metrowerks

Application Note
Using CodeTEST Native with Rhapsody by I-Logix

Revision 0.2

Revision History

Revision	Description	Date
0.1	First Draft.	01/23/2001
0.2	CodeTEST v3.0 and Windows 2000 snapshots added	10/29/01

1. About This Document

1.1 Introduction

The CodeTEST software verification tool from Applied Microsystems, and the Rhapsody UML visual programming environment from I-Logix form a natural partnership as a solution for the development of real-time safety critical embedded systems.

Rhapsody is a visual UML development environment supporting object-oriented development, from analysis, through design, implementation, and test. Rhapsody's production quality code generation capabilities allow developers to quickly target their application to virtually any architecture and operating system. With Rhapsody, the model and code always stay in sync, and automatically generated documents naturally match the actual implementation.

CodeTEST is a software verification tool designed specifically for embedded systems software developers and testers. Using a combination of code instrumentation and data collection technologies, it can be used to analyze software Performance, code Coverