this warning.

For example, in the above example, the following output message of the first warning from ATH-cmp-float:

```
foo.c:211 warning: Goanna[ATH-cmp-float] Severity-Low, Comparison with a
    float using == or !=.
    CERT-FLP06-C, CERT-FLP35-CPP, MISRAC2004-13.3, MISRAC++2008-6-2-2
```

Indicates that:

- On foo.c line 211, there is a comparison on a variable or a constant value of a type float or double using equality operators,
- This warning is determined by Goanna to have Low severity,
- This warning comes from a Goanna check called ATH-cmp-float, and
- The presence of this warning may mean that the project does not satisfy requirements imposed by one or more of the following rules:

- CERT C Secure Coding Standard, Rule FLP06-C
- CERT C++ Secure Coding Standard, Rule FLP35-CPP
- MISRA-C:2004 Guideline, Rule 13.3
- MISRA C++:2012 Guideline, Rule 6-2-2

It is possible to customize the format of the output message with **--output-format** option. See the [goannacc](#) reference (11.7) for details about this option.

### Warning Traces

Goanna also has an ability to show a trace through the execution path leading to a warning; this is called a *warning trace*. This is useful to identify important events during the execution and understand why the warning was issued.

The following is an example of warning output with a trace:

```
foo.c:254: warning: Goanna[PTR-null-fun-pos] Severity-High, Function call
'get_obj(o)' is immediately dereferenced, without checking for NULL.
CERT-EXP34-C,CWE-476
252: ^ - if (flags & SOME_CONDITION) is false
254: ! - possible_null
254: > - Entering into get_obj
315: ^ - if (!o) is false
316: ^ - switch (o->o_ty)
320: ! - Return NULL
```

To display traces for warnings, pass **--trace** option to `goanna` or `gokeil` when running analysis.

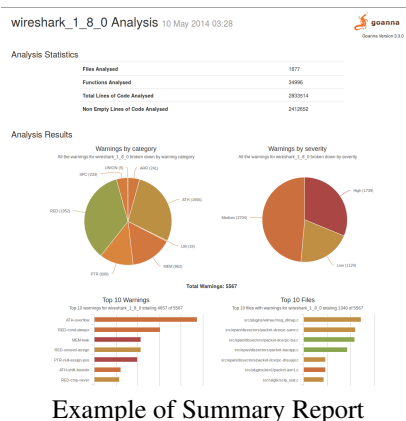
### 4.4.2 HTML Report of Analysis Results

Goanna can optionally generate an HTML report of the analysis results. There are two types of HTML reports:

- *Summary Report*, which shows a summary of the analysis result, including:
  - Basic statistics, such as the number of files analyzed,
  - Per-category and per-severity warnings pie chart,
  - Bar charts of the top 10 warnings per check and top 10 files with warnings, and
  - List of warning numbers per check.

This report shows the high-level statistics of the analysis results in an easy-to-see format; and

- *Warnings Report*, which shows a table of all warnings found; this table is interactive and supports filtering.



Example of Summary Report

File	Warning	Severity	Rules	Message
wiresarkmain.c	PTR-null-var	Medium	CWE-121,MISRA-C2004-17,MISRA-C2004-10-11,UBSAN-2009-0-14,UBSAN-4.3	Pointer arithmetic is performed on the address of variable 'var' that is neither a pointer nor an array.
wiresarkmain.c	RED-obj-always	Low	CWE-617,MISRA-C2004-13.7	Comparison 'var_tag(32) <= 0' always holds.
wiresarkmain.c	RED-obj-always	Low	CWE-617,MISRA-C2004-13.7,UBSAN-2.0.2	Comparison 'var_tag(32) <= 0' always holds.
wiresarkmain.c	RED-obj-always	Low	CWE-617,MISRA-C2004-13.7,UBSAN-2.0.2	Comparison 'var_tag(32) <= 0' always holds.
wiresarkmain.c	RED-obj-always	Low	CWE-617,CWE-617	Constant used in condition.
wiresarkmain.c	PTR-null-var	Medium	CWE-121,MISRA-C2004-17,MISRA-C2004-10-11,UBSAN-4.3	Pointer arithmetic is performed on the address of variable 'var' that is neither a pointer nor an array.
wiresarkmain.c	C-enum-addr	Medium	CDE1-FROM-C,UBSAN-2009-0-14,UBSAN-4.3	Pointer arithmetic is performed on the address of variable 'var' that is neither a pointer nor an array.
wiresarkmain.c	RED-enum-addr	Low	CWE-617,CWE-617	Condition 'var_tag(32) <= 0' is always true.
wiresarkmain.c	RED-enum-addr	Medium	CDE1-FROM-C,MISRA-C2004-10-11,MISRA-C2004-10-11	Constant expression '1' is always true.

Example of Warnings Report

To generate HTML report files at the end of analysis, pass **--html-report** option to `goanna` or `gokeil` when running analysis. This option accepts an optional argument to specify the type(s) of report to be generated:

- **--html-report=summary**: Generates summary report file.
- **--html-report=warnings**: Generates warnings report file.
- **--html-report=all** or **--html-report** with no extra argument: Generates both summary and warnings report files.
- **No --html-report option**: HTML report files will not be generated.

#### 4.4.3 Analysis Results In XML File

Goanna can optionally output the analysis result to a XML file. This is useful if you need Goanna to be used in conjunction with some other platform or framework and need programmatic access to the analysis result.

To generate a XML output file at the end of analysis, pass **--output-xml=<xml-file-name>** option to goanna or gokeil when running analysis.

#### 4.4.4 Using Goanna Dashboard Web Interface To Interact With Analysis Results

Goanna comes with the Goanna Dashboard, a web-based interface that allows you to read and interact with the analysis result. By default, Goanna automatically publishes the analysis results to the Goanna Dashboard at the end of analysis.

For more details about this feature, see [Section 8](#).

#### 4.4.5 Using SonarQube To Interact With Analysis Results

Goanna also has an ability to publish the analysis results to SonarQube.

For more details about this feature, see [Section 9](#).

## 5 Configuring Goanna Analysis

### 5.1 Setting Checks

#### 5.1.1 Introduction

Goanna provides a set of options to configure what *checks*, or rules, to be used in the analysis to control types of issues that Goanna should look for.

By default, Goanna runs analysis with the default set of checks which is designed to find many common issues for most domains of application in the short time.

Goanna checks are identified by names such as `ATH-div-0` and `MEM-leak`. Refer to the *Goanna Reference Guide* for the list of available and default checks, and their full description and code examples.

#### 5.1.2 Selecting Checks With Command Line Options

To specify a set of checks to be used during analysis, pass `--check=<check-name>` to `goanna` or `gokeil` when running analysis. For example:

```
goanna --analyze=myproject.goannabuild --check=ATH-div-0 --check=MEM-leak
```

You can pass this option multiple times to specify a set of multiple checks.

#### Selecting All Checks Within The Coding Standard

Goanna also provides `--checks=<name-of-standard>` option (note the extra `s`) to specify whole group(s) of checks from popular coding standards. For example, running the following command:

```
goanna --analyze=myproject.goannabuild --checks=misrac2004
```

causes Goanna to run analysis with all available checks for MISRA C:2004 coding standard.

It is also possible to specify a particular rule within the coding standard. For example, to run only the check(s) corresponding to MISRA C:2004 Rule 12.8, run:

```
goanna --analyze=myproject.goannabuild --checks=misrac2004-12.8
```

The available coding standards in this version of Goanna Central are as follows:

Standard code	Standard name
cert	Computer Emergency Response Team (CERT) C/C++ Coding Standard
cwe	Common Weakness Enumeration (CWE)
misrac2004	Motor Industry Software Reliability Association (MISRA) C:2004
misrac++2008	Motor Industry Software Reliability Association (MISRA) C++:2008
misrac2012	Motor Industry Software Reliability Association (MISRA) C:2012

In addition to the above list, there is a special “standard” called **default-on** which, if specified, will enable the Goanna default set of checks.

Like `--check` option, you can pass `--checks` option multiple times to specify a set of multiple coding standards or coding standard rules. It is also possible to pass a mix of `--check` and `--checks` options to select all of chosen checks, coding standards and coding standard rules.



### 5.1.3 Using Checks File To Select Multiple Checks

In addition to the above options, Goanna also provides a mechanism called *checks file* to specify a set of manually chosen checks.

A checks file is a text file with a list of Goanna check names (separated by a breakline), and looks like the following:

```
# This is a comment.  
MEM-double-free  
MEM-double-free-alias  
MEM-leak  
MEM-leak-alias
```

# symbol indicates a one-line comment; # symbol and anything after that (until the end of the line) will be ignored by Goanna.

You can use any text editor to create a checks file.

Once a checks file is created, pass **--checks-file=<path-to-file>** to use the checks file; for example:

```
goanna --analyze=myproject.goannabuild --checks-file=mystandard.checks
```

**Note:** --checks-file option is mutually exclusive to --check and --checks options. It is not possible to use --checks-file option in conjunction to --check or --checks options.

### 5.1.4 Enabling All Available Checks

Generally, you should only enable Goanna checks that are relevant to your environment.

Goanna analysis engine is designed to perform some computationally expensive checking algorithm only when required by the enabled checks; if you have unnecessary checks turned on, this may cause the analysis to take substantially longer.

In addition, some Goanna checks are either syntactic or specialized to particular rules within coding standards. If you have these checks turned on, this may result in a large number of spurious warnings.

Nevertheless, Goanna provides an option to enable all available Goanna checks if this is required.

To enable all available Goanna checks, pass **--all-checks** to goanna or gokeil when running analysis. For example:

```
goanna --analyze=myproject.goannabuild --all-checks
```

**Note:** Using --all-checks option will cause any --check, --checks and --checks-file options to be ignored.

## 5.2 Checks Packages

Goanna Central provides over 180 individual checks, out of the box, that can be used to analyse your C/C++ source code. It is also possible to enable an additional 450+ checks that are dedicated to providing coverage for popular C/C++ standards. Controlling which checks are available can be done through the checks package commands of `goannacc`. When additional packages are installed and enabled, the checks they provide can be selected for analysis by using the check selection options described in Section 5.1.

Goanna Central ships with the following checks packages:

- **stdchecks** (enabled by default): "Goanna Core" checks. This checks package contains a set of checks for common C/C++ issues.
- **misrac2004**: Dedicated checks for MISRA-C:2004 coding standard.
- **misrac++2008**: Dedicated checks for MISRA C++:2008 coding standard.
- **misrac2012**: Dedicated checks for MISRA C:2012 coding standard.

### Important Note

- All checks package commands are available only with `goannacc`; `goanna` or `gokeil` do not support these commands.
- Checks package operations are global and may affect analysis on all existing *and new* projects. Additionally, enabling a checks package will also change the default set of checks to be enabled for all existing *and new* projects.
- Goanna does not provide an option to use checks package for a single analysis run only; if you wish to do this, you must first enable a checks package, run analysis, and then disable the checks package.
- Checks package commands require write access to the installation directory of Goanna. Generally, this means:
  - On Windows, you need to issue checks package commands from within Command Prompt opened as Administrator.
  - On Linux, if you have installed Goanna Central with `sudo` or from within root terminal (i.e. under root privilege), then you need to issue checks package commands with `sudo`, or from within root terminal.

### 5.2.1 Listing Checks Packages

To see which checks packages are installed or available, run:

```
goannacc --package-list
```

This will give you the list of installed and enabled checks packages, for example:

```
Package Available: misrac2004
Package Available: misrac++2008
Package Available: misrac2012
Package Installed: stdchecks
```

The checks packages that are installed but not enabled yet are shown as “Available”, and the checks packages that are installed and enabled are shown as “Installed”.

### 5.2.2 Enabling Available Checks Package

To enable an already installed checks package, run:

```
goannacc --package-enable=<package-name>
```

**Note:** Use this command only for checks packages that are already installed. If you wish to enable a checks package that is not installed yet, use **--package-install** option; see 5.2.4 for details.

### 5.2.3 Disabling Installed Checks Package

To disable an enabled checks package, run:

```
goannacc --package-disable=<package-name>
```

### 5.2.4 Installing Custom Checks Package

In addition to the pre-installed checks packages, it is possible to install a custom checks package that is provided in a `.goannapackage` file.

To install and enable a custom checks package, run:

```
goannacc --package-install=<package-file>
```

Contact Red Lizard Software for more information about custom checks packages.

## 5.3 Including Headers Into Analysis

By default, Goanna analyzes source files only; any header files included are read by Goanna, but are not analyzed. To include user headers (generally, those included using `#include "..."` syntax) into analysis, pass **--user-headers** option to `goanna` or `gokeil` when running analysis. For example:

```
goanna --analyze=myproject.goannabuild --user-headers
```

To include system headers (generally, those included using `#include <...>` syntax, such as C and C++ standard header files) into analysis, pass **--system-headers** option to `goanna` or `gokeil` when running analysis. For example:

```
goanna --analyze=myproject.goannabuild --system-headers
```

Please note that using these options may increase time needed to run analysis.

## 5.4 Excluding Certain Files From Analysis

By default, Goanna analyzes all source files within the build. If you wish to only analyze some source files, it is possible to use **--exclude=<pattern>** option to exclude some files from analysis. For example:

```
goanna --analyze=myproject.goannabuild --exclude=tests/*.c
```

`--exclude` option takes a regular expression pattern of file paths, relative from the project root directory.

You can specify `--exclude` option multiple times to specify a set of multiple file path patterns.

## 5.5 Setting Analysis Timeouts

By default, Goanna has a timeout of 240 seconds to spend in each analysis phase within one source file. To change the timeout, pass **--timeout=<timeout-in-seconds>** to `goanna` or `gokeil` when running analysis. For example:

```
goanna --analyze=myproject.goannabuild --timeout=60
```

Generally speaking, increasing timeout may result in more accurate results, but will take longer to complete the analysis. Decreasing timeout will improve the running time, but may result in less accurate results. Due to the underlying technology of the Goanna analysis engine, this timeout is essential.

## 5.6 Ignoring Certain Warnings ("Warning Suppression")

If you run Goanna frequently on the same project, in some cases, you may wish to ignore some warnings that do not require immediate attention.

Goanna provides a mechanism to allow you to manually specify such warnings to be ignored; these warnings will then not be reported in subsequent analysis. This is called *warning suppression*.

The recommended way to specify warnings to be suppressed is the Goanna Dashboard. See [8.2](#) for more details.

## 5.7 Other Configuration Options

In addition to those listed above, Goanna provides a range of analysis options to fine tune the analysis. For the list of all analysis options, see `goanna` reference (see [11.1](#)), `gokeil` reference (see [11.6](#)) and `goannacc` reference (see [11.7](#)).

Some analysis options may refer to advanced concepts not explained here; see [10](#) for details.

## 6 Running Goanna Analysis From Within IDEs

Goanna Central supports a limited form of IDE integration with the following IDEs:

- IAR Embedded Workbench version 5.40 and higher
- Keil  $\mu$ Vision version 4 and higher

If you use one of the above IDEs, you can configure the IDE to allow Goanna to run analysis from within the IDE. This provides a convenient method to perform analysis while working on the project, without falling back to the command line.

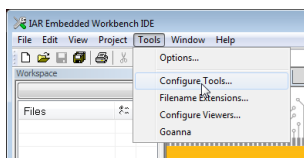
**Note:** Goanna Central does not provide full IDE integration, in the sense that it does not provide a full GUI to configure analysis. Such configurations generally need to be done through command line options, similar to command line analysis.

### 6.1 Running Goanna Analysis From IAR Embedded Workbench®

With `goannaiarbuild` utility, it is possible to set up IAR Embedded Workbench IDE to allow Goanna analysis to be run directly from within IAR Embedded Workbench.

To set up IAR Embedded Workbench IDE, follow these steps:

1. Start IAR Embedded Workbench.



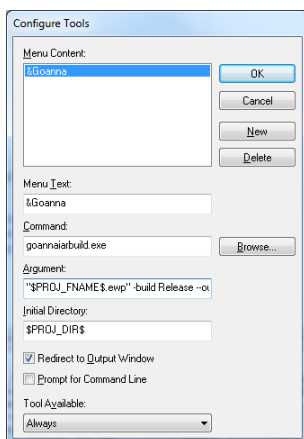
2. From the Tools menu, select Configure Tools....

