Extend scons\_tools to switch between MSVC and gcc compiler for Cantata

[1 Installing GCC compilers and gdb debugger 2](#_Toc432088839)

[1.1 Install the GCC compilers coming with the Cantata installer 2](#_Toc432088840)

[1.2 Installing ‘gdb’ debugger 2](#_Toc432088841)

[2 Developing the tool chain 3](#_Toc432088842)

[2.1 Development Steps Overview 3](#_Toc432088843)

[2.2 Introduce a flag to switch between MSVC and gcc 3](#_Toc432088844)

[2.3 Create a Cantata Workspace Preferences file for use with gcc 3](#_Toc432088845)

[2.4 Change the ipg.cop file generation to use the configured compiler 3](#_Toc432088846)

[2.3 Modifying SConscript\_unittests.py 4](#_Toc432088847)

[2.4 Find out which scons-Tool to use with gcc 4](#_Toc432088848)

[2.5 Translate MSVC compiler flags and linker flags to gcc syntax 4](#_Toc432088849)

[2.5.1 Compiler command line 5](#_Toc432088850)

[2.5.2 Linker command line 5](#_Toc432088851)

[3 Installing and using the tool chain 6](#_Toc432088852)

[3.1 Installing the tool chain 6](#_Toc432088853)

[3.2 Enabling gcc/g++ together with Cantata 6](#_Toc432088854)

[3.2.1 Create new test scripts 6](#_Toc432088855)

[3.2.2 Deal with existing Cantata Test Scripts 7](#_Toc432088856)

[4 Use the ‘gdb’ debugger 8](#_Toc432088857)

[4.1 Enable debugging Cantata instrumented source files 8](#_Toc432088858)

[4.2 Enable debug info generation in compiler settings 8](#_Toc432088859)

[4.3 Set additional debugger settings in Cantata frontend 9](#_Toc432088860)

[4.3.1 Setup debugger 9](#_Toc432088861)

[4.3.2 Configure project paths in debugger preference 9](#_Toc432088862)

[5 Possible problems 11](#_Toc432088863)

[6 Future SCons features regarding gcc usage 12](#_Toc432088864)

[7 Additional info: Use other Visual Studio version than VS 2005 (8.0) 12](#_Toc432088865)

# 1 Installing GCC compilers and gdb debugger

## 1.1 Install the GCC compilers coming with the Cantata installer

Run the Cantata Installer and install the GCC compiler and GCC Cantata library if not done. A compiler for C and C++ will be installed. On Microsoft Windows systems, the MinGW, a gcc distribution for Windows, will be installed.

Using Cantata 6.2 on MS Windows, offers gcc 4.6.2.

The table contains the directory structure in Cantata installation directory having both, MSVC and gcc compiler installed:

|  |  |
| --- | --- |
| **MSVC** | **Gcc** |
| **Compiler** | |
| \extras\MSVC6,7,8,9,10Addin (contains installer) | MinGW |
| **header files** | |
| \inc\ports\ x86-Win32-5.x-msvc.h | \inc\ports\ x86-Win32-gcc4.6.2-bundled.h (c)  \inc\ports\x86-Win32-gpp4.6.2-bundled.h (C++) |
| **Cantata library** | |
| \libs\lib-x86-Win32-5.x-msvc8 | \libs\ lib-x86-Win32-gcc4.6.2-bundled  \libs\lib-x86-Win32-gpp4.6.2-bundled |
| **Compiler Section in \bin\cantpp.cfg** | |
| [x86-Win32-5.x-msvc8] | [x86-Win32-gcc4.6.2-bundled]  [x86-Win32-gpp4.6.2-bundled] |
| Please select the Compiler Section in your current workspace, see chapter “3 Create a WorkspacePreferences file for use with gcc” | |

## 1.2 Installing ‘gdb’ debugger

Installation packages:

|  |  |
| --- | --- |
| Cantata 6.2 | \\cw01.contiwan.com\lndp\didk7708\\_public\Tools\Cantata\Installation\Cantata++6.2\ cantata\_installer.exe |
|  | LANDesk: Cantata\_6.2.1.0\_ENU |
| MinGW | MingGW32 + gcc 4.6.4 in included in Cantata installation |

But gdb is missing in Cantata installation.

To install ‘gdb’ run following steps from Windows console (cmd.exe):

1. Test whether application mingw-get comes with the Cantata installation:  
   *> where mingw-get*  
   It should print to stdout <Cantata installation directory>\MinGW\bin\mingw-get.exe
2. Search the gdb package’s name by showing list of all available packages  
   *> mingw-get list*The package is *mingw32-gdb*
3. Install gdb package  
   *> mingw-get install mingw32-gdb*

Alternately, gdb can be installed from exisiting MinGw repository, for example  
<https://mingw-builds.googlecode.com/files/mingw32-gcc-4.6.2-release-c%2Cc%2B%2B%2Cfortran-sjlj-FINAL.7z>

# 2 Developing the tool chain

## 2.1 Development Steps Overview

1. Introduce a flag to switch between MSVC and gcc
2. Create a Cantata Workspace Preferences file for use with gcc
3. Change the ipg.cop file generation to use the configured compiler
4. Find out which scons-Tool to use with gcc
5. Translate MSVC compiler flags and linker flags to gcc syntax
6. Create a new Cantata/Eclipse project generation, find the C++ Builder settings in Cantata (not necessary at the moment)

## 2.2 Introduce a flag to switch between MSVC and gcc

The correct place to put this in unit\_test.scfg

Three flags are necessary, one to distinguish between MSVC and gcc, the other ones to select the compiler section (see <Cantata installation dir>\bin\cantpp.cfg (gcc distinguishes between C and C++)

# Switch between msvc and gcc  
# 1 = MSVC  
# 0 = gcc  
use\_msvc = 0  
# GCC compiler sections for C and C++  
compiler\_section\_gcc = “x86-Win32-gcc4.6.2-bundled”  
compiler\_section\_gpp = “x86-Win32-gpp4.6.2-bundled”  
If you have another version for gcc compilers, than you can change the flags accordingly.

## 2.3 Create a Cantata Workspace Preferences file for use with gcc

The scons command *scons xxx\_ide\_unittests* loads file 05\_Testing\05\_Test\_Environment\algo\modtests\cantata\_tests\<component\_name>\ CantataWorkspacePreferences.epf.  
It contains Cantata settings to select the compiler, setting for test script generation and report generation.  
Only the compiler section needs to be changed.

For GCC usage, replace the following settings:  
/instance/com.ipl.products.eclipse.cantpp.core/com.ipl.products.eclipse.cantpp.core.cfg\_section\_c=x86-Win32-gcc4.6.2-bundled  
/instance/com.ipl.products.eclipse.cantpp.core/com.ipl.products.eclipse.cantpp.core.cfg\_section\_cpp=x86-Win32-gpp4.6.2-bundled

Place file CantataWorkspacePreferencesGcc.epf into folder scons\_templates\05\_Testing\05\_Test\_Environment\algo\modtests\cantata\_tests\xxx.   
Installation:  
The components copy both .epf file to 05\_Testing\05\_Test\_Environment\algo\modtests\cantata\_tests\xxx.

## 2.4 Change the ipg.cop file generation to use the configured compiler

Function to modify: \_GenerateIpgCop() defined in SConscript\_unittests.py.

Because the GCC compiler collection distinguished between C and C++ compiler, three different separation will be needed: MSVC ↔ GCC and GCC than C ↔ C++

|  |  |
| --- | --- |
| gcc | "--comp:x86-Win32-gcc4.6.2-bundled" |
| g++ | "--comp:x86-Win32-gpp4.6.2-bundled" |
| msvc | "--comp:x86-Win32-5.x-msvc8" |

## 2.5 Modifying SConscript\_unittests.py

Function to modify: SetupIDE()

* For Gcc use, turn off MSVC generation: ut\_env.MSVSProject(), ut\_env.MSVSProjectDebug(), ut\_env.MSVSSolution()
* Select the correct preferences file: **ut\_env.EclipseWorkspace()**
* Change global compiler settings and linker settings (see chapter 2.4)

## 2.6 Find out which scons-Tool to use with gcc

At the top of file scons\_tools\scons\_common\_scripts\modtests\SConscript.py, a list of SCons extensions are specified. These extensions are placed in at least following scons\_tools subdirectories: scons\scons-local\SCons\Tool, scons\_adas\_extensions

if use\_msvc == 1:

ut\_env = Environment(# list of scons extensions used

tools = ["msvc", "msvs-patched", "mslib", "mslink",

"msvc-addon", "doxygen", "fingerprint",

"eclipse\_cdt", "unittest"],

# path to scons extensions

toolpath = [scons\_adas\_extensions\_path],

MSVS\_USE\_MFC\_DIRS = 1,

TARGET\_ARCH = "X86",

# path to doxygen tool

DOXYGEN\_TOOLS = doxygen\_dir,

# manifest files need to be included in the dlls/apps

WINDOWS\_EMBED\_MANIFEST = True,

# use Visual Studio 2005

MSVC\_VERSION = "8.0",

# specify processor for scons build

BUILD\_TARGET = "SIM")

else:

ut\_env = Environment(# list of scons extensions used

tools = [**"mingw"**, "doxygen", "fingerprint",

"eclipse\_cdt", "unittest"],

# path to scons extensions

toolpath = [scons\_adas\_extensions\_path],

MSVS\_USE\_MFC\_DIRS = 0,

TARGET\_ARCH = "X86",

# path to doxygen tool

DOXYGEN\_TOOLS = doxygen\_dir,

# specify processor for scons build

BUILD\_TARGET = "SIM")