3. In the Configure Tools dialog that follows, click the New button.
4. For the Menu Text of the new menu item, type &Goanna.
5. Next to Command:, type `goannaiarbuild.exe`.
6. Next to Arguments:, copy and paste the following text:

```
--output-format="%SOH%%RELFILE%:%LINENO%%SOH%: warning: Goanna[%CHECKNAME%]  
Severity-%SEVERITY%, %MESSAGE%. %RULES%%EOL%" "$PROJ_FNAME$.ewp" -build <target-  
name>
```

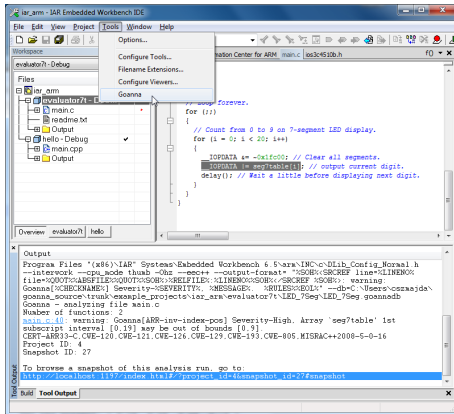
where `<target-name>` is a name of target configuration you wish to build (and therefore analyze). To see the list of available configurations, navigate to Project »Edit Configurations....

7. Check Redirect to Output Window.



8. Click OK.

You should now see a new menu item Goanna on your Tools menu. Clicking this menu entry runs full build, and performs analysis on the active project, with the results appearing in the Output window.



## Notes On Using Goanna With IAR Embedded Workbench

The above method of running Goanna analysis from within the IDE does *not* generate a build specification. Instead, goannaiarbuild operates in a special mode where it performs full build and the analysis at the same time (see 10.4 for details of this mode).

You will need to perform the set up process again:

- When you wish to add, modify or remove any Goanna analysis options; these need to be added at the start of goannaiarbuild.exe arguments (in Arguments: text box), or
- When you wish to change the target configuration used for the analysis.

## 6.2 Running Goanna Analysis From Keil™ μVision®

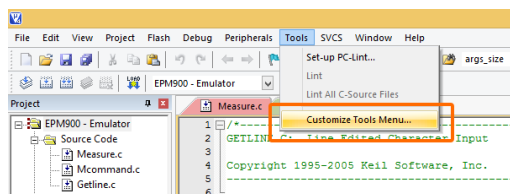
**Important:** You need to install Keil μVision Support Package to use Goanna with Keil μVision. Contact your distributor to obtain the installer.



With gotrace and gokeil utilities, it is possible to set up Keil μVision IDE to allow Goanna analysis to be run directly on your project from with Keil μVision IDE.

**Note:** This procedure works only for a single project. If you use this method on a multi-project workspace, then the analysis will be performed only on the active project.

To set up Keil μVision IDE, follow these steps:

1. Start Keil μVision.
2. From the Tools menu, select Customize Tools Menu...

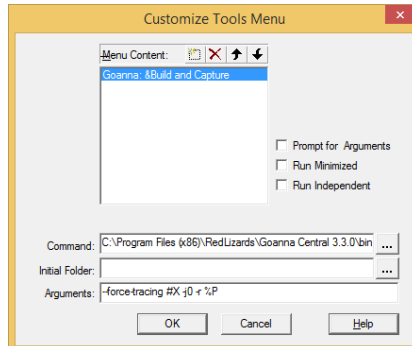




3. In the Customize Tools Menu dialog that follows, click the New (Insert)  button. This will create a new menu entry to run gotrace to capture the build.
4. Type Goanna: &Build and Capture and press Enter. This will become the name of the new menu entry. You can assign different name here if you wish.
5. Click "Browse"  button next to Command entry. A file selection dialog will open. Navigate to the directory where Goanna Central is installed, and choose gotrace.exe.

6. Next to Arguments:, copy and paste the following text:

**--force-tracing #X -j0 -r %P**

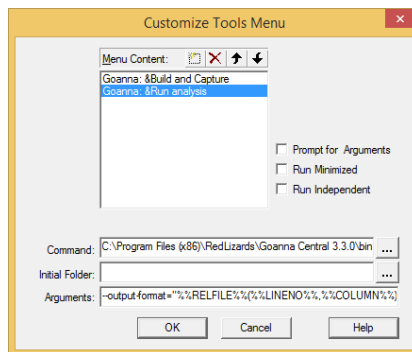
**Note:** If your project requires additional Keil  $\mu$ Vision options to build, then you will need to append these here. See [Description](#) section above.



7. Click the New (Insert)  button again to create a new menu entry to run `gokeil` to run the analysis.
8. Type `Goanna: &Run analysis` and press Enter. Again you can assign different name here if you wish.
9. Click "Browse"  button next to Command entry. Navigate to the directory where Goanna Central is installed, and choose `gokeil.exe`.
10. Next to Arguments:, copy and paste the following text:

**--output-format="%%RELFILE%%(%%LINENO%%,%%COLUMN%%): %%TYPE%%: Goanna[%%CHECKNAME%%] Severity-%%SEVERITY%%, %%MESSAGE%%. %%RULES%%"**

**Note:** You will need to append Goanna options here to change analysis configurations. See [Running analysis from Keil  \$\mu\$ Vision](#) section below.



11. Click OK.

You should now see a new menu item `Goanna: Build and Capture` and `Goanna: Run analysis` on your Tools menu (unless if named differently).

## Running analysis from Keil $\mu$ Vision

After setting up your Keil  $\mu$ Vision, you can follow these steps to run the analysis:

1. Before running analysis, make sure to save any changes to the project or any source files within.
2. You can proceed to the next step (skipping the build specification generation) if your project meets all of the following conditions:
  - You have run the analysis on the project before (i.e. there is an existing build specification file)
  - Since the last analysis, there are no newly added or removed files
  - Since the last analysis, there are no changes to the project settings

Otherwise, you will need to build the project and generate the build specification first. To do so, follow these steps:

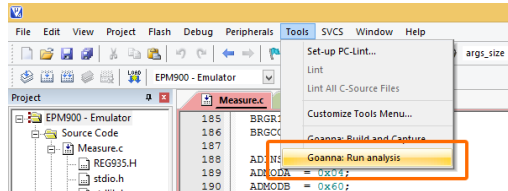
- (a) Make sure that your desired target is selected. To select a different target, you can use the drop-down selection in the Keil  $\mu$ Vision's toolbar.
- (b) Go to Tools menu and click Goanna: Build and Capture. This will rebuild your project and generate the build specification.
- (c) A console window appears, and the build starts. Wait until this windows is automatically closed; when the window is closed, the build is complete. Once the build is complete, you will also see a message like:

```
gotrace: Build information recorded to the build spec file 'goanna.goannaspec'
(54 compiler calls, 2 linker calls).
```

**Note:** gotrace will not show any progress information during build. This is due to technical limitation of Keil  $\mu$ Vision.

3. To run the analysis with default settings (or custom settings saved from the last analysis - see below), go to Tools menu and click Goanna: Run analysis.
4. Alternatively, you can change the analysis option before running the analysis. To do so, follow these steps:
  - (a) Go to Tools menu and click Customize Tools Menu....
  - (b) Select Goanna: Run analysis from the menu list.
  - (c) Add any extra Goanna options you would like to add at the end of Arguments: text box. You can also remove options that are no longer needed. Changes made here will be saved for any subsequent analysis.

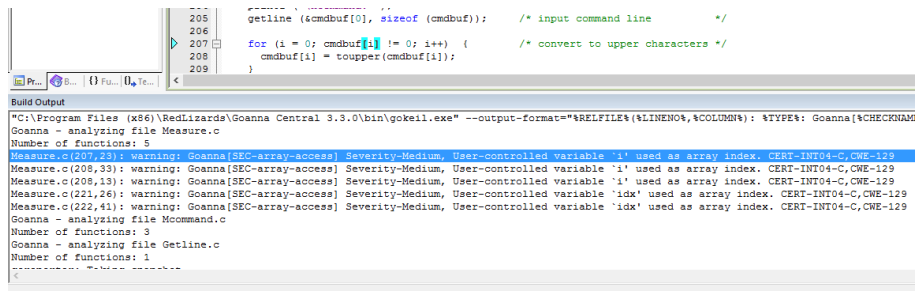
**Tips:** If you frequently change the analysis options, you can enable Prompt for Arguments option to make Keil  $\mu$ Vision ask for Goanna analysis options every time.
  - (d) Press OK.
  - (e) Now open Tools menu again and click Goanna: Run analysis to run the analysis.



5. Once the analysis starts, a console window appears. Wait until this window is automatically closed; when the window is closed, the analysis is complete. Analysis results should be shown in the "Build Output" window.

**Note:** gokeil will not show any progress information during analysis. This is due to technical limitation of Keil  $\mu$ Vision.

6. You can jump to the line with a Goanna warning by double clicking at the Goanna warning line in the "Build Output" window.





## 7 Getting the Best Results from Goanna

### 7.1 Interprocedural Analysis

Goanna's interprocedural analysis propagates information about function behaviour to other functions. This information includes parameter values, return values, and function effects that may impact other parts of the code. This enables Goanna to detect things in your program such as freeing of memory through function calls, functions that never return, and input values to some functions.

Interprocedural analysis is not limited to a specific set of checks, but rather enhances the precision of many checks. An example of what interprocedural analysis can find can be seen in the sample of function `myAlloc`.

```
void *myAlloc(int param){
    void *p = malloc(param);
    if (p)
        return p;
    else
        return NULL;
}

int main(int argc, char ** argv) {
    int * n;
    n = (int*)myAlloc(sizeof(int) * 10);
    n[0] = 5; // this may be a dereference of NULL
    return *n;
}
```

Here, Goanna learns that `myAlloc` may return `NULL`. This means that when the return value of `myAlloc` is assigned to `n`, Goanna knows this value may be `NULL`. Therefore, the expression `n` may be dereferencing a `NULL` pointer, and Goanna will warn accordingly.

There is some additional computation overhead in running interprocedural analysis. If you need rapid results without much depth, then turning off interprocedural analysis will provide faster results (at the cost of accuracy in some checks). To turn off interprocedural analysis, use the `--no-ipa` option.

By default interprocedural analysis does two passes (in optimized order) over each file. This provides a good approximation for function behaviours, but may miss some complex behaviours that require many passes to accurately detect. Additional precision can be gained by increasing the iteration limit (the maximum number of passes Goanna will do). To change the interprocedural analysis iteration limit, use the `--ipa-iterations` option.

### 7.2 A Word on False Positives

Goanna considers all possible execution paths in your program, and will warn you if it finds potential defects (such as use of an uninitialised variable) that occur only on particular execution paths and not others. But sometimes, the execution path leading to a potential defect is actually not possible when the program is executed. If Goanna is able to deduce this through static analysis, then it won't warn you. But if it can't, then you may receive a spurious warning for a defect that isn't really there. Such warnings are called *false positives*.

Some false positives occur because Goanna currently does not track dependencies between variables in loops. For example, if you have a loop with two counters and only test one:

```
char buffer[11];
int i, count;

i = 0;
count = 10;
while (--count >= 0) {
    buffer[i++] = 'x';
}
buffer[i] = '\0';
```

Goanna may issue a false positive warning because it doesn't deduce that `i = 10` when the loop terminates.

Such false positives can often be suppressed with the `assert` macro (sec. 7.4). Otherwise you can suppress false positives using the Goanna Dashboard (see 8.2).

### 7.3 Using the `_GOANNA` Preprocessor Symbol

Goanna has a built-in preprocessor definition, defined by the macro

```
#define _GOANNA 1
```

This allows code to be explicitly included in or excluded from analysis by Goanna. For example:

```
#ifndef _GOANNA
//Code only to be included while the program is being analysed
#endif
#ifdef _GOANNA
//Code not to be analyzed by Goanna
#endif
```

### 7.4 Using the `assert` macro

Goanna can sometimes use information provided by `assert()` to refine its analysis of numerical and pointer values. It does this by using `assert` statements as assumptions for value ranges and pointer validity.

For example, in the code below:

```
void my_fun(void) {
    int my_array[20];
    int x = rand();

    assert(x == 10);
    f(my_array[x]);
}
```

the `assert()` means that the array reference must be in-bounds, even though the index variable `x` has a randomly-assigned value. Therefore, Goanna does not issue an out-of-bounds warning.

### 7.5 Sample Code

A package containing a number of sample C and C++ files is available on our website. Go to <http://redlizards.com/resources/example-code/> and download the Goanna Central Sample Code package. The files contained in this package may be useful for practicing using Goanna, or ensuring that it is working correctly.

A build specification is included in the package, which analyzes all the files included. To run analysis on this sample code package, run:

```
goanna --analyze=central.goannabuild
```

You will need to install `gcc` (GNU C Compiler) and make sure that `PATH` environment variable contains a path to where `gcc` is installed before running analysis on this sample code package.

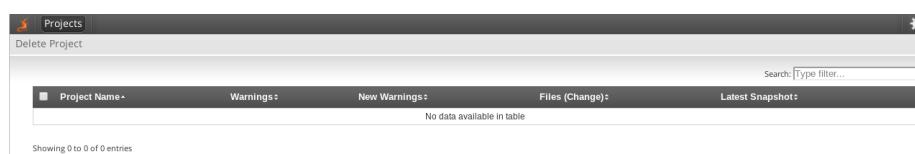
## 8 Using the Goanna Dashboard

Goanna Dashboard allows you to store and visualise the history of your Goanna results. It includes a web server, `goreporter`, used to display these results in a web browser. Each of your analysis runs is captured in a snapshot, which comprises of the warnings for that run as well as the source code analysed in the run. These snapshots are used to track the history of your project in the Goanna Dashboard.

### 8.1 Getting to the Goanna Dashboard

On Windows, and also on Linux when Goanna Central was installed under `sudo` or from within the root terminal, the Goanna Dashboard is configured to start automatically at the computer startup. In this case, you can simply navigate to <http://localhost:1197/> to access the Goanna Dashboard. This will show the Project Page with all projects Goanna has taken a snapshot of.

Otherwise, you need to start the Goanna Dashboard server first. See [11.9](#) for instructions on how to do this.



If you load the Goanna Dashboard without first taking a snapshot you will see an empty table with the text “No data available in table”. To use the Goanna Dashboard you must first take a snapshot. (refer to [11.9](#))

### 8.2 Bug Statuses

The Goanna Dashboard allows you to classify bugs into one of five statuses:

**Unclassified** This is the default status for when a new warning is added to the Goanna Dashboard.

**Ignore** Ignore this warning; useful when the warning is valid, but does not require immediate attention.

**Analyse** This warning need to be investigated further before it can be classified properly.

**Fix** This warning is a problem that needs to be fixed.

**Not a Problem** This is warning is not a real bug (false positive).

If you select a warning to be either **Ignore** or **Not a Problem**, then these warnings are automatically *suppressed*; Goanna will then ignore these warnings in future analysis runs.

### 8.3 Severity

Every check in Goanna has been assigned a static severity of either; High, Medium or Low. These severities are represented throughout the Goanna Dashboard in three colours (or a blend of these three colours):

- **High:** Red
- **Medium:** Yellow
- **Low:** Green

### 8.4 Dashboard Views

#### 8.4.1 Project Page

The project page is usually where you begin when using the Goanna Dashboard. It gives an overview of all projects Goanna has analysed and taken a snapshot of. The table provides a high level overview of the current state of all projects, including the number of warnings, number of new warnings, number of files analysed and the last time a snapshot was taken.

If you do not want to have a project in your dashboard anymore, you can use the checkboxes on the left hand side of the table to select the project(s) you wish to delete, then use the Delete Project button in the Dashboard’s toolbar.