Running with this changes causes a Python error like “mingw.py: missing g++ tool”.  
To solve this copy scons\_tools\scons\scons-local\Tool\gpp.py to scons\_tools\scons\scons-local\Tool\g++.py

## 2.7 Translate MSVC compiler flags and linker flags to gcc syntax

The MSVC-based module tests are using the same compiler settings like the sim\_swc build of the component. Therefore modtests/SConscropt.py include 04\_Engineering\03\_Workspace\algo\xxx\_sim\sim\_swc\_xxx\ simenv\_config.scfg.  
At least following flags needs to be overwritten to match the gcc syntax because they are accessed in modtests/SConscript.py:

* compiler command line
* Linker command line
* compiler flags (common compiler flags, no include paths)
* linker flags: at the moment the MSVC linker flags are thrown away:

if use\_msvc == 1:

algo\_ut\_ccflags = variant[1]

algo\_ut\_cppdefines = variant[2]

algo\_ut\_linkflags = variant[3]

else:

algo\_ut\_ccflags = "-O0 -g3 -Wall "

algo\_ut\_cppdefines = variant[2]

algo\_ut\_linkflags = ""

**Note:** the **space sign** at the end of algo\_ut\_ccflags is necessary not to fall into compiler command line syntax errors.

### 2.7.1 Compiler command line

C++:

*ut\_env['CXXCOM']="ipg\_comp --optfile " + Dir('#').abspath + "\\ipg.cop " + '--comp $CXX -o ${TARGET.path} -c -fmessage-length=0 $SOURCES $CXXFLAGS $CCFLAGS $\_CCCOMCOM $\_CXXCOMCOM'*

C:

*ut\_env['CCCOM']="ipg\_comp --optfile " + Dir('#').abspath + "\\ipg.cop " + '--comp $CC -o ${TARGET.path} -c -fmessage-length=0 $SOURCES $CCFLAGS $\_CCCOMCOM'*

*Modify in Sconscript\_unittest.py.*

### 2.7.2 Linker command line

Because the gcc compiler tool are using different applications for C and C++ (instead of MSVC) and there’s only one SCons environment variable to store the linker command line, a distinguishment at runtime is necessary to overwrite SCons environment variables ‘LINK’ and ‘LINKCOM’. This is done in function BuildAndRunUnitTest in file SConscript\_unittests.py.

if ut\_source\_exts[utprogram] == '.cpp':

ut\_env['LINK']='$CXX'

ut\_env['LINKCOM']="ipg\_comp --optfile " + Dir(unit\_test\_path).abspath + '\\' + utprogram + '\\ipg.cop --link ' + '$CXX -o $TARGET $LINKFLAGS $\_\_RPATH $SOURCES $\_LIBDIRFLAGS $\_LIBFLAGS'

if ut\_source\_exts[utprogram] == '.c':

ut\_env['LINK']='$CC'

ut\_env['LINKCOM']='ipg\_comp --optfile ' + Dir(unit\_test\_path).abspath + "\\" + utprogram+ '\\ipg.cop --link ' + '$CC -o $TARGET $LINKFLAGS $\_\_RPATH $SOURCES $\_LIBDIRFLAGS $\_LIBFLAGS'

# 3 Installing and using the tool chain

Please checkout a new sandbox.

## 3.1 Installing the tool chain

File list:

|  |
| --- |
| **Copy following files to <Sandbox>\04\_Engineering\02\_Development\_Tools\..** |
| scons\_tools\scons\_common\_scripts\modtests\SConscript\_unittests.py |
| scons\_tools\scons\_common\_scripts\modtests\SConscript.py |
| scons\_tools\scons\scons-local\SCons\Tool\g++.py |
| **Copy following files to <Sandbox>\05\_Test\_Environment\algo\modtests\cantata\_tests\xxx where xxx = <component\_name>** |
| scons\_tools\scons\_templates\05\_Testing\05\_Test\_Environment\algo\modtests\cantata\_tests\xxx\CantataWorkspacePreferencesGcc.epf |
| **Merge following files to <Sandbox>\05\_Test\_Environment\algo\modtests\cantata\_tests\xxx** |
| scons\_tools\scons\_templates\05\_Testing\05\_Test\_Environment\algo\modtests\cantata\_tests\xxx\unit\_test.scfg |

files based on CP AL\_ETK\_SCT\_01.13.00

Minimum scons\_tools checkpoint: AL\_ETK\_SCT\_01.11.00 (1.39)

Using other checkpoints:  
Merge scons\_tools\scons\_common\_scripts\modtests\\*.py to the 04\_Engineering\02\_Development\_Tools equivalents.

**Enabling the scripts:**  
Look into <Sandbox>\04\_Engineering\03\_Workspace\algo\<component\_name>\sconscript\_setup\_config.scfg

{

"name" : "modtest",

"copy" : True,

"dest\_folder" : "05\_Testing/05\_Test\_Environment/algo/modtests/cantata\_tests/" + component\_name,

"source" : ["modtests/SConscript\_unittests.py",

"modtests/SConscript.py"]

}

If copy = False:  
 Merge the new python scripts with those in <Sandbox>/05\_Testing/05\_Test\_Environment/algo/modtests/cantata\_tests/ + <component\_name>

Sandboxes with copy = False: VCL, PFC

## 3.2 Enabling gcc/g++ together with Cantata

1. In unit\_test.scfg, set use\_msvc = 0.
2. run ‘scons xxx\_ide\_unittests’

The following sections 3.2.1 and 3.2.2 descibe how to create a new test scripts and how to deal with existing test scripts.

### 3.2.1 Create new test scripts

1. As usual, insert target (called *‘utprg’* now) name into array ‘utprograms’ in unit\_test.scfg.
2. Call the scons build once to generate the ipg.cop file: ‘scons utprg\_ut\_debug’. You can do this from command line or by starting the project build from inside Cantata GUI.
3. The ipg.cop for usage with gcc/g++ is generated.
4. If necessary update the coverage settings in ipg.cop file and rebuild running ‘scons utprg\_ut\_debug’ You can do this from command line or by starting the project build from inside Cantata GUI.
5. Ensure the utprg.csi is generated and File-Under-Test copiles.
6. Generate the test script in Cantata GUI
7. Implement the module tests
8. Execute the module test and generate the HTML report by running ‘scons utprg\_ut\_debug’.

### 3.2.2 Deal with existing Cantata Test Scripts

The following description assumes, the IDE is generated, the test script and ipg.cop file are existing.

1. Update the compiler setting in your ipg.cop file: "--comp:x86-Win32-gcc4.6.2-bundled" for C files, "--comp:x86-Win32-gpp4.6.2-bundled" for C++ files
2. Build the test project running ‘scons utprg\_ut\_debug’. You can do this from command line or by starting the project build from inside Cantata GUI.
3. Ensure the the FUT and the test script compiles. Perhaps modifications are necessary.
4. Execute the module test and generate the HTML report by running ‘scons utprg\_ut\_debug’.

# 4 Use the ‘gdb’ debugger

Following steps are necessary:

## 4.1 Enable debugging Cantata instrumented source files

Insert option “—keepmod” into your ipg.cop file.  
This will keep the ci.sm.c file on your file sytem. This are Cantata-instrumented source file, you can examine within you source editor and step into within the debugger. They are stored in following directory <Sandbox>\04\_Engineering\03\_Workspace\algo\<component\_name>

## 4.2 Enable debug info generation in compiler settings

Insert “-g” option to your compiler flags and disable optimization:

if use\_msvc == 1:

algo\_ut\_ccflags = variant[1]

algo\_ut\_cppdefines = variant[2]

algo\_ut\_linkflags = variant[3]

else:

algo\_ut\_ccflags = **"-O0 -g3 -Wall "**

algo\_ut\_cppdefines = variant[2]

algo\_ut\_linkflags = ""

Need modification in Sconscript.py.

Insert option -fmessage-length=0 into compiler command line:  
ut\_env['CXXCOM']="ipg\_comp --optfile " + Dir('#').abspath + "\\ipg.cop " + '--comp $CXX -o ${TARGET.path} -c -**fmessage-length=0** $SOURCES $CXXFLAGS $CCFLAGS $\_CCCOMCOM $\_CXXCOMCOM'

ut\_env['CCCOM']="ipg\_comp --optfile " + Dir('#').abspath + "\\ipg.cop " + '--comp $CC -o ${TARGET.path} -c **-fmessage-length=0** $SOURCES $CCFLAGS $\_CCCOMCOM'

Modified in sconscript\_unittest.py.