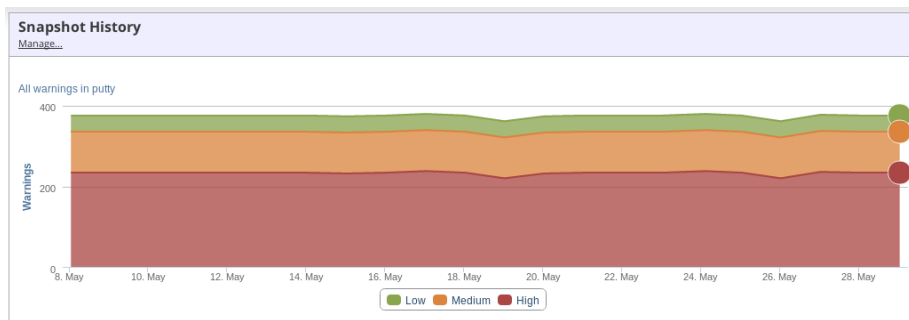
Project Name	Warnings	New Warnings	Files (Change)	Latest Snapshot
audacity	1954	0	742 (0)	2013/05/29 05:50:51
brssj	147	0	161 (0)	2013/05/29 00:18:32
jsone	2	0	7 (0)	2013/05/29 00:03:38
ivip_misrac2004	5723	0	74 (0)	2013/05/29 00:27:14
mongodb	1990	6	593 (1)	2013/05/28 16:03:53
putty	376	0	102 (0)	2013/05/29 00:47:26
qpc_dpp	197	0	14 (0)	2013/05/29 00:28:42

Showing 1 to 7 of 7 entries

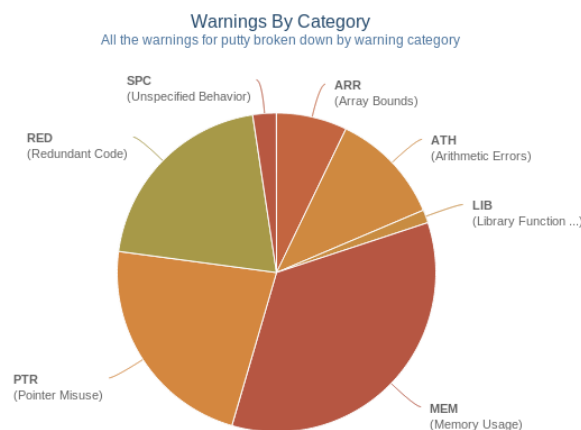
## 8.4.2 Report Page

Once you select a project you are taken to the project's report page. This page shows four graphs:

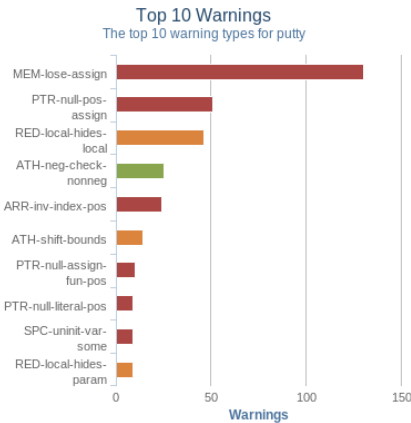
**Snapshot History** shows the overall progress of your project over time. Each point on the graph is one of your previous snapshots, showing the total number of warnings by severity. Clicking on any point in the graph will change the report page to show details about warnings in that snapshot. By default, the most recent snapshot is selected.



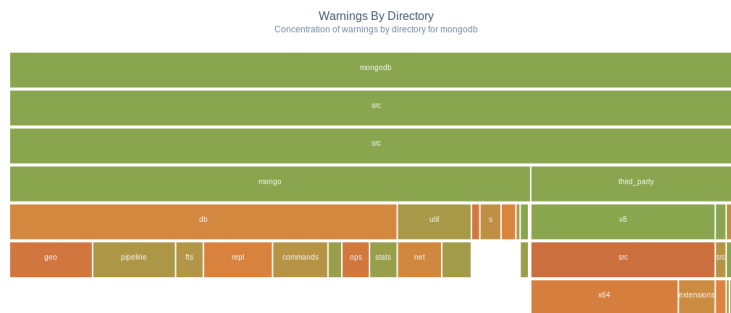
**Warnings By Category** shows all warnings for your snapshot broken down into each warning type. Clicking on a wedge will show the break down of warnings of that category. Clicking on a subsequent wedge will take you to the warnings browser filtered for that particular warning type.



**Top 10 Warnings** shows the top ten warning types in this snapshot. Clicking on one of the bars in the chart will take you to the warnings browser filtered by that warning type.

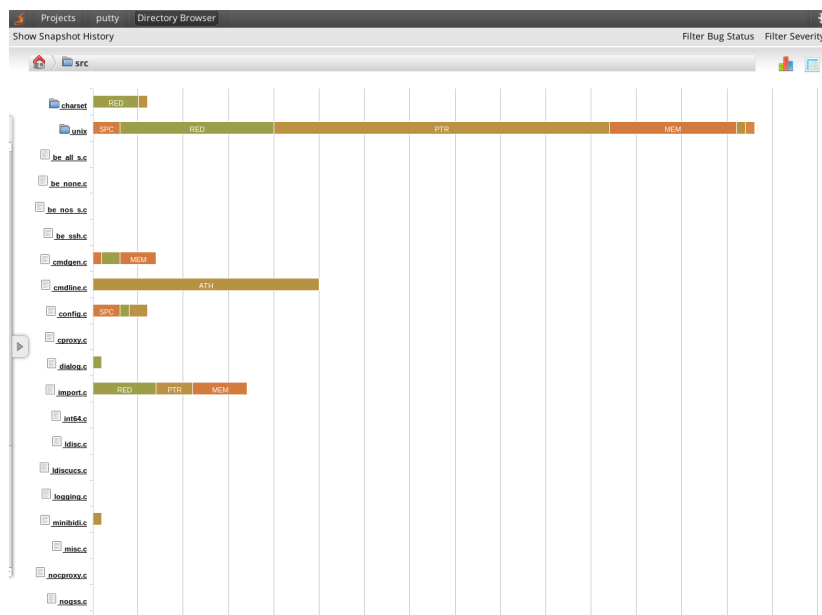


**Warnings By Directory** shows the concentration of warnings in your directory structure. A red node means that there are more than 10 warnings per file (average) in the directory, a green one means there are zero warnings per file (average). Clicking on a node will load the directory browser in that folder.



Global filters can be applied from the toolbar to filter by warning severity and/or bug status. These filters apply to all charts.

### 8.4.3 Directory Browser

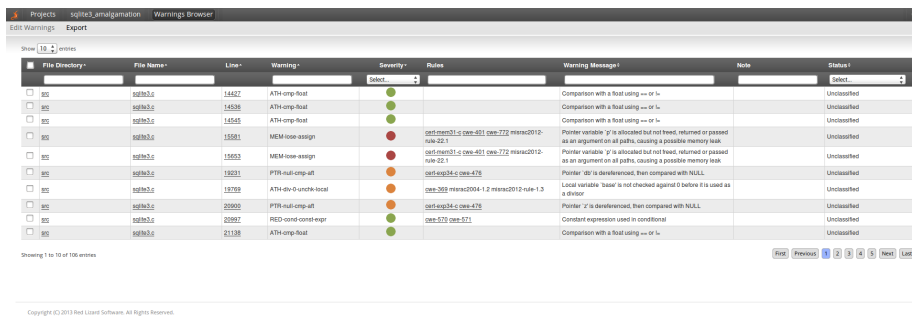


The directory browser is a way to browse through your project's directory structure to see what files and folders have what warnings. Total number of warnings per file or folder are broken down into warning categories and displayed in a bar chart.

The directory browser allows you to browse through your source tree in a few ways. The location bar above the chart allows you to see the path to your current location. Clicking on an item in the location bar will take you there in the directory browser. Similarly, clicking on a directory name in the chart will reload the directory browser with the contents of that folder. To view the contents of a source file, click on its name to load the code browser. To see the details about a particular warning category for a file, click on the segment for that category in the bar of the file or folder to load the warnings browser filtered for your selection.

The chart can be filtered in two ways. The sidebar allows for particular warning types, or warning categories to be turned on and off. In addition, the global filters in the toolbar (severity and bug status) also apply to this chart.

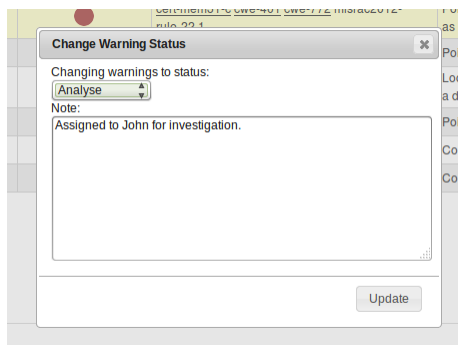
### 8.4.4 Warnings Browser



The warnings browser shows details of all the warnings in your project. Filtering is possible through the filter boxes in the header of the warnings table. The arrows in the table header allow for sorting.

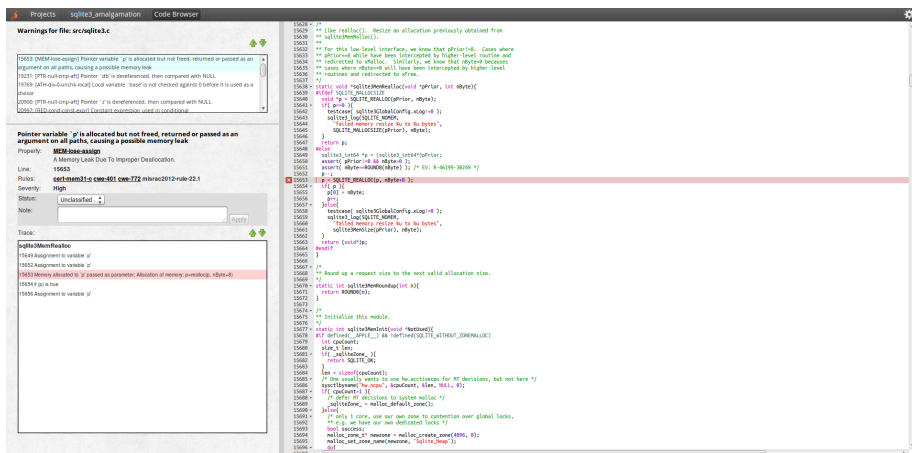
Clicking on a directory name will take you to the directory browser for that directory. A file name or line number will take you to the source code browser for that file and warning. Clicking on a Rule or Warning name will give you a description of that rule.

Selecting warnings then clicking "Edit Warnings" button opens a dialog where you can change their status and also add a note to the warnings. Clicking Update saves these changes.

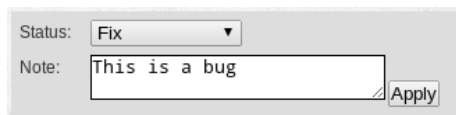


The 'Export' button allows you to export all warnings or visible warnings to a CSV file.

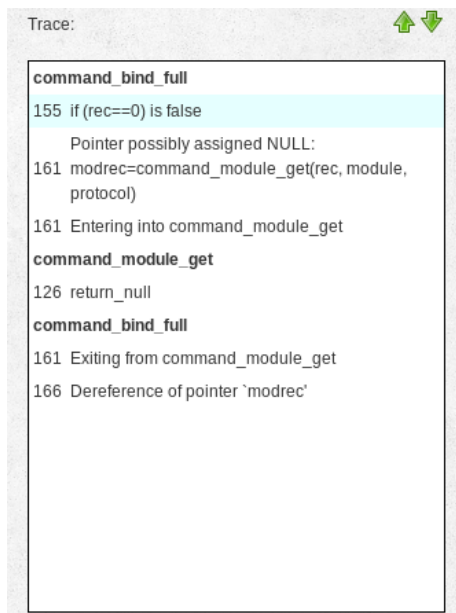
## 8.4.5 Code Browser



The code browser displays a file and all its Goanna warnings. The right hand side shows the source code of the file currently opened with warnings highlighted in three colours based on severity. The left hand side has the details of warnings for this file. The box at the top of the left hand pane allows you to select a warning to see in more detail. Like in the warnings browser, you are able to change the status and add a note to each warning in the code browser. This can be done by changing the status and/or adding a note and pressing Apply.



If there is trace information for a warning it will also appear in this pane. You can step through the trace just like a debugger. To go to a step in the trace, click on it and the source code browser will jump to the corresponding line. You can then navigate through the trace using either the up/down arrows on top of the trace dialog, or by using the up/down arrow keys on your keyboard.



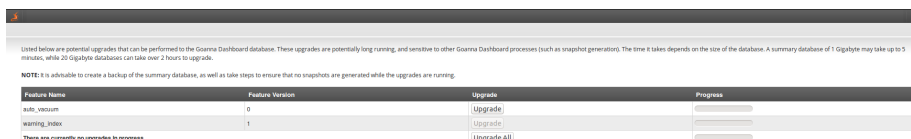
## 8.5 Database Upgrades

Goanna 3.3.0 introduces two optional “database upgrades” you can perform to improve the performance of the Goanna Dashboard.

When you open Goanna Dashboard with the existing database, you will see the following notification:

[Click here](#) to perform database maintenance required to improve the responsiveness of the Goanna Dashboard. ✕

Clicking the link will show a list of available optional upgrades:



Feature Name	Feature Version	Upgrade	Progress
auto_vacuum	0	Upgrade	<input type="text"/>
warning_index	1	Upgrade	<input type="text"/>
There are currently no upgrades in progress		Upgrade All	<input type="text"/>

Goanna 3.3.0 ships with the following optional upgrades (called *features*):

- **auto\_vacuum:** In previous versions of Goanna, deleting a snapshot or a project did not immediately delete corresponding data from the database. This may result in the database size to never shrink.  
Performing this upgrade causes Goanna to delete all residual data from removed snapshots and projects, and set up the database so that when a snapshot or a project is removed, Goanna removes corresponding data immediately. This ensures that the database size is always minimal.
- **warning\_index:** Performing this upgrade causes Goanna to apply optimizations to database indexes, resulting in improved performance, especially when loading Warnings Browser view.

To apply an upgrade, click “Upgrade” button next to a desired feature. Alternatively, clicking “Upgrade All” button will apply all available upgrades.

### Important Notes

We recommend that you take a backup of the database before applying any of the upgrade. The database is located in the following location:

- On Windows XP and Windows Server 2003:  
C:\Documents and Settings\*<name-of-user-who-installed-goanna>*\Local Settings\Application Data\RedLizards\Goanna Central\summary.goannadb
- On Windows Vista, Windows Server 2008, and all later versions of Windows:  
C:\Users\*<name-of-user-who-installed-goanna>*\AppData\Local\RedLizards\Goanna Central\summary.goannadb
- On Linux:  
*<installation-directory>*/reporter/summary.goannadb

You should not run Goanna analysis, interact with the Goanna Dashboard, or otherwise run any Goanna commands, until the upgrade is complete.

Performing upgrade will take a long time depending on the size of the database. If your database is significantly large, this may take hours to complete.



## 8.6 Project Settings (Advanced)

The Goanna Dashboard should work for your project out of the box. However, there are a few advanced settings available if you want to customise the behaviour.

Settings are applied on a per-project basis. To access your project's settings click the gear menu in the top right hand corner after loading your project (through the project page), and select Project Settings.

### 8.6.1 External Code Browser Support

Goanna Dashboard provides an ability to link a snapshot to a particular commit in your version control system, so that it can provide a link to a web-based source code browser, instead of using the Goanna Dashboard's source code browser.

To do this, you will first need to pass `--revision=<vcs-revision>` option to `goanna` or `gokeil` when running analysis. This will link the analysis run (i.e. the Goanna Dashboard snapshot) to the specified revision of the version control system.

Once this is complete, you can provide a template of URL to the source code browser in Project Settings dialog.

**Important:** Enabling external code browser support will disable Goanna Dashboard's source code browser.

### 8.6.2 Code Browser Character Encodings

By default the Dashboard will recognize source files in ASCII and UTF-8 (and additionally on Windows, the default character encoding used by your system). If your source files are not in any of these encodings, you will need to specify the encoding here in order for it to display correctly in the source code browser. A link is available in the Project Settings window listing all the encodings supported.

## 9 Using Goanna With SonarQube Code Quality Platform

### 9.1 Introduction

SonarQube (previously known as “Sonar”) is an open platform to manage quality of a software project (see <http://www.sonarqube.org/>).