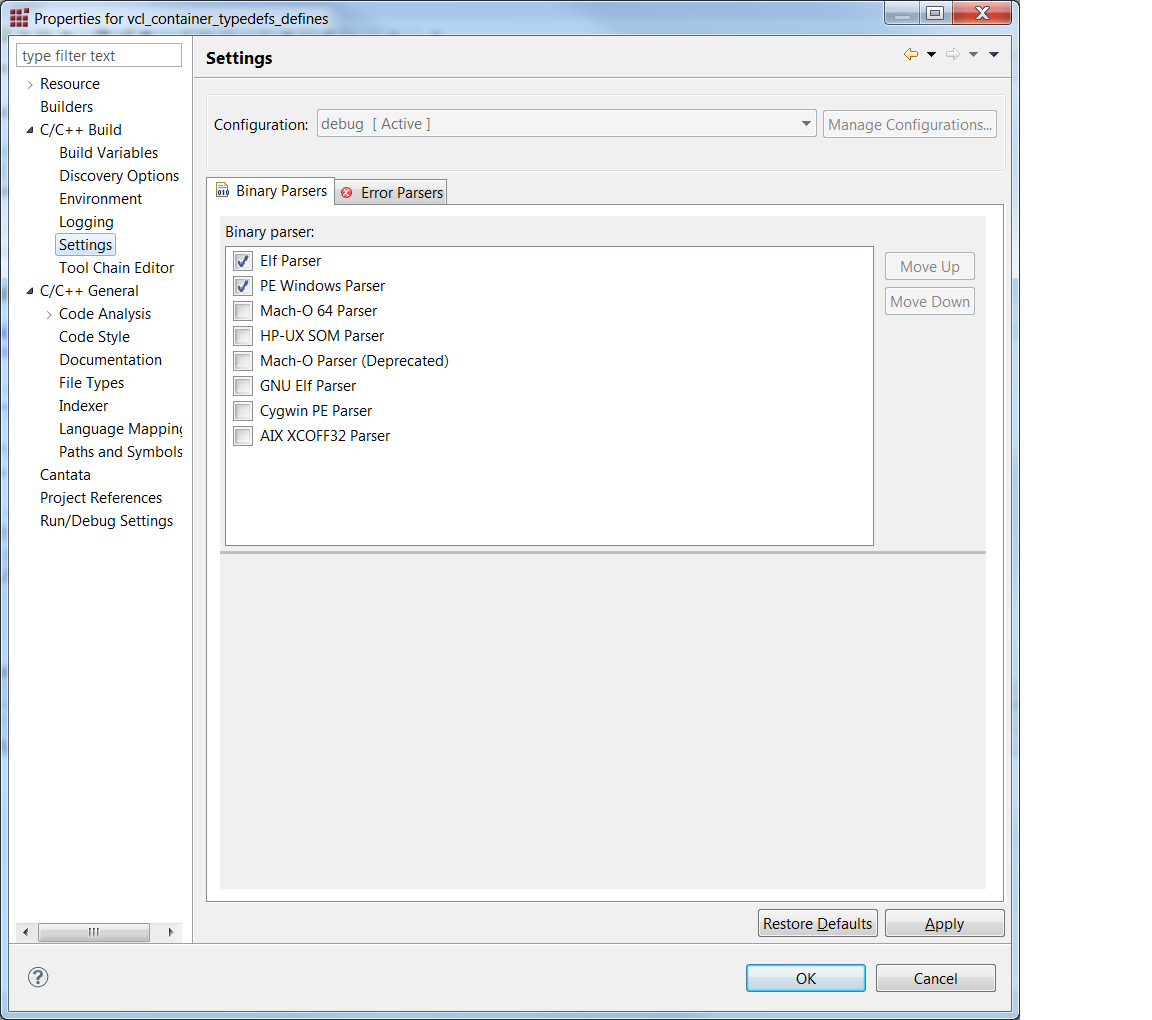
## 4.3 Set additional debugger settings in Cantata frontend

The following instructions describe how to setup the ‘gdb’ debugger in Cantata. All settings are done manually and needs to be redone in each new sandbox.

### 4.3.1 Setup debugger

After selecting your workspace, the first step will be setting up your single project file. First set the active configuration to “debug”. In Project Explorer tab, right click on your single project and select from context menu: Build Configurations → Set Active → debug.

Setup Binary Parser: Open the project properties window and navigate to “C/C++ Build → Settings → Binary Parsers” and select “Elf Parser” and “PE Windows Parser”. And save your changes by clicking button “Apply”.