Goanna Central provides a mechanism to publish the analysis results to existing SonarQube installation. See the next section for instructions on how to set up Goanna Central to integrate with SonarQube.

Goanna Central also bundles Goanna customized version of SonarQube 3.2.1; see 10.10 for details.

**Important:** On Windows, SonarQube integration support, and the bundled SonarQube are not installed by default. If you have already installed Goanna Central without SonarQube support, then you will first need to uninstall Goanna Central, and then re-install Goanna Central with “Codehaus Sonar” option; see 2.3 for details.

### 9.2 Setting Up Goanna To Integrate With SonarQube Installation

#### Installing Goanna SonarQube Plugin

First, you will need to download Goanna enabled version of SonarQube C++ Community plugin from: <http://archive.redlizards.com/sonar-cxx-plugin-0.1.jar>.

Once downloaded, copy the downloaded .jar file to SONARQUBE\_HOME/extensions/plugins.

Finally, restart the SonarQube server.

#### Configuring Goanna To Use Your SonarQube Installation And Database Server

Goanna Central bundles and uses SonarQube Runner (sonar-runner) version 2.0 to publish the analysis result to SonarQube. Because of this, you will need to modify sonar-runner.properties file that is *bundled with Goanna Central* to ensure that Goanna knows where the SonarQube installation is, and which database server to use.

To do so, follow these steps:

1. Navigate to `<goanna-installation-directory>/sonar/sonar-runner-2.0/conf`.
2. Open `sonar-runner.properties` file with any text editor.
3. Edit this file according to your SonarQube installation to specify the SonarQube server location, database server location, database username and password. Generally, you will need to edit the following settings:
  - **sonar.host.url:** URL to the SonarQube installation
  - **sonar.jdbc.url:** JDBC address to the database server; corresponds to `sonar.jdbc.url` property in `<sonarqube-installation-directory>/conf/sonar.properties` file.
  - **sonar.jdbc.driver:** JDBC driver to use. Generally, you will only need to uncomment the line that corresponds to your database server.
  - **sonar.jdbc.username:** User name of the database server.
  - **sonar.jdbc.password:** Password of the database server.

For more details, see SonarQube installation guide (<http://docs.codehaus.org/display/SONAR/Installing>) and SonarQube Runner installation guide (<http://docs.codehaus.org/display/SONAR/Installing+and+Configuring+SonarQube+Runner>).

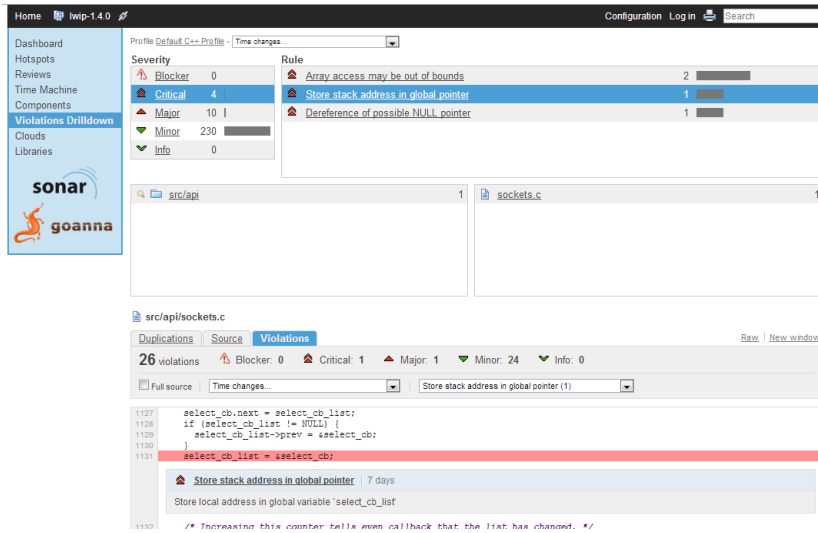
### 9.3 Running Goanna Analysis With SonarQube Publish

Once the set up is complete, you can now perform Goanna analysis and publish the results to SonarQube by passing **--sonar** option to `goanna` or `gokeil` when running analysis.

Once the analysis is complete, the analysis result should be available in the SonarQube dashboard.



Goanna warnings can be found in the Violations Drilldown for the project.



## 10 Advanced Features, Concepts and Configurations

### 10.1 Proceed With Caution

This section describes a number of advanced Goanna concepts and configuration options. In most cases, there is no need to know these concepts or use these options. We recommend that you use these advanced options only when required.

Many of these advanced features contain limitations and/or caveats; if you wish to use any of these features, then make sure to read the description carefully to understand these limitations and caveats.

**Important:** Red Lizard Software may deprecate, and later remove, any feature within this section in the future releases of Goanna.

### 10.2 Manually Running Analysis On Source Files

`goannacc` and `goannac++` utilities can be used to run Goanna analysis on C or C++ source file(s), without first performing build recording.

To run Goanna analysis directly on C source files, run:

```
goannacc --nc --with-cc=<path-to-C-compiler> --db=<path-to-database> <extra-arguments-to-goannacc> <arguments-to-compiler>
```

To run Goanna analysis directly on C++ source files, run:

```
goannac++ --nc --with-cxx=<path-to-C++-compiler> --db=<path-to-database> <extra-arguments-to-goannac++> <arguments-to-compiler>
```

For example, to run analysis on `src/main.c` (when the source file is written for `gcc` compiler):

```
goannacc --nc --with-cc=gcc --db=myfiles.goannadb src/main.c
```

This will run analysis on `src/main.c`, and outputs the result to the console.

Additionally, this command stores information about the analysis into a specified *project database* file via `--db=<path-to-database>` option (see 10.7 for details). Passing `--db` option is essential to ensure the maximum accuracy of the analysis, and allows Goanna to cache some information to speed up subsequent analysis.

You can pass most analysis configuration options (such as `--check` option) to `goannacc` or `goannac++` to control the analysis. See 11.7 for the complete list of available options.

#### Compile and Run Analysis At The Same Time

`goannacc` and `goannac++` can also act as compiler wrappers; that is, they can be used to compile source files and perform analysis on these files at the same time.

To compile and run analysis at the same time, run the same command as above, but without `--nc` option. For example:

```
goannacc --with-cc=gcc --db=myfiles.goannadb src/main.c
```

This will compile `src/main.c` using `gcc`, and then performs analysis on that file.

**Note:** Goanna will not perform analysis in this mode if the specified files fail to compile. If this behaviour is not desirable, pass `--ignore-errors` option.

### 10.3 Manually Running Link Time Analysis On Object Files

Similar to source files, it is possible to perform link time analysis directly using `goannald`.

Because link time analysis relies on information in the project database about source files, you must first run source file analysis on all relevant source files before running link time analysis.

Once source file analysis is complete, run the following command to run link time analysis:

```
goannald --nc --with-ld=<path-to-linker> --db=<path-to-database> <extra-arguments-to-goannald> <arguments-to-linker>
```

For example, to run analysis on `src/main.o` object file (when using GNU Linker (`ld`)):

```
goannald --nc --with-ld=ld --db=myfiles.goannadb src/main.o
```

This will run link time analysis on `src/main.o`, and outputs the result to the console.

#### Link and Run Link Time Analysis At The Same Time

Like `goannacc`, `goannald` can also act as a linker wrapper to link object files and perform link time analysis at the same time.

To link and run link time analysis at the same time, run the same command as above, but without `--nc` option. For example:

```
goannald --with-ld=ld --db=myfiles.goannadb src/main.o
```

This will perform linking with an object file `src/main.o` using `ld`, and then performs link time analysis on that object file..

**Note:** Goanna will not perform link time analysis in this mode if the specified object files fail to link. If this behaviour is not desirable, pass `--ignore-errors` option.

### 10.4 Build And Run Analysis On A Project At The Same Time

It is possible to use `goannamake`, `goannascons` and `goannaiarbuild` utilities to perform build and run analysis at the same time. This is called *Compile & Analyze* mode. Build specification files are not generated in this mode.

`goannacmake-conv` also supports similar mode where it reads a CMake compilation database and performs analysis based on information in the compilation database. This is called *Convert & Analyze* mode (in some cases, the term *Compile & Analyze* mode also includes *Convert & Analyze* mode).

To perform build and analysis at the same time with `goannamake`, `goannascons` or `goannaiarbuild`, follow these steps:

1. Make sure that your project is ready for build integration (see 3). This is required even in *Compile & Analyze* mode since Goanna utilities use the same mechanism to perform analysis.
2. Similarly, make sure that `PATH` and any other environment variables are set appropriately for build.
3. Clean your build.
4. Now run `goannamake`, `goannascons` or `goannaiarbuild`, in the same way as for build recording, but without `--record` option. For example, for GNU Make project:

```
goannamake all
```

This will build your project and perform analysis at the same time.

To read a CMake compilation database and analysis at the same time with `goannacmake-conv`, follow these steps:

1. Clean your build.
2. Perform a CMake build, with a special option `CMAKE_EXPORT_COMPILE_COMMANDS` set to `ON` to generate a compilation database.

3. Now run `goannacmake-conv`, in the same way as for normal build specification generation, but without `--record` option. For example:

```
goannacmake-conv
```

### Important Notes

- `gotrace` (for Keil  $\mu$ Vision projects) does not support this mode of analysis.
- If you use *Compile & Analyze* mode on recursive Makefile projects, then some Goanna operations may execute multiple times. This means that:
  - Multiple HTML reports may be generated for a single run if you use HTML report option,
  - Multiple XML output files may be generated for a single run if you use XML output option,
  - Goanna may publish the same analysis run multiple times to SonarQube if you use SonarQube publish option, and
  - CppCheck or CppNcss may execute multiple times, if CppCheck or CppNcss is enabled.

## 10.5 Using Embedded Build Information To Perform Analysis

During build recording (or, in the case of `goannacmake-conv`, conversion to Goanna build specification), in addition to build specification files, `goannamake`, `goannascons`, `goannaiarbuild` and `goannacmake-conv` utilities also store build information to project database (this is called *database buildspec*).

It is possible to use the database buildspec to perform analysis, rather than using build specification files. To do this, remove a path to build specification file from the `--record` option during build recording. For example, for GNU Make project:

```
goannamake --record all
```

This command then performs full build and record the build, without generating a build specification file. Instead, you should see a file with an extension `.goannadb` in the working directory of `goannamake`, `goannascons`, `goannaiarbuild` or `goannacmake-conv`. This is a project database file and contains build information (database buildspec).

To run analysis with the database buildspec, remove a path to build specification file from the `--analyze` option of `goanna`. For example:

```
goanna --analyze
```

This command runs analysis with the database buildspec.

**Important:** `gotrace` and `gokeil` (for Keil  $\mu$ Vision projects) do not use the database buildspec and do not support this mode of analysis.

## 10.6 Append New Build Information Into Existing Build Specification

By default, when recording your build, `goannamake`, `goannascons` and `goannaiarbuild` utilities overwrite the existing build specification file. This ensures that the build specification file contains the most accurate information about your build.

It is possible to instruct these utilities to instead append build information into the end of the existing build specification file. Using this feature allows you to perform incremental build (rather than full build) when recording the build to add new files.

However, if you use this feature, note that this is *only* useful when you *only added new files* into your build. If you have removed files from the build, or modified build settings, then you need to perform full build recording by re-running full build.

To perform an incremental build and add new build information into the existing build specification with `goanna-make`, `goannascons` or `goannaiarbuild`, follow these steps:

1. Add new files into your build.
2. Make sure that PATH and any other environment variables are set appropriately for build.
3. **Do not** clean your build; this feature works only if you perform incremental build.
4. Now run `goannamake`, `goannascons` or `goannaiarbuild`, in the same way as for build recording, but with **--record-incremental** option. For example, if your project uses GNU Make and the existing build specification file is called `myproject.goannabuild`:

```
goannamake --record=myproject.goannabuild --record-incremental all
```

This will perform an incremental build of your project, and add information about new files into the existing build specification.

**Important:** `goannacmake-conv` and `gotrace` do not support this feature.

## 10.7 The Project Database

When analyzing a whole project, Goanna stores information about the project in a database file (called *project database* or *project DB*) that is used by all the `goanna` analysis commands. The project database is also used by `goreporter` to summarize information for the Goanna Dashboard.

The project database file by convention has the extension `.goannadb`, and usually resides at the top directory of your project.

Goanna stores the following information in the project database:

- Interprocedural (or whole-program) analysis information, that is, information about bugs that occur as a result of calls between functions.
- Information about your project's build when you use the `--record` and `--analyze` options with no file name argument (database buildspec).
- Cached information about your source files, which enables incremental analysis (that is, re-analysis of a project after small changes) to be performed much faster.
- Suppression information, if you have suppressed warnings.

By default, build integration utilities (`goannamake`, `goannascons`, `goannaiarbuild` and `goannacmake-conv`) and project-wide analysis utilities (`goanna` and `gokeil`) use a database file taken from the name of the project root directory with `.goannadb` added. If you wish to use a different database file, you can override this default by specifying `--db=<file>`.

You should not delete this database file unless when required. If you delete this database file, the next analysis run take much longer (due to cached information being removed), and you will also lose warnings suppression information.

Unlike other Goanna utilities, if you use `goannacc`, `goannac++`, or `goannaId` directly to perform analysis (see [10.2](#)), then you must explicitly specify the location of the project database with `--db=<file>` option. Not specifying this option causes:

- Goanna analysis to produce inaccurate results because information needed for interprocedural analysis becomes unavailable,
- link-time analysis to become unavailable (link-time analysis relies on the project database), and
- Warnings suppression functionality to not work.



## 10.8 How Goanna's Compiler Support Work

Most C/C++ code uses non-standard language extensions to some extent. Even if your own code is 100 percent standard compliant, you must almost certainly include libraries and header files that use non-standard extensions provided by the compiler.

Goanna analyzes C/C++ code at a very deep semantic level. In order to do this, it must analyze your source code exactly as understood by your compiler, including all the same system headers, built-in predefined macros, and language extensions.

In some dialects, the macros and include paths predefined in the compiler are not fixed, but vary depending on your operating system, how the compiler has been configured and built, and even on what command line options are given to it at run time.

Goanna utilities are designed to handle all these complexity automatically, so that Goanna can fully understand all of your C/C++ source files, even if they contain compiler-specific syntax extensions or include library headers in other directories.

When you perform a project-wide analysis, Goanna automatically does the following things to ensure Goanna can understand your C/C++ source files:

1. Goanna switches its C/C++ parsing mode, based on information in the build specification. Each parsing mode is called *dialect in use*,
2. Goanna detects all predefined macros, include paths, and any other built-in configuration options from various sources (usually by invoking a compiler in question, or fetching information from environment variables), and
3. Goanna reads all compiler-specific command line options and configures the parser accordingly to ensure any syntax related options are applied correctly.

If you run source file or link time analysis manually by directly calling `goannacc`, `goannac++` or `goannald`, then you need to ensure that you pass `--with-cc=<path-to-C-compiler>`, `--with-cxx=<path-to-C++-compiler>`, and `--with-ld=<path-to-linker>` options. These options specify which compilers and linkers your project uses, and ensures that Goanna is configured appropriately for your environment.

## 10.9 Suppressing Warnings Manually Using goannacc

In addition to the Goanna Dashboard (see 8.2), goannacc also provides a way to specify warnings to be suppressed directly. However, if you wish to use this method, note that the bug statuses for any warnings that are suppressed this way will not be synced back to the Goanna Dashboard.

Warning suppression commands of goannacc are based on the concept of *warning ID*. In Goanna, every warning issued has a unique ID assigned to it; this can be seen by passing **--warning-ids** option to goanna, gokeil, goannacc, goannacc++ or goannald. Passing this option causes Goanna to show warning ID in the output, for example:

```
foo.c:254: warning:7: Goanna[PTR-null-fun-pos] Severity-High, Function call
'get_obj(o)' is immediately dereferenced, without checking for NULL.
CERT-EXP34-C,CWE-476
```

Notice that there is now a number (7) after the text warning; this is the warning ID of this warning. In this particular case, this warning was assigned an ID of 7.