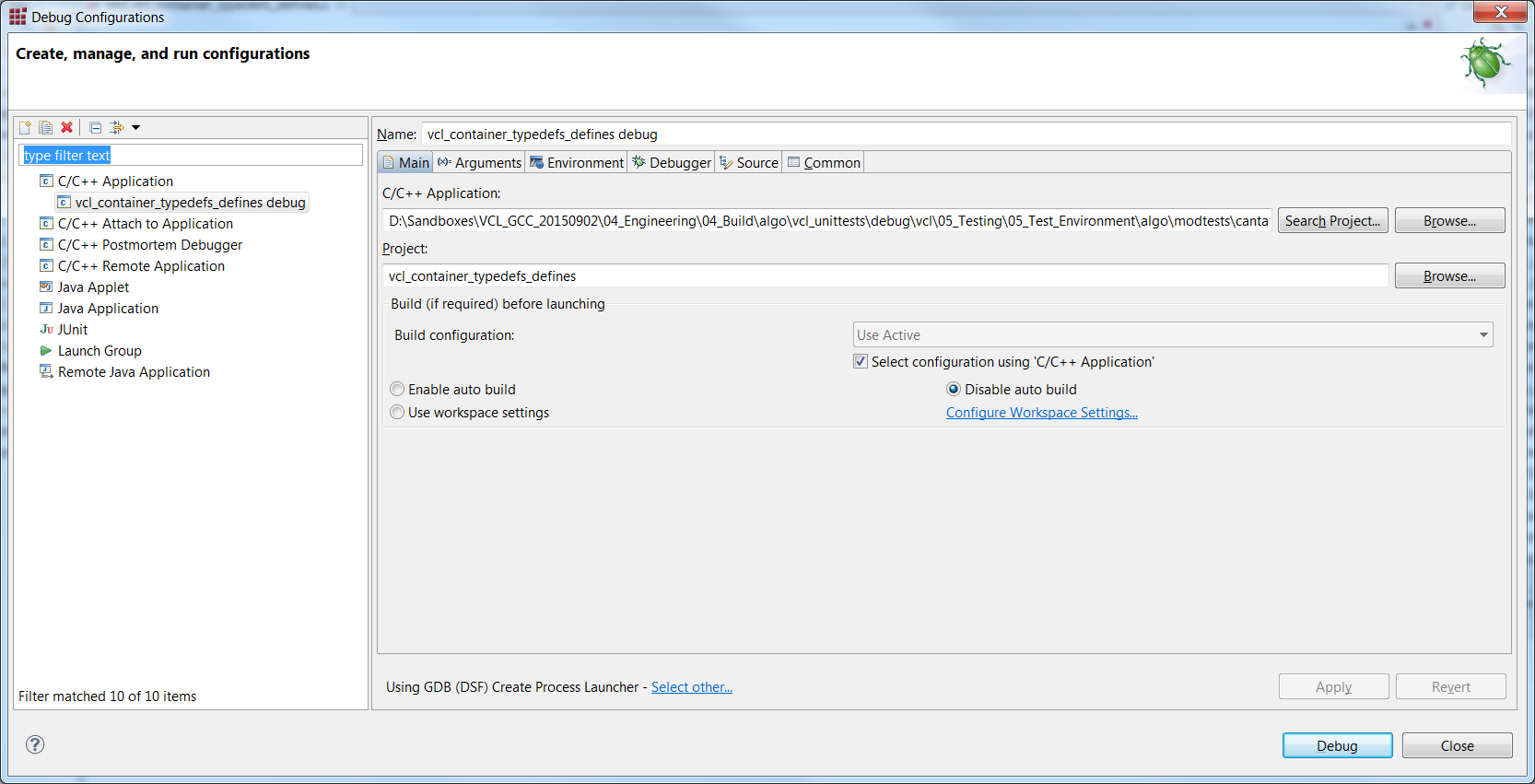
### 4.3.2 Configure project paths in debugger preference

Ensure “debug” is your active configuration. From your Project Explorer Window, right click on your test project and select “Debug As → Debug Configurations” from context menu. The dialogue “Debug Configurations” appears.