### 10.9.1 Suppressing Warnings

To suppress a warning, run:

```
goannacc --suppress=<warning-id> --db=<path-to-project-database>.
```

This will "suppress" the specified warning and cause it to not be displayed in future analysis runs.

Note that suppressed warnings are only ignored; they are still recorded into the project database by Goanna during the analysis. You can use **--suppression-status** option during the analysis to show all warnings (including suppressed ones); see 10.9.3.

**Important:** Since warning suppression information is stored in the project database, you need a project database file to suppress a warning. In addition, if you call goannacc, goannacc++ or goannald manually to perform analysis, then you must make sure to always pass **--db** option every time to ensure suppression information is loaded.

### 10.9.2 Un-suppressing Warnings

To un-suppress a warning, run:

```
goannacc --unsuppress=<warning-id> --db=<path-to-project-database>.
```

This will "un-suppress" the specified warning and cause it to be displayed again in future analysis runs.

### 10.9.3 Displaying Warnings Status

To display all warnings, including suppressed ones, pass **--suppression-status** option to goanna, gokeil, goannacc, goannacc++ or goannald; for example:

```
goanna --suppression-status --analyze=<build-specification-file>
```

This will cause Goanna to show all warnings in the output, including suppressed ones:

```
test.c:4: warning:*: Goanna[CST-local] ...
test.c:8: warning:: Goanna[PTR-unchk-param] ...
test.c:15: warning:: Goanna[ARR-inv-index-ptr-pos] ...
test.c:15: warning:*: Goanna[PTR-null-pos-assign] ...
```

An asterisk (\*) symbol next to warning text indicates that the warning is suppressed.

## 10.10 Using Goanna Central Bundled SonarQube Installation

Goanna Central bundles Goanna customized version of SonarQube 3.2.1. It is possible to use the bundled version of SonarQube with Goanna instead of integrating into the existing installation of SonarQube.

### 10.10.1 Running Goanna Central Bundled SonarQube On Linux

On Linux, the bundled SonarQube is installed by default but not started automatically.

To start SonarQube, follow these steps:

1. Navigate to `<goanna-installation-directory>/goanna/sonar/sonar-3.2.1/bin/linux-x86-64` on 64-bit Linux, or to `<goanna-installation-directory>/goanna/sonar/sonar-3.2.1/bin/linux-x86-32` for 32-bit Linux.

2. Run:

```
./sonar.sh start
```

to start SonarQube server.

To stop the SonarQube server, run:

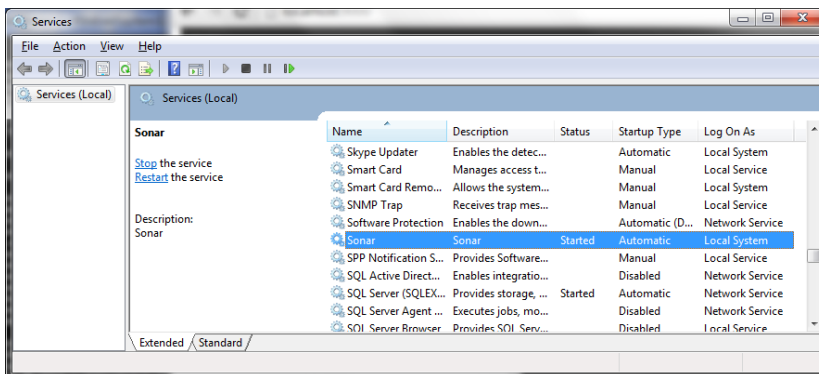
```
./sonar.sh stop
```

### 10.10.2 Running Goanna Central Bundled SonarQube On Windows

On Windows, SonarQube is an optional component, so be sure to check the ‘Codehaus Sonar’ option when installing Goanna Central if you would like SonarQube installed.

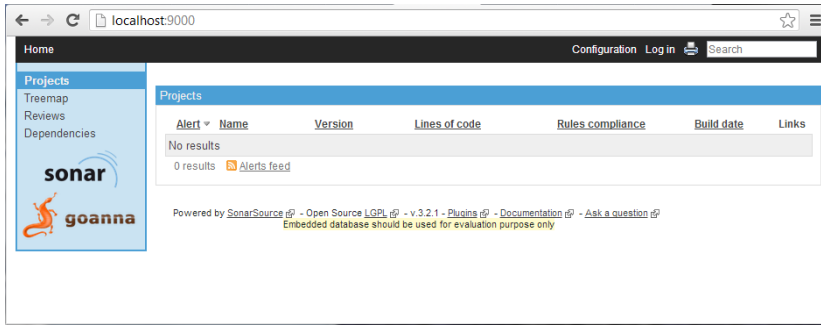
**Note:** If you have already installed Goanna Central without the bundled SonarQube, then you will need to first uninstall Goanna Central, and then re-install Goanna Central with “Codehaus Sonar” option to install the bundled SonarQube.

When Goanna Central is installed with “Codehaus Sonar” option, it will also install a Windows service (called “Sonar”) to start the SonarQube server at the computer startup. You can also start and stop the service manually from Services window.



### 10.10.3 Browsing Goanna Central Bundled SonarQube Dashboard

Once started, you can browse projects in the SonarQube dashboard by going to <http://localhost:9000/>.



#### 10.10.4 Running Goanna Analysis With Goanna Central Bundled SonarQube

To run analysis and publish the analysis results to the bundled SonarQube, pass **--sonar** option to `goanna` or `gokeil` when running analysis.

## 10.11 Using CppCheck And CppNcss With Goanna

### Important Notes

- CppCheck and CppNcss are third party components bundled with Goanna Central. Red Lizard Software does not offer full support for any usage of CppCheck and CppNcss.
- On Windows, CppCheck and CppNcss are not installed by default. If you have already installed Goanna Central without CppCheck or CppNcss support, then you will first need to uninstall Goanna Central, and then re-install Goanna Central with “Codehaus Sonar” option; see 2.3 for details.

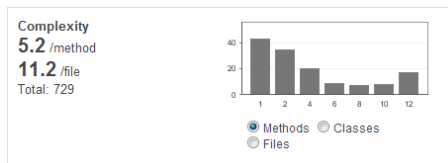
Goanna Central bundles CppCheck 1.56 (see <http://cppcheck.sourceforge.net/>) and CppNcss 1.0.3 (C++ Non-commenting Source Statements; see <http://cppncss.sourceforge.net/>) tools to run additional analysis, and metrics calculation (specifically, Non Commenting Source Statements and Cyclomatic Complexity Number) on the project.

To run CppCheck in addition to Goanna analysis, pass **--cppcheck** option to `goanna` or `gokeil` when running analysis. In addition to Goanna analysis results, results from CppCheck will be saved to a XML file called `cppcheck-result-1.xml`.

To run CppNcss in addition to Goanna analysis, pass **--cppncss** option to `goanna` or `gokeil` when running analysis. In addition to Goanna analysis results, results from CppNcss will be saved to a XML file called `cppncss-result-1.xml`.

Results from CppCheck and CppNcss will not be published to the Goanna Dashboard. Additionally, CppCheck and CppNcss results will not appear in HTML report (with `--html-report`) or XML report (with `--output-xml`) either.

However, if `--sonar` option is also passed, CppCheck and CppNcss results will be published to SonarQube, in addition to Goanna analysis results. In SonarQube, CppCheck warnings will be visible in the Violations Drilldown for the project, alongside Goanna warnings. The CppNcss metrics are visible by adding a Complexity widget for your project.



**Important:** CppCheck and CppNcss are third party components and use different C/C++ parser from the rest of Goanna. Because of this, CppCheck and CppNcss do not support most compiler-specific syntax extensions, and some C++11 features.

In addition, any compiler arguments passed to Goanna (such as `gcc`'s `-I` and `-D` to specify include paths and predefined macros) are not passed to CppCheck or CppNcss. This may lead to parse errors when running CppCheck or CppNcss.

## 11 Goanna Central Utility Reference

### 11.1 goanna – Analyze C/C++ Projects

#### Synopsis

```
goanna --analyze=<build-specification-file> [options] ...
```

#### Description

goanna is the main command used to analyze whole projects. There are many more analysis commands in the Goanna suite of utilities (goannacc, goannac++, goannamake, goannaId, goreporter, and so on). But most of the time you will just run goanna, and it will run all the other analysis commands for you, depending on what needs to be done in your project.

Whenever you run goanna to analyze a project, you must give it a *build specification* to tell it how your project is organized. This build specification file is automatically generated by goannamake, goannascons, goannaIar-build and goannacmake-conv build integration utilities.

#### Command Line Options

- analyze=<file>**, **--analyse=<file>**, **--replay=<file>**, **--build=<file>** Analyze a project from the build steps previously recorded with goannamake --record. If <file> is not specified, the DB is used.
- color**, **--colour** Only available on Linux. Output in color.
- db=<file>** Specify the database file to use for persistent information.
- exclude=<file>** Exclude the specified <file> from analysis.
- help** Print help message for common options.
- jobs=<number>** The number of parallel jobs (1-20) to run simultaneously. Default is the number of CPU cores detected on your system, note that running goanna --help will display the number of detected cores as the default.
- html-report=<output type>** After analysis, also generate analysis report files in HTML format. You can optionally specify type of HTML reports to be generated ('summary', 'warnings' or 'all'). --html-report with no type will generate all available reports.
- output-xml=<file>** After analysis, also output warnings in XML format to <file>.
- sonar=<url>** Publish results to SonarQube server at <url>. --sonar without <url> uses SonarQube's default, typically <http://localhost:9000>.
- summary-db=<file>** The database in which to save snapshots.
- version** Print version information.

#### Advanced Command Line Options

The following options are intended to be used only in cases where your environment requires them. In general, you do not need to use these options.

- advanced-help** Print help message for advanced options.
- cppcheck** Also run CppCheck on the project.
- cppncss** Also run CppNcss on the project.
- html-report-location=<directory>** The directory to output a HTML report to. Default is current directory.
- no-goanna-path** When running goannacc etc. subprocesses, use just the executable name, not the full path. This is useful to work around programs that can't handle long path names, but does require you to place the Goanna bin directory in your PATH environment.

- `--no-snapshot` Don't take a snapshot of the project.
- `--no-sonar-runner` Generate the necessary files for "sonar-runner", but do not run it
- `--revision=<number>` The revision number in your version control system.
- `--sonar-dir=<dir>` Specify the root directory of source files. Can be specified multiple times. Default: current working directory.
- `--ungroup` With `--jobs=2` or more, print all output immediately instead of grouped by job. This reduces latency and memory usage but may cause output from different jobs to be mixed.

## Diagnostics Command Line Options

The following options are provided for diagnostics purposes only. **Do not use these options unless directed by Red Lizard Software support team.**

- `--record=<file>`
- `--sonar-db-name=<database name>`
- `--sonar-db-pass=<password>`
- `--sonar-db-user=<user name>`

Any unrecognized options will be passed through to the executables called by goanna; specifically goannacc, goannac++, or goannaId.

## Return Codes

This section describes goanna return codes when running in Build Mode only. For information about return codes in Record Mode, refer to the corresponding build integration interface (for example, the documentation for goannamake).

**0 (Zero)** A return code of 0 generally means that no errors were encountered, and goanna completed execution successfully.

- goanna completes execution without errors
- goanna prints help or version information

**1 (One)** A return code of 1 generally means that a user-provided parameter is invalid.

- An argument to goanna needs to be a positive integer, but the user-provided value is not a positive integer. For example, timeout values
- goanna encounters an error while parsing command-line arguments

**Inherited Return Values** goanna returns the following return codes if one of the sub-programs goanna uses encounters an error, in the following order:

- If goanna is interrupted by a UNIX signal, its return code is the UNIX signal number
- If goanna encountered an error when running the build specification replay engine or analysing source files, its return code is the goannacc or goannaId return code, depending on what the last used sub-program is
- If goanna encountered an error when publishing to Sonar, or when creating the One Page Summary Report, or when writing to the Dashboard, its return code is the goreporter return code

## 11.2 goannamake – Analyze Makefile Projects

### Synopsis

```
goannamake --record=<build-specification-file> [options] ... [make-options] ...
goannamake --record [options] ... [make-options] ...
goannamake [options] ... [make-options] ...
```

### Description

goannamake is a wrapper for GNU Make, QNX Make and Microsoft NMake utility that can inject Goanna analysis into a Makefile-based project.

goannamake has two modes to perform integration with a project:

- **Build Recording:** Use goannamake to monitor the build process, and generate a build specification file to be used for project-wide analysis.  
See 3.2 for details of this mode.
- **Compile & Analyze (Advanced):** Use goannamake to perform build and analyze at the same time.  
See 10.4 for details of this mode.

### Command Line Options

**--color, --colour** Only available on Linux. Output in color.

**--db=<file>** Specify the database file to use for persistent information.

**--exclude=<file>** Exclude the specified <file> from analysis.

**--help** Print help message for common options.

**--html-report=<output type>** After analysis, also generate analysis report files in HTML format. You can optionally specify type of HTML reports to be generated ('summary', 'warnings' or 'all'). --html-report with no type will generate all available reports.

**--jobs=<number>** The number of parallel jobs (1-20) to run simultaneously. Default is the number of CPU cores detected on your system, note that running goanna --help will display the number of detected cores as the default.

**--microsoft** Only available on Windows. Use Microsoft NMake as the Make utility.

**--output-xml=<file>** After analysis, also output warnings in XML format to <file>.

**--record=<file>** Don't analyze, just record the build steps for later analysis with goanna --analyze. If <file> is specified, also save a build specification file.

**--sonar=<url>** Publish results to SonarQube server at <url>. --sonar without <url> uses SonarQube's default, typically <http://localhost:9000>.

**--summary-db=<file>** The database in which to save snapshots.

**--version** Print version information.

### Advanced Command Line Options

The following options are intended to be used only in cases where your environment requires them. In general, you do not need to use these options.

**--advanced-help** Print help message for advanced options.

**--cppcheck** Also run CppCheck on the project.

**--cppncss** Also run CppNcss on the project.

**--cygwin** Only available on Windows. Use Cygwin version of GNU Make as the Make utility.

**--gnu** Only available on Windows. Use GNU Make as the Make utility. This is the default.



- html-report-location=<directory>** The directory to output a HTML report to. Default is current directory.
- ignore-errors** Ignore errors from the compiler. Note that this will also pass the `-i` flag to the underlying build system.
- no-autodetect** Disable auto detection of make compiler variables.
- no-goanna-path** When running `goannacc` etc. subprocesses, use just the executable name, not the full path. This is useful to work around programs that can't handle long path names, but does require you to place the Goanna bin directory in your PATH environment.
- no-recursive** With `--autodetect`, disable auto detection in all subdirectories of a recursive Make project.
- no-sonar-runner** Generate the necessary files for "sonar-runner", but do not run it
- no-snapshot** Don't take a snapshot of the project.
- record-incremental** With `--record`, append to the existing build spec instead of replacing it.
- revision=<number>** The revision number in your version control system.
- sonar-dir=<dir>** Specify the root directory of source files. Can be specified multiple times. Default: current working directory.

### Diagnostics Command Line Options

The following options are provided for diagnostics purposes only. **Do not use these options unless directed by Red Lizard Software support team.**

- sonar-db-name=<database name>**
- sonar-db-pass=<password>**
- sonar-db-user=<user name>**

Other arguments will be handled in one of either two ways. If the other argument is for a Goanna Central executable (such as `goannacc`, `goannacc++`, or `goanna ld`) then this will be passed on to that executable. Otherwise the argument will be passed to the underlying build system (usually `make` or `nmake`).

### Return Codes

`goannamake` returns the return code from GNU Make, QNX Make or Microsoft NMake.

### Environment

**GOANNAMAKE\_CC, GOANNAMAKE\_CXX, GOANNAMAKE\_LD** Unless you specify `--no-autodetect`, `goannamake` auto-detects the values of the `CC`, `CXX`, etc. variables defined in your Makefile. These values are passed to `goannacc`, `goannacc++`, and `goanna ld` using the `--with-cc`, `--with-cxx` and `--with-ld` options, to aid with auto-detection of your C/C++ dialect and compiler-specific settings.