#### 4.3.2.1 Set executable

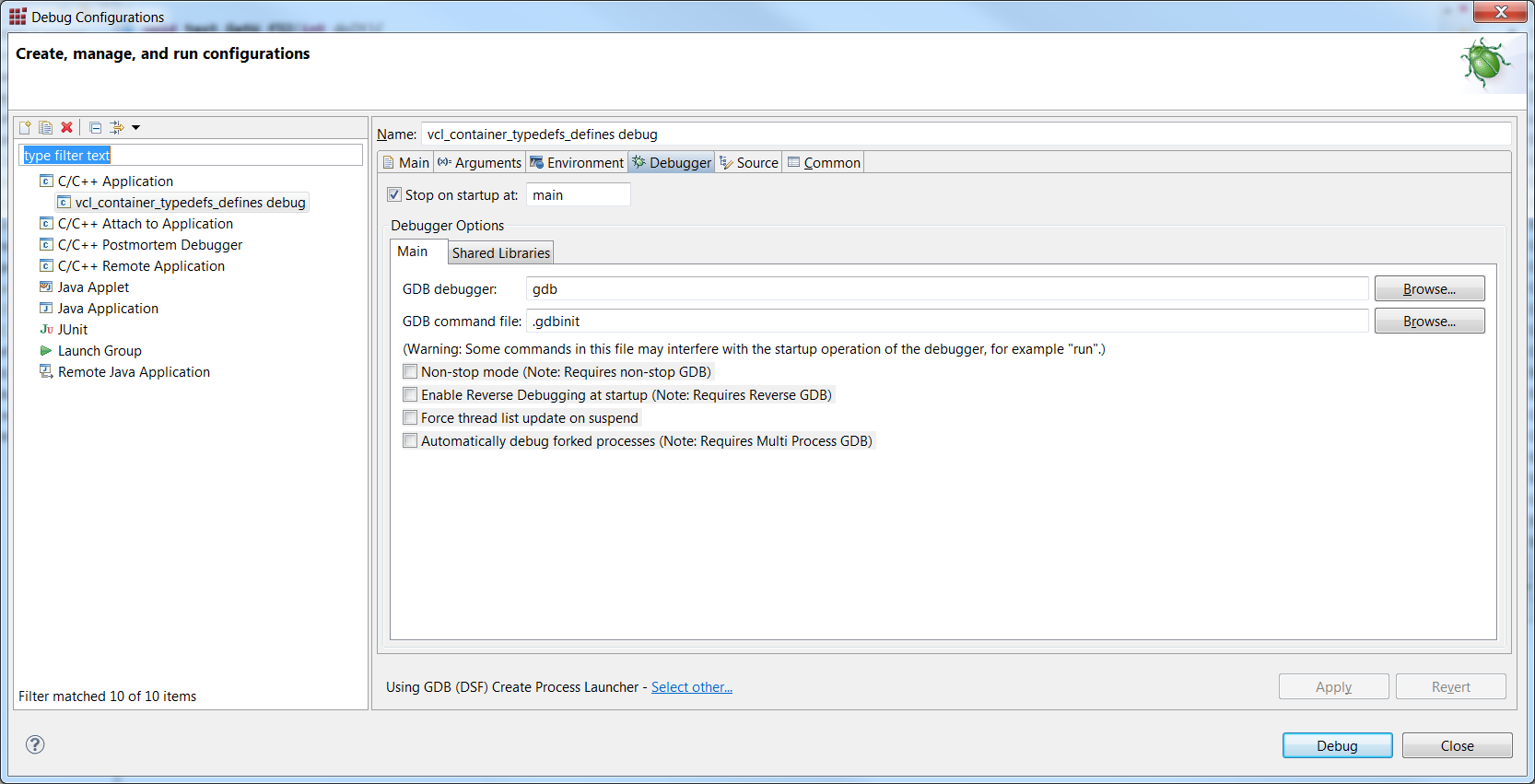
Add C/C++ Application. Keep the name: <ut\_program> debug.

In “Main” tab, select the path to your executable: <Sandbox>\ 04\_Engineering\04\_Build\algo\<component\_name>\_unittests\debug\<component\_name>\ 05\_Testing\05\_Test\_Environment\algo\modtests\cantata\_tests\<component\_name>\<ut\_program>\<ut\_program\_without\_path>.exe  
Disable auto build.



#### 4.3.2.2 Choose ‘gdb’ as debugger.

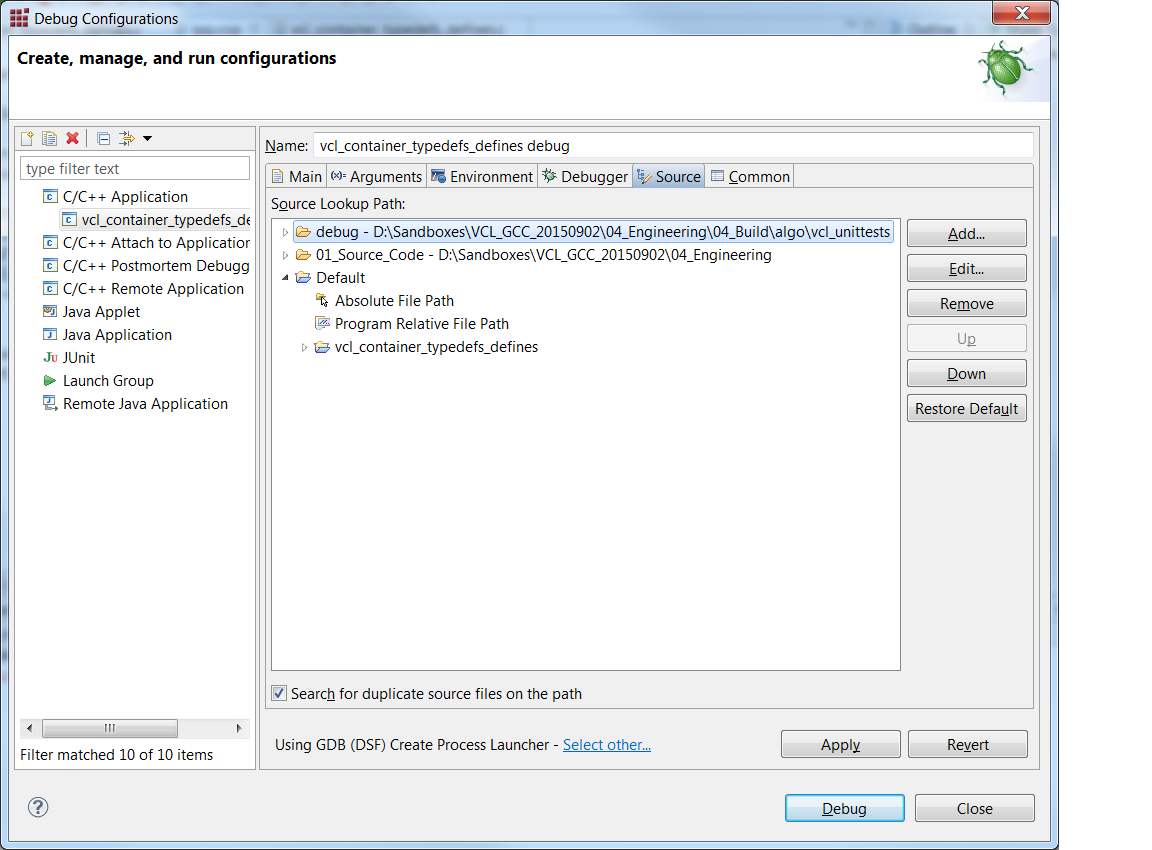
In the “Debug Configurations” window, navigate to tab “Debugger”.