By default, `goannamake` recognizes several conventional variable names as denoting the C compiler, C++ compiler, and C/C++ linker. If your Makefile does not use one of these conventions, `goannamake` may not inject Goanna correctly or auto-detect your C/C++ dialect correctly. To force `goannamake` to recognize different conventions for Makefile variables, you can set the `GOANNAMAKE_CC`, `GOANNAMAKE_CXX`, and `GOANNAMAKE_LD` environment variables to a colon-separated list of variable names. For example:

```
export GOANNAMAKE_CC=MY_C_COMPILER:MY_C_CROSS_COMPILER
export GOANNAMAKE_CXX=MY_CXX_COMPILER:MY_CXX_CROSS_COMPILER
goannamake
```

The defaults if these variables are not set are:

variable	default value
GOANNAMAKE_CC	CC:HOST_CC:TARGET_CC
GOANNAMAKE_CXX	CXX:HOST_CXX:TARGET_CXX:CCC
GOANNAMAKE_LD	LD

**GOANNAFLAGS** The flags to be passed to `goannacc` (see Section 11.7)

**MAKE** Unless specified, `goannamake` assumes your make utility is `make`. If you use an alternative make utility specify it with this variable. If `--microsoft` is specified, `goannamake` assumes the make utility is `nmake`.

## 11.3 goannascons – Analyze SCons Projects

### Synopsis

```
goannascons --record=<build-specification-file> [options] ... [scons-options]
...
goannascons --record [options] ... [scons-options] ...
goannascons [options] ... [scons-options] ...
```

### Description

goannascons is a wrapper for SCons that can inject Goanna analysis into a SCons project.

goannascons has two modes to perform integration with a project:

- **Build Recording:** Use goannascons to monitor the build process, and generate a build specification file to be used for project-wide analysis.  
See 3.3 for details of this mode.
- **Compile & Analyze (Advanced):** Use goannascons to perform build and analyze at the same time.  
See 10.4 for details of this mode.

### Command Line Options

**--color, --colour** Only available on Linux. Output in color.

**--db=<file>** Specify the database file to use for persistent information.

**--exclude=<file>** Exclude the specified <file> from analysis.

**--help** Print help message for common options.

**--html-report=<output type>** After analysis, also generate analysis report files in HTML format. You can optionally specify type of HTML reports to be generated ('summary', 'warnings' or 'all'). --html-report with no type will generate all available reports.

**--output-xml=<file>** After analysis, also output warnings in XML format to <file>.

**--record=<file>** Don't analyze, just record the build steps for later analysis with goanna --analyze. If <file> is specified, also save a build specification file.

**--sonar=<url>** Publish results to SonarQube server at <url>. --sonar without <url> uses SonarQube's default, typically <http://localhost:9000>.

**--summary-db=<file>** The database in which to save snapshots.

**--version** Print version information.

### Advanced Command Line Options

The following options are intended to be used only in cases where your environment requires them. In general, you do not need to use these options.

**--advanced-help** Print help message for advanced options.

**--cppcheck** Also run CppCheck on the project.

**--cppncss** Also run CppNcss on the project.

**--html-report-location=<directory>** The directory to output a HTML report to. Default is current directory.

**--ignore-errors** Ignore errors from the compiler. Note that this will also pass the -i flag to the underlying build system.

**--no-goanna-path** When running goannacc etc. subprocesses, use just the executable name, not the full path. This is useful to work around programs that can't handle long path names, but does require you to place the Goanna bin directory in your PATH environment.

- `--no-sonar-runner` Generate the necessary files for "sonar-runner", but do not run it
- `--no-snapshot` Don't take a snapshot of the project.
- `--record-incremental` With `--record`, append to the existing build spec instead of replacing it.
- `--revision=<number>` The revision number in your version control system.
- `--sonar-dir=<dir>` Specify the root directory of source files. Can be specified multiple times. Default: current working directory.

### Diagnostics Command Line Options

The following options are provided for diagnostics purposes only. **Do not use these options unless directed by Red Lizard Software support team.**

- `--sonar-db-name=<database name>`
- `--sonar-db-pass=<password>`
- `--sonar-db-user=<user name>`

Other arguments will be handled in one of either two ways. If the other argument is for a Goanna Central executable (such as `goannacc`, `goannac++`, or `goanna1d`) then this will be passed on to that executable. Otherwise the argument will be passed to SCons.

### Return Codes

`goannascons` returns the return code from SCons.

## 11.4 goannacmake-conv – Convert CMake Compilation Database To Goanna Build Specification

### Synopsis

```
goannacmake-conv --record=<build-specification-file> [options] ... <path-to-
compilation-database>
goannacmake-conv --record [options] ... <path-to-compilation-database>
goannacmake-conv [options] ... <path-to-compilation-database>
```

If you do not specify the path to the compilation database file, the default is to read `compile_commands.json`.

### Description

`goannacmake-conv` is a utility which converts a compilation database file generated by CMake to Goanna build specification. See <http://clang.lvm.org/docs/JSONCompilationDatabase.html> for more details about this compilation database file.

`goannacmake-conv` has two modes to perform integration with a project:

- **Conversion Only:** Use `goannacmake-conv` to convert a compilation database file to a Goanna build specification file.  
See 3.4 for details of this mode.
- **Convert & Analyze (Advanced):** Use `goannacmake-conv` to perform conversion (as above), and then run analysis at the same time.  
See 10.4 for details of this mode.

### Command Line Options

`--color`, `--colour` Only available on Linux. Output in color.

`--db=<file>` Specify the database file to use for persistent information.

`--exclude=<file>` Exclude the specified `<file>` from analysis.

`--help` Print help message for common options.

`--html-report=<output type>` After analysis, also generate analysis report files in HTML format. You can optionally specify type of HTML reports to be generated ('summary', 'warnings' or 'all'). `--html-report` with no type will generate all available reports.

`--output-xml=<file>` After analysis, also output warnings in XML format to `<file>`.

`--record=<file>` Specify the name of Goanna build specification to generate. If not specified, save to the project database.

`--sonar=<url>` Publish results to SonarQube server at `<url>`. `--sonar` without `<url>` uses SonarQube's default, typically <http://localhost:9000>.

`--summary-db=<file>` The database in which to save snapshots.

`--version` Print version information.

### Advanced Command Line Options

The following options are intended to be used only in cases where your environment requires them. In general, you do not need to use these options.

`--advanced-help` Print help message for advanced options.

`--cppcheck` Also run CppCheck on the project.

`--cppncss` Also run CppNcss on the project.

`--html-report-location=<directory>` The directory to output a HTML report to. Default is current directory.

- `--no-goanna-path` When running `goannacc` etc. subprocesses, use just the executable name, not the full path. This is useful to work around programs that can't handle long path names, but does require you to place the Goanna bin directory in your PATH environment.
- `--no-sonar-runner` Generate the necessary files for "sonar-runner", but do not run it
- `--no-snapshot` Don't take a snapshot of the project.
- `--revision=<number>` The revision number in your version control system.
- `--sonar-dir=<dir>` Specify the root directory of source files. Can be specified multiple times. Default: current working directory.

### Diagnostics Command Line Options

The following options are provided for diagnostics purposes only. **Do not use these options unless directed by Red Lizard Software support team.**

- `--record-incremental`
- `--sonar-db-name=<database name>`
- `--sonar-db-pass=<password>`
- `--sonar-db-user=<user name>`

Any unrecognized options will be passed through to the executables called by goanna; specifically `goannacc`, `goannacc++`, or `goannaId`.

### Return Codes

`goannacmake -conv` returns 0 if the input CMake compilation database is converted to a Goanna build specification successfully.

`goannacmake -conv` returns -1 if the input CMake compilation database is not in a valid format, or the Goanna build specification cannot be written to the project database.

## 11.5 goannaiarbuild – Analyze IAR Embedded Workbench Projects

### Synopsis

```
goannaiarbuild --record=<build-specification-file> [options] ... [iarbuild-options]
...
goannaiarbuild --record [options] ... [iarbuild-options] ...
goannaiarbuild [options] ... [iarbuild-options] ...
```

### Description

goannaiarbuild is a wrapper for iarbuild.exe utility. Using goannaiarbuild you can integrate Goanna into IAR Embedded Workbench projects.

goannaiarbuild has two modes to perform integration with a project:

- **Build Recording:** Use goannaiarbuild to monitor the build process, and generate a build specification file to be used for project-wide analysis.  
See 3.5 for details of this mode.
- **Compile & Analyze (Advanced):** Use goannaiarbuild to perform build and analyze at the same time.  
See 10.4 for details of this mode.

In addition, Goanna provides a mechanism to perform analysis directly from within IAR Embedded Workbench IDE. See 6.1 for more details.

### Command Line Options

**--color, --colour** Only available on Linux. Output in color.

**--db=<file>** Specify the database file to use for persistent information.

**--exclude=<file>** Exclude the specified <file> from analysis.

**--help** Print help message for common options.

**--html-report=<output type>** After analysis, also generate analysis report files in HTML format. You can optionally specify type of HTML reports to be generated ('summary', 'warnings' or 'all'). --html-report with no type will generate all available reports.

**--output-xml=<file>** After analysis, also output warnings in XML format to <file>.

**--record=<file>** Don't analyze, just record the build steps for later analysis with goanna --analyze. If <file> is specified, also save a build specification file.

**--sonar=<url>** Publish results to SonarQube server at <url>. --sonar without <url> uses SonarQube's default, typically <http://localhost:9000>.

**--summary-db=<file>** The database in which to save snapshots.

**--version** Print version information.

### Advanced Command Line Options

The following options are intended to be used only in cases where your environment requires them. In general, you do not need to use these options.

**--advanced-help** Print help message for advanced options.

**--cppcheck** Also run CppCheck on the project.

**--cppncss** Also run CppNcss on the project.

**--html-report-location=<directory>** The directory to output a HTML report to. Default is current directory.

- no-goanna-path** When running `goannacc` etc. subprocesses, use just the executable name, not the full path. This is useful to work around programs that can't handle long path names, but does require you to place the Goanna bin directory in your PATH environment.
- no-sonar-runner** Generate the necessary files for "sonar-runner", but do not run it
- no-snapshot** Don't take a snapshot of the project.
- record-incremental** With `--record`, append to the existing build spec instead of replacing it.
- revision=<number>** The revision number in your version control system.
- sonar-dir=<dir>** Specify the root directory of source files. Can be specified multiple times. Default: current working directory.

### Diagnostics Command Line Options

The following options are provided for diagnostics purposes only. **Do not use these options unless directed by Red Lizard Software support team.**

- sonar-db-name=<database name>**
- sonar-db-pass=<password>**
- sonar-db-user=<user name>**

Other arguments will be handled in one of either two ways. If the other argument is for a Goanna Central executable (such as `goannacc`, `goannacc++`, or `goannald`) then this will be passed on to that executable. Otherwise the argument will be passed in to the `IarBuild.exe` build system.

### Return Codes

`goanna iarbuild` returns 0 if the IAR build specification is converted to a Goanna build specification successfully. `goanna iarbuild` returns -1 if the input IAR build specification is not in a valid format, or if an unexpected error occurred during the conversion process.

## 11.6 gotrace and gokeil – Analyze Keil™ $\mu$ Vision® Projects

### Synopsis

```
gotrace [gotrace-options] UV4.exe [uv4-options]  
gokeil [goanna-options]
```

**Important:** You need to install Keil  $\mu$ Vision Support Package before using these tools. Contact your distributor to obtain the installer.

### Description

gotrace and gokeil are the utilities that allow Keil™  $\mu$ Vision® projects to be analyzed. The typical analysis process is as follows:

1. gotrace wraps over UV4.exe (command line invocation of Keil  $\mu$ Vision) to monitor the build process of your Keil  $\mu$ Vision project, and generates a build specification containing necessary information to understand how your project is built.
2. gokeil reads the generated build specification file and runs analysis over the source files.

For more details about how these utilities work, see 3.6. For instruction to set up Keil  $\mu$ Vision IDE to allow Goanna analysis to be run from within IDE, see 6.2.

### gotrace Command Line options

- help Print help message for common options.
- version Print version information.
- output-file=<file> Save the build specification into the specified file; default is goanna.goannaspec.
- force-tracing Force build recording even when the specified build spec file already exists.

### Diagnostics Command Line Options

The following options are provided for diagnostics purposes only. **Do not use these options unless directed by Red Lizard Software support team.**

- debug-help

These options must come before UV4.exe. Any arguments after UV4.exe will be passed directly to Keil  $\mu$ Vision.

### gokeil Command Line options

- buildspec=<file> Specify the location of the build specification file; default is goanna.goannaspec.
- color, --colour Only available on Linux. Output in color.
- db=<file> Specify the database file to use for persistent information.
- exclude=<file> Exclude the specified <file> from analysis.
- help Print help message for common options.
- html-report=<output type> After analysis, also generate analysis report files in HTML format. You can optionally specify type of HTML reports to be generated ('summary', 'warnings' or 'all'). --html-report with no type will generate all available reports.
- output-xml=<file> After analysis, also output warnings in XML format to <file>.
- sonar=<url> Publish results to SonarQube server at <url>. --sonar without <url> uses SonarQube's default, typically <http://localhost:9000>.
- summary-db=<file> The database in which to save snapshots.
- version Print version information.



## Advanced Command Line Options

The following options are intended to be used only in cases where your environment requires them. In general, you do not need to use these options.

- `--advanced-help` Print help message for advanced options.
- `--cppcheck` Also run CppCheck on the project.
- `--cppncss` Also run CppNcss on the project.
- `--html-report-location=<directory>` The directory to output a HTML report to. Default is current directory.
- `--no-sonar-runner` Generate the necessary files for "sonar-runner", but do not run it
- `--no-snapshot` Don't take a snapshot of the project.
- `--revision=<number>` The revision number in your version control system.
- `--sonar-dir=<dir>` Specify the root directory of source files. Can be specified multiple times. Default: current working directory.

## Diagnostics Command Line Options

The following options are provided for diagnostics purposes only. **Do not use these options unless directed by Red Lizard Software support team.**

- `--sonar-db-name=<database name>`
- `--sonar-db-pass=<password>`
- `--sonar-db-user=<user name>`

Valid `goannacc`, `goannac++`, or `goanna1d` arguments are also accepted and can be used to configure the analysis. Any other arguments are treated as Keil `armcc`, `C51` or `C166` compiler arguments and can used to configure how the source files are treated by Goanna.

## 11.7 goannacc and goannacc++ – Analyze C/C++ Source Files

### Synopsis

```
goannacc [options] ... [compiler options] ... [file] ...
goannacc++ [options] ... [compiler options] ... [file] ...
```

### Description

goannacc and goannacc++ are the utilities that analyze individual C and C++ source files. In general, goannacc should be used for C source files, and goannacc++ for C++ files.

goannacc and goannacc++ can operate in two modes:

- **Analysis Only:** Perform analysis on the specified source files only.
- **Compile & Analyze:** Compile the specified source files, and then perform analysis.

For more details about these modes, see [10.2](#).

In most cases, there is no need to call goannacc or goannacc++ directly, as these are called automatically by project-wide analysis utilities (goanna and gokeil).

In addition to the above, goannacc is also used to perform some general management tasks, such as:

- Enabling, disabling and installing checks packages. See [5.2](#) for details.
- Suppressing and un-suppressing warnings. See [10.9](#) for details.