Edit box “GDB debugger” shows only “gdb”. The default setting assumes gdb is in your PATH environment variable and takes the first occurance. If you want to use another gdb installation, please insert the absolute path.

#### 4.3.2.3 Add source directories to debugger

The debugger needs following two additional directories: path to source files, path to object files. In the “Debug Configurations” window, open tab “Source”.  
Add the source folders: Click button “Add”, choose folder <Sandbox>\04\_Engineering\01\_Source\_Code and tick option “Search subfolders”.  
Add the binaries into search path: Click button “Add”, choose folder <Sandbox>\04\_Engineering\04\_Build\algo\<component\_name>\_unittests\debug and tick option “Search subfolders”.  
Store your change by clicking button “Apply”.



# 5 Possible problems

1. FUT or existing test script throw compiler error. They need to be fixed first.  
   Known problems: CMatrix class of CML library
2. Microsoft-specific data types in components’ PC\_SIM build. Types \_\_int64, \_\_int32, \_\_int16, \_\_int8 are translated by a #define in SConscript.py, see line 16fff  
   But there might be a more elegant solution, for example inserting a header file into 05\_Testing\05\_Test\_Environment\algo\modtests\cantata\_tests\xxx\Framework.
3. Linking foreign libraries into the module tests. This is done in PFC component. It links against simulatin libraries placed in 04\_Engineering\\03\_Workspace\\algo\\externals\\sim\\lib.  
   This also results in a customized scons script, see 05\_Testing\05\_Test\_Environment\algo\modtests\cantata\_tests\pfc\Sconscript.py, line 87.  
   A possible solution is including the relevant .cpp files directly into the test script and setting up the paths for that (perhaps copying or sharing the files into Framwork directory)
4. Customizations of the unit test scons scripts, see chapter 3.1

# 6 Future SCons features regarding gcc usage

1. Automatically update “—comp” section in existing ipg.cop files
2. Enabling custom compiler settings, linker settings in unit\_test.scfg.  
   This can be done similar to file 03\_Workspace\algo\xxx\_sim\sim\_swc\_xxx\sim\_env\_config.scfg, ‘variable variant\_list’.

# 7 Additional info: Use other Visual Studio version than VS 2005 (8.0)

Add a Visual Studio version parameter to unit\_test.scfg: *msvc\_version = “10.0”*

Change file modtests\Sconscript.py:

ut\_env = Environment(# list of scons extensions used

tools = ["msvc", "msvs-patched", "mslib", "mslink",

"msvc-addon", "doxygen", "fingerprint",

"eclipse\_cdt", "unittest"],

# path to scons extensions

toolpath = [scons\_adas\_extensions\_path],

MSVS\_USE\_MFC\_DIRS = 1,

TARGET\_ARCH = "X86",

# path to doxygen tool

DOXYGEN\_TOOLS = doxygen\_dir,

# manifest files need to be included in the dlls/apps

WINDOWS\_EMBED\_MANIFEST = True,

# use Visual Studio 2005

MSVC\_VERSION = **msvc\_version**,

# specify processor for scons build

BUILD\_TARGET = "SIM")

Create a CantataWorkspacePreferences.epf file for MSVC 10: x86-Win32-5.x-msvc10

Update the ipg.cop file accordingly.