### Command Line Options

```
--absolute-path Print absolute paths in warnings.
--all-checks Run all available checks (overrides all other check related options).
--brief-trace Show immediately relevant decisions in trace output, not the majority of decisions.
--c++ Indicate that file(s) contain C++ code.
--check=<name> Run a specific check (overrides any checks file).
--checks-file=<file> Use the checks listed in <file> instead of the default checks in properties.init file.
--checks=<standard> Run all checks in the specified coding standard. For example, --checks=misrac2004 runs all
available checks in the MISRA C:2004 standard.
--checks=<standard>-<rule> Run the check(s) corresponding to one rule in the specified coding standard. For
example, --checks=misrac2004-12.8 runs the check(s) that implement MISRA C:2004 rule 12.8.
--color, --colour Only available on Linux. Output in color.
--columns Print column positions in warnings.
--db=<file> Specify the database file to use for persistent information.
--directory=<dir> Before doing anything, change to <dir>.
--force-analysis Re-analyze files that have not changed since last run.
--help Print help message for common options.
--ignore-errors Ignore errors from the compiler.
--license-server=<server[:port]> Attempt to contact a license server at address <server>. <Port> is optional
(defaults to 6200).
--no-compile, --nc Do not run the compiler.
--output-checks Output the checks that are currently loaded.
--output-format=<format> Specify a warning format used by Goanna to output warnings. The following special
strings in <format> are expanded:
%FILENAME% the filename
```

**%RELFILE%** the filepath and filename  
**%RELPATH%** the filepath  
**%ABSFILE%** the absolute filepath and filename  
**%ABSPATH%** the absolute filepath  
**%DBRELFILE%** the filepath relative to the database file and filename  
**%DBRELPATH%** the filepath relative to the database file  
**%LINENO%** the line number  
**%COLUMN%** the column number  
**%CHECKNAME%** the check identifier  
**%SEVERITY%** the checks severity rating  
**%MESSAGE%** the warning message  
**%RULES%** corresponding rule(s) from coding standards, if any  
**%TRACE%** counter example if any  
**%FUNCTION%** the function name  
**%SUPPRESSED%** a \* if the warning is suppressed  
**%WARNINGID%** the id of this warning in the database  
**%EOL%** a line break  
**%%** a literal %.

The default warning format is: "%FILE%:%LINENO%: warning: Goanna[%CHECKNAME%] - %MESSAGE%%EOL%".

**--project-dir=<dir>** Only include header files in the given directory.

**--quiet** Only display warnings and no other output.

**--suppress=<warning id>** Suppress warning <warning id>.

**--suppression-status** Output suppression status markers, without suppressing warnings.

**--system-headers** Process system header files.

**--timeout=<n>** Set a timeout (in seconds) for analysis of each source file. Default: 240.

**Important:** Setting this value to 0 (meaning infinite) is discouraged; this may cause Goanna to not terminate!

**--trace** Prints out a trace through the function that leads to the warning. This is helpful for understanding why the warning occurs.

**--trace-format=<format>** Specify the format to output traces. The following special strings are used in the trace format:

**%FILENAME%** the filename  
**%RELFILE%** the filepath and filename  
**%RELFILEX%** the filepath and filename followed by ":", or blank if in the current source file  
**%RELPATH%** the filepath  
**%ABSFILE%** the absolute filepath and filename  
**%ABSFILEX%** the absolute filepath and filename followed by ":", or blank if in the current source file  
**%ABSPATH%** the absolute filepath  
**%DBRELFILE%** the filepath relative to the database file and filename  
**%DBRELPATH%** the filepath relative to the database file  
**%FUNCTION%** the function name  
**%LINE%** the line number  
**%TEXT%** text describing the event on the trace  
**%TYPE%** the type of the trace line  
**%EOL%** a line break  
**%%** a literal %.

The default trace format is: "%LINE%: %TYPE% - %TEXT%%EOL%".

**--unsuppress=<warning id>** Unsuppress warning <warning id>.

**--user-headers** Force the processing of user header files.

- verbose** Display additional output information.
- version** Print version information.
- warning-ids** Output warning ids.
- with-cc=<compiler>** Specify the C compiler executable to run (if `--nc` is not specified). Also affects the default dialect when no `--dialect` is specified.
- with-cxx=<compiler>** Specify the C++ compiler executable to run (if `--nc` is not specified). Also affects the default dialect when no `--dialect` is specified.

### Advanced Command Line Options

The following options are intended to be used only in cases where your environment requires them. In general, you do not need to use these options.

- 32** Analyze code for 32-bit targets (longs and pointers are 32 bits wide).
- 64** Analyze code for 64-bit targets (longs and pointers are 64 bits wide).
- advanced-help** Print help message for advanced options.
- dialect=<file>** Specify the dialect of C/C++ compilers. Available dialects are:
 

armcc	microsoft
c166	qnx
c51	renesas-h8
cc21k	renesas-rx
cosmic	tasking-c166
cygwin	ti-cl16x
diab	ti-cl2000
gnu	ti-cl430
iar-8051	ti-cl470
iar-arm	ti-cl500
iar-msp430	ti-cl55
metrowerks	

If you use this option, you should also specify `--with-cc`, `--with-cxx` and/or `--with-ld` to specify the paths to the compiler(s) and linker. If these are not specified, then Goanna will assume the default name for the specified dialect, which may not be what is available on your system.

If none of `--dialect`, `--with-cc`, `--with-cxx` or `--with-ld` are specified, then Goanna will assume the default of `gnu` dialect with `gcc` C compiler, `g++` C++ compiler and `ld` linker.

- error** Exit with error status code when warnings emitted.
- exclude=<file>** Exclude the specified `<file>` from analysis.
- input-encoding=<type>** Specify the character encoding of the source file:
  - us-ascii** ASCII (default)
  - utf-8** UTF-8
  - ansi** (Available on Windows only) default character encoding of the system
- internal-error=<value>** Exit with `<value>` on internal error.
- ipa-iterations=<value>** Specify the number of times interprocedural analysis iterates towards a fixed point. The default is 2.
 

**Important:** Setting this value to 0 (meaning keep iterating until a fixed point is reached) is discouraged; this may cause Goanna to not terminate!
- ipa-trace-depth=<value>** How many levels of inlining are performed for interprocedural traces. Default: 5.
 

**Important:** Setting this value to -1 (meaning infinite) is discouraged; this may cause Goanna to not terminate!
- issue-report=<type>** Control generation of issue report files:
  - never** Never
  - on-failure** On failures only

**on-error** On failures and analysis errors  
**timeout** On failures, errors, and timeouts  
**always** Always (even if successful)

- license-borrow-hours=<number>** When contacting license server, borrow license for <number> of hours. Defaults to 1, maximum of 24.
- license-dir=<directory>** Set directory in which to look for a license file. Defaults to “.”, or \$LMX\_LICENSE\_PATH.
- no-globals** Do not analyze global integer variables.
- no-ipa** Disable interprocedural analysis.
- output-file=<file>** Append warning messages to a specified file.
- output-spec=<file>** Use the contents of <file> as the output-format.
- parse-error-log=<file>** Log parse errors to the specified file instead of stderr.
- timeout-error=<value>** Exit with status code <value> when too many timeouts occur.
- timeout-limit=<value>** Maximum number of per-phase timeouts. Default: 3.  
**Important:** Setting this value to 0 (meaning infinite) is discouraged; this may cause Goanna to not terminate!
- timeout-per-phase=<n>** Set a timeout (in seconds) for each phase of analysis. This is useful if you have a few functions that take very long to analyze and you would like to limit the time spent on these, while still getting as many results as possible on everything else. Default: 60.  
**Important:** Setting this value to 0 (meaning infinite) is discouraged; this may cause Goanna to not terminate!

### Diagnostics Command Line Options

The following options are provided for diagnostics purposes only. **Do not use these options unless directed by Red Lizard Software support team.**

- alias**
- configure=<dialect>**
- dialect-mod=<dialect-mod>**
- dataflow**
- diagnostics-mode**
- no-alias**
- no-dataflow**
- no-default-packages**
- package=<package>**
- package-dir=<directory>**

Any unrecognized options will be passed through to the compiler as compiler arguments, unless **--no-compile** is specified, in which case they will be ignored.

### Return Codes

Here is a list of return codes goannacc returns, listed as headings. Under the headings are a list of modes that provides that return code.

**0 (Zero)** A return code of 0 generally means that no errors were encountered, and goannacc completed execution successfully.

- goannacc completes execution without errors
- goannacc configures the default dialect
- goannacc prints help or version information
- goannacc invokes the C/C++ compiler, but is configured to ignore the compiler return code. goannacc returns 0 even if the compiler returns an error
- goannacc finds code warnings from source code analysis, and is not configured to return an error in the presence of code warnings (this is the default setting, also see return code 1 (One))

**1 (One)** A return code of 1 generally means that a user-provided parameter is invalid.

- goannacc attempts to create or change into a directory that does not exist, or a permission error occurred
- goannacc performs an operation that requires the project database, but the database is not provided or not found
- An invalid license server is provided to goannacc
- An argument to goannacc needs to be a positive integer, but the user-provided value is not a positive integer. For example, the number of IPA<sup>2</sup> iterations, or timeout values
- goannacc encountered an error while parsing command-line arguments or the dialect file
- goannacc finds code warnings from source code analysis, and is configured to return an error in the presence of code warnings

**127**

- goannacc encounters an error when trying to invoke the compiler (for example, if the compiler cannot be found)

**-1**

- goannacc attempts to call goanna`ld`, but goanna`ld` does not exist or cannot be found

### **Inherited Return Values**

- If goannacc invokes the C/C++ compiler, it returns the compiler's return code unless explicitly configured otherwise
- If goannacc invokes goanna`ld`, it returns goanna`ld`'s return code unless explicitly configured otherwise

**Configurable Return Values** The following error modes have configurable return values.

- Internal Error: Goanna encounters an error internally and cannot complete analysis.
- Parser Error: The Goanna C/C++ parser cannot parse the input source code file. This is often attributed to syntax errors in the source code, or uncommon C/C++ language constructs that the Goanna parser does not recognise
- License Expired Error: The Goanna License has expired
- Timeout Error: Goanna is unable to complete before the timeout expired

---

<sup>2</sup>Inter-Procedural Analysis

## 11.8 goanna1d – C/C++ Link Time Analysis

### Synopsis

```
goanna1d [options] ... [linker options] ... [file] ...
```

### Description

goanna1d is the utility that performs link-time analysis on a whole program consisting of multiple source and/or object files, that have already been individually analyzed with goannacc and goannacc++.

goanna1d can operate in two modes:

- **Analysis Only:** Perform link time analysis only.
- **Link & Analyze:** Link the specified object files, and then perform link time analysis.

For more details about these modes, see [10.2](#).

In most cases, there is no need to call goanna1d directly, as this is called automatically by project-wide analysis utilities (goanna and gokeil). goanna1d can also be called by goannacc and goannacc++ if command line arguments have been specified to indicate that a linking step should be done (in your particular compiler). For example, with gcc (gnu dialect), the following commands do *not* invoke goanna1d:

```
goannacc -c a.c
goannacc -c f1.c f2.c
```

but the following commands do:

```
goannacc a.o b.o c.o      # implicitly invokes goanna1d a.o b.o c.o
goannacc f1.o f2.o ver.c # analyzes ver.c, then implicitly invokes goanna1d f1.o f2.o ver.o
```

goanna1d has a set of link time checks that can be specified with check selection options (see [5.1](#)). Further information about the link time checks are available in the Goanna Central Reference Guide.

### Command Line Options

- all-checks** Run all available checks (overrides all other check related options).
- check=<name>** Run a specific check (overrides any checks file).
- checks-file=<file>** Use the checks listed in <file> instead of the default checks in properties.init file.
- checks=<standard>** Run all checks in the specified coding standard. For example, --checks=misrac2004 runs all available checks in the MISRA C:2004 standard.
- checks=<standard>-<rule>** Run the check(s) corresponding to one rule in the specified coding standard. For example, --checks=misrac2004-12.8 runs the check(s) that implement MISRA C:2004 rule 12.8.
- color, --colour** Only available on Linux. Output in color.
- columns** Print column positions in warnings.
- db=<file>** Specify the database file to use for persistent information.
- help** Print help message for common options.
- no-compile, --nc** Do not run the compiler.
- verbose** Display additional output information.
- version** Print version information.
- warning-ids** Output warning ids.
- with-ld=<linker>** Specify the C/C++ linker executable to run (if --nc is not specified).

## Advanced Command Line Options

The following options are intended to be used only in cases where your environment requires them. In general, you do not need to use these options.

**--dialect=<file>** Specify the dialect of C/C++ compilers. Available dialects are:

armcc	microsoft
c166	qnx
c51	renesas-h8
cc21k	renesas-rx
cosmic	tasking-c166
cygwin	ti-cl16x
diab	ti-cl2000
gnu	ti-cl430
iar-8051	ti-cl470
iar-arm	ti-cl500
iar-msp430	ti-cl55
metrowerks	

If you use this option, you should also specify `--with-cc`, `--with-cxx` and/or `--with-ld` to specify the paths to the compiler(s) and linker. If these are not specified, then Goanna will assume the default name for the specified dialect, which may not be what is available on your system.

If none of `--dialect`, `--with-cc`, `--with-cxx` or `--with-ld` are specified, then Goanna will assume the default of `gnu` dialect with `gcc` C compiler, `g++` C++ compiler and `ld` linker.

**--error** Exit with error status code when warnings emitted.

**--exclude=<file>** Exclude the specified *<file>* from analysis.

## Diagnostics Command Line Options

The following options are provided for diagnostics purposes only. **Do not use these options unless directed by Red Lizard Software support team.**

**--diagnostics-mode**

Any unrecognized options will be passed through to the linker as linker arguments, unless `--nc` is specified, in which case they will be ignored.

## Return Codes

Here is a list of return codes `goannald` returns, listed as headings. Under the headings are a list of modes that provides that return code.

**0 (Zero)** A return code of 0 generally means that no errors were encountered, and `goannald` completed execution successfully.

- `goannald` completes execution without errors
- `goannald` prints help or version information
- `goannald` invokes the C/C++ linker, but is configured to ignore the linker return code. `goannald` returns 0 even if the linker returns an error
- `goannald` finds code warnings from source code analysis, and is not configured to return an error in the presence of code warnings (this is the default setting, also see return code 1 (One))

**1 (One)** A return code of 1 generally means that a user-provided parameter is invalid.

- `goannald` encountered an error while parsing command-line arguments
- `goannald` performs an operation that requires the project database, but the database is not provided or not found
- `goannald` finds code warnings from source code analysis, and is configured to return an error in the presence of code warnings



**Inherited Return Values**

- If `goanna ld` invokes the C/C++ linker, it returns the linker's return code unless explicitly configured otherwise

**Configured Return Values** The following error modes have configurable return values.

- Internal Error: Goanna encounters an error internally and cannot complete analysis.

## 11.9 goreporter – Goanna Dashboard Server and Administration Tool, and Publish Analysis Results

### Synopsis

```
goreporter add-project [options]  
goreporter add-snapshot [options]  
goreporter view-projects [options]  
goreporter view-snapshots [options]  
goreporter remove-snapshot [options]  
goreporter export-snapshot [options]  
goreporter export-warnings [options]  
goreporter start-server [options]  
goreporter stop-server [options]  
goreporter db-upgrade [options]  
goreporter html-report [options]  
goreporter publish-sonar [options]  
goreporter output-xml [options]
```

### Description

goreporter is the command line utility used to drive the Goanna Dashboard. It has a number of commands:

**add-project** Add a project to the Goanna Dashboard. Does not take a snapshot of the project.

**add-snapshot** Add a snapshot to a project already in the Goanna Dashboard.

**view-projects** View all the projects in the Goanna Dashboard. This is useful to get a project's id.

**view-snapshots** Lists all the snapshots for the supplied project id. This is useful to get a snapshots' id.

**remove-snapshot** Removes a given snapshot id from the Goanna Dashboard.

**export-snapshot** Exports the files and total warnings by category for a snapshot.

**export-warnings** Exports the warnings for a project

**start-server** Starts the Goanna Dashboard server

**stop-server** Stops the Goanna Dashboard server (when started with the `start -server`) command.

**db-upgrade** Perform the database upgrade; see [8.5](#) for details about this feature.

In addition, `goreporter` is also used to publish the analysis results to HTML report, XML file and SonarQube. The commands are:

**html-report** Generate HTML report files of the analysis result from the project database.

**publish-sonar** Publish the analysis result from the project database to SonarQube.

**output-xml** Generate a XML file of the analysis result from the project database.

### Running The Goanna Dashboard

The Goanna Dashboard is accessible through the embedded web server `goreporter`. The server can be run in two ways; either as a standalone binary being executed from the command line, or as a Linux or Windows Service.

## Standalone Use

To start the server run the following command:

```
goreporter start-server
```

When `goreporter` starts the server, it will display the port the server is running on. The port can be specified by using the `--port` flag. By default the server starts at 1197 and increments until it finds a free port. It is only possible to run one server instance at a time using this command.

To access the server, open a web browser and browse to <http://localhost:<port>>, where `<port>` is the port number `goreporter` starts the server on. To stop the server, run the command:

```
goreporter stop-server
```

## Linux Service

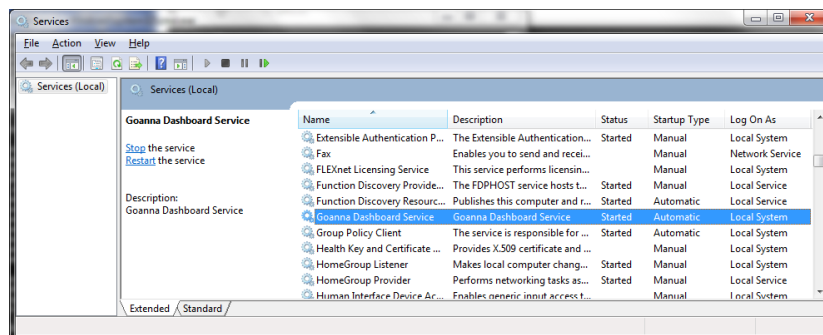
**Note:** Using Linux service feature requires Goanna Central to be installed under root (often with `sudo` command). A service script for the web server is installed as part of the Goanna installation process (if installed with `sudo` or from within the root terminal). It uses the configuration file located at `/etc/goreporter.conf`. This file is used to configure the server, and takes the same options that the `goreporter` executable does.

The service is installed in either `/etc/init.d` or `/etc/rc.d`, depending on your distribution. To start the service on a distribution using `init.d` use:

```
/etc/init.d/goreporterd {start|stop|reset|restart|force-restart}
```

## Windows Service

The Windows service is installed and started on installation of Goanna Central. A configuration file is stored in the same location as your summary database. To change settings with the service, modify the configuration file and then restart the service through the Windows Service Manager.



## Configuration File

To point `goreporter` to a configuration file, use the following command:

```
goreporter start-server --config=goreporter.conf &
```

Where `goreporter.conf` is the name of your configuration file.

`goreporter` configuration files support all the options specified on the command line. The most commonly used of these options are:

**summary-db** the location of the summary database

**port** the port to run the server on

**log** the location to put the `goreporter` log file

They are specified as `<option>=<value>` pairs, with `#` used for comments.

## Running the Dashboard as a Standalone Web Server

The Goanna Dashboard server can be run as a web server for local intranet or remote access in two ways. If you want to run the server on a machine with no other web server running on it, the simplest way to set up a GoReporter server is to run the server either as a service or as a standalone instance with the port set to 80 (standard web port). Provided no firewall or similar is blocking access, the Goanna Dashboard will now be visible by browsing to <http://<yourserveraddress>>. If you run the server standalone and the computer you're running the server on is restarted, the Goanna Dashboard server will not automatically be reset.

## Running the Dashboard through Apache

It is also possible to use the Goanna Dashboard server in conjunction with your existing web server. The following example shows how to do this with Apache web server.

The way to run the Goanna Dashboard with existing Apache installation is to run `goreporter` on the different port than Apache (Apache normally uses port 80), and then use Apache's `mod_proxy` to forward all traffic on the special Dashboard URL to `goreporter`.

Firstly, modify your `httpd.conf` to allow proxy access. For example, if your site is located at <http://mysite.com> and you want to navigate to the Dashboard through the url <http://mysite.com/reporter>, then add these lines into `httpd.conf`:

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
...
ProxyRequests Off
ProxyPass /reporter http://localhost:1197
ProxyPassReverse /reporter http://localhost:1197
```

Then restart Apache.

You can then start the Goanna Dashboard server either as a service or as a standalone instance using the port 1197.

It is important to specify the port number when running `goreporter` in this instance, as Apache has been configured to only connect to `goreporter` on this port.

The Goanna Dashboard is now available at <http://mysite.com/reporter/>.

## Goanna Dashboard Database (Summary Database)

Goanna Dashboard uses its own database (called *summary database*, or *summary DB*) to store details about your project. By default this database is stored in:

- `<installed-directory>/reporter` on Linux where the Goanna installation directory is `<installed-directory>`, or
- `C:\Documents and Settings\<username>\Local Settings\Application Data\RedLizards\Goanna Central\` for Windows XP and Server 2003, or
- `C:\Users\<username>\AppData\Local\RedLizards\Goanna Central\` for Windows Vista, Server 2008 and later, where `<username>` is the name of the user account who installed Goanna Central.

It is possible to change where the summary database is stored by using the `--summary-db=<path>` flag. However, if you do change the default summary database location, be sure to specify new location of the database with either:

- through `--summary-db` flag when you run Goanna commands, or
- by specifying the new location in the `goreporter` configuration file.

## Integrating Reporting into Your Build

Snapshots are automatically taken at the conclusion of project-wide analysis. Generally, there is no need for a manual action to take a snapshot.

However, it is also possible to manually take a snapshot from the project database. To manually take a snapshot, `add-snapshot` command. For example:

## **goreporter add-snapshot --db=project.goannadb**

**Important:** Once the snapshot is taken, make sure that the project database stays at the same location. This is because some features of the Goanna Dashboard relies on the project database being always available at the same location.

### **Command Line Arguments**

#### **add-project**

- summary-db=<file>** The database in which to save snapshots.
- db=<project-db>** The database generated by Goanna for the project (path must be relative to the summary database)
- name=<project name>** This will set the name of a new or existing project to its argument

#### **add-snapshot**

- summary-db=<file>** The database in which to save snapshots.
- db=<project-db>** The database generated by Goanna for the project (path must be relative to the summary database)
- config=<path to configuration file>** When specified overwrites the given options with those specified in the file
- revision=<number>** The revision number in your version control system.
- name=<project name>** This will set the name of a new or existing project to its argument

#### **view-snapshots**

- summary-db=<file>** The database in which to save snapshots.
- config=<path to configuration file>** When specified overwrites the given options with those specified in the file
- project-id=<project-id>** The id of the project to show the data for

#### **export-snapshot**

- summary-db=<file>** The database in which to save snapshots.
- snapshot-id=<snapshot-id>** The id of the snapshot to view or delete. Can be found in the view-snapshots command
- config=<path to configuration file>** When specified overwrites the given options with those specified in the file
- delimiter=<delimiting character>** Character to delimit the csv file. Defaults to comma

#### **export-warnings**

- summary-db=<file>** The database in which to save snapshots.
- project-id=<project-id>** The id of the project to show the data for
- config=<path to configuration file>** When specified overwrites the given options with those specified in the file
- delimiter=<delimiting character>** Character to delimit the csv file. Defaults to comma

#### **start-server**

- summary-db=<file>** The database in which to save snapshots.
- port=<port number>** Port number (default 1197, if 1197 is not available then 1198, 1199...). If this option is set and the specified port is unavailable then the server will quit.
- log=<true/false>** Port number (default 1197, if 1197 is not available then 1198, 1199...). Toggles logging to stdout. Default true
- log-file=<path to log file>** Toggles logging to a file.
- config=<path to configuration file>** When specified overwrites the given options with those specified in the file

## db-upgrade

**Note:** See 8.5 for details about this feature.

- `--summary-db=<file>` The database in which to save snapshots.
- `--progress=<path to progress>` Specify the file to write progress information into.
- `--feature=<feature>` Specify the feature(s) to upgrade; valid values are `auto_vacuum` and `warning_index`.

## html-report

- `--db=<project-db>` The database generated by Goanna for the project (path must be relative to the summary database)
- `--html-type=<report-type>` Specifies which report to generate, can be one of `summary`, `warnings` or `all`. If you do not specify this option, all reports will be generated.
- `--html-report-location=<directory>` The directory to output a HTML report to. Default is current directory.

## publish-sonar

- `--db=<project-db>` The database generated by Goanna for the project (path must be relative to the summary database)
- `--sonar-url=<url>` Specify the location of the SonarQube URL. Default: The default URL of SonarQube, typically `'http://localhost:9000'`
- `--sonar-db-name=<database name>` Specify the SonarQube database name. Default: `'sonar'` .
- `--sonar-db-pass=<password>` Specify the SonarQube database password. Default is to use SonarQube's default, typically `'sonar'` .
- `--sonar-dir=<dir>` Specify the root directory of source files. Can be specified multiple times. Default: current working directory.
- `--no-sonar-runner` Generate the necessary files for "sonar-runner", but do not run it
- `--sonar-exclude=<path-pattern>` Exclude the files whose path is matching the specified pattern.
- `--cppcheck-report=<file>` Specify the path to CppCheck analysis report to include. Default: do not include CppCheck report.
- `--cppncss-report=<file>` Specify the path to CppNcss analysis report to include. Default: do not include CppNcss report.

## output-xml

- `--db=<project-db>` The database generated by Goanna for the project (path must be relative to the summary database)
- `--output-xml-file=<file>` Specify the location of the output XML file. Default: `warnings.xml`

## Return Codes

**0 (Zero)** A return code of 0 generally means that no errors were encountered, and `goreporter` completed execution successfully.

- `goreporter` completes execution without errors
- `goreporter` prints help or usage information

**1 (One)** A return code of 1 generally means that a user-provided parameter is invalid.

- `goreporter` encounters an error while parsing command-line arguments
- `goreporter` attempts to read or write to a file, but the file cannot be found or `goreporter` does not have the correct permissions
- `goreporter` is given a database argument, but the database is not a valid Goanna project database
- `goreporter` attempts to start the web server process on a particular TCP/IP port, but the port is already in use

2

- `goreporter` encounters an unexpected or internal error when processing a subcommand

## Index

- `_GOANNA` preprocessor symbol, 41
- `assert` macro, 41
- `goannac++`, 73
- `goannacc`, 73
- `goannacmake-conv`, 67
- `goannaiarbuild`, 69
- `goannald`, 78
- `goannamake`, 63
  - `GOANNAFLAGS` environment variable, 64
  - `GOANNA_MAKE_CC` environment variable, 64
  - `GOANNA_MAKE_CXX` environment variable, 64
  - `GOANNA_MAKE_LD` environment variable, 64
  - `MAKE` environment variable, 64
- `goannascons`, 65
- `goanna`, 61
- `gokeil`, 71
- `goreporter`, 81
- `gotrace`, 71
- Bundled SonarQube (Advanced)
  - Browsing, 58
  - Service, 58
- Checks
  - Changing check set, 31
  - Custom Packages, 34
  - Disabling Packages, 34
  - Enabling Packages, 33
  - Listing Packages, 33
  - Packages, 33
  - Setting, 31
- Command Line Options
  - `--32`, 75
  - `--64`, 75
  - `--absolute-path`, 73
  - `--advanced-help`, 61, 63, 65, 67, 69, 72, 75
  - `--alias`, 76
  - `--all-checks`, 73, 78
  - `--analyse`, 61
  - `--analyze`, 61
  - `--autodetect`, 64
  - `--brief-trace`, 73
  - `--build`, 61
  - `--buildspec`, 71
  - `--c++`, 73
  - `--check`, 73, 78
  - `--checks`, 73, 78
  - `--checks-file`, 73, 78
  - `--color`, `--colour`, 61, 63, 65, 67, 69, 71, 73, 78
  - `--columns`, 73, 78
  - `--config`, 84
  - `--configure`, 76
  - `--cppcheck`, 61, 63, 65, 67, 69, 72
  - `--cppcheck-report`, 85
  - `--cppncss`, 61, 63, 65, 67, 69, 72
  - `--cppncss-report`, 85
  - `--cygwin`, 63
  - `--dataflow`, 76
  - `--db`, 61, 63, 65, 67, 69, 71, 73, 78, 84, 85
  - `--debug-help`, 71
  - `--delimiter`, 84
  - `--diagnostics-mode`, 76, 79
  - `--dialect`, 75, 79
  - `--dialect-mod`, 76
  - `--directory`, 73
  - `--error`, 75, 79
  - `--exclude`, 61, 63, 65, 67, 69, 71, 75, 79
  - `--feature`, 85
  - `--force-analysis`, 73
  - `--force-tracing`, 71
  - `--gnu`, 63
  - `--help`, 61, 63, 65, 67, 69, 71, 73, 78
  - `--html-report`, 61, 63, 65, 67, 69, 71
  - `--html-report-location`, 61, 64, 65, 67, 69, 72, 85
  - `--html-type`, 85
  - `--ignore-errors`, 64, 65, 73
  - `--input-encoding`, 75
  - `--internal-error`, 75
  - `--ipa-iterations`, 40, 75
  - `--ipa-trace-depth`, 75
  - `--issue-report`, 75
  - `--jobs`, 61, 63
  - `--license-borrow-hours`, 76
  - `--license-dir`, 76
  - `--license-server`, 73
  - `--log`, 84
  - `--log-file`, 84
  - `--microsoft`, 63, 64
  - `--name`, 84
  - `--nc`, 75, 78, 79
  - `--no-alias`, 76
  - `--no-autodetect`, 64
  - `--no-compile`, 76
  - `--no-compile`, `--nc`, 73, 78
  - `--no-dataflow`, 76
  - `--no-default-packages`, 76
  - `--no-globals`, 76
  - `--no-goanna-path`, 61, 64, 65, 68, 70
  - `--no-ipa`, 40, 76
  - `--no-recursive`, 64
  - `--no-snapshot`, 62, 64, 66, 68, 70, 72
  - `--no-sonar-runner`, 62, 64, 66, 68, 70, 72, 85
  - `--output-checks`, 73
  - `--output-file`, 71, 76
  - `--output-format`, 73
  - `--output-spec`, 76
  - `--output-xml`, 61, 63, 65, 67, 69, 71
  - `--output-xml-file`, 85



- package, 76
- package-dir, 76
- parse-error-log, 76
- port, 84
- progress, 85
- project-dir, 74
- project-id, 84
- quiet, 74
- record, 62, 63, 65, 67, 69
- record-incremental, 64, 66, 68, 70
- replay, 61
- revision, 62, 64, 66, 68, 70, 72, 84
- snapshot-id, 84
- sonar, 61, 63, 65, 67, 69, 71
- sonar-db-name, 62, 64, 66, 68, 70, 72, 85
- sonar-db-pass, 62, 64, 66, 68, 70, 72, 85
- sonar-db-user, 62, 64, 66, 68, 70, 72
- sonar-dir, 62, 64, 66, 68, 70, 72, 85
- sonar-exclude, 85
- sonar-url, 85
- summary-db, 61, 63, 65, 67, 69, 71, 84, 85
- suppress, 74
- suppression-status, 74
- system-headers, 74
- timeout, 74
- timeout-error, 76
- timeout-limit, 76
- timeout-per-phase, 76
- trace, 74
- trace-format, 74
- ungroup, 62
- unsuppress, 74
- user-headers, 74
- verbose, 75, 78
- version, 61, 63, 65, 67, 69, 71, 75, 78
- warning-ids, 75, 78
- with-cc, 64, 75
- with-cxx, 64, 75
- with-ld, 64, 78

Snapshot generation, 83

Views, 42

Installation

- Linux, 14
- Windows, 15

Interprocedural analysis, 40

License, 14

- Activation, 15

Motor Industry Software Reliability Association (MISRA) C++:2008, 31

Motor Industry Software Reliability Association (MISRA) C:2004, 31

Running analysis from Keil  $\mu$ Vision, 38

Sample Code, 41

SonarQube, 49

Standards, 31

Traces

- Dashboard, 46

Warning

- Suppression, 42, 45

Warning Suppression With goannacc (Advanced)

- Status, 57
- Suppression, 57
- Un-suppression, 57

Common Weakness Enumeration (CWE), 31

Computer Emergency Response Team (CERT) C/C++ Coding Standard, 31

CppCheck (Advanced), 60

CppCheck and CppNcss With SonarQube (Advanced)

- Complexity widget, 60
- Violations Drilldown, 60

CppNcss (Advanced), 60

Database

- Project, 55
- Summary, 83

Dialect, 56

False positives, 40

- Bug status, 42

Goanna Dashboard, 42

- Bug status, 42
- Server, 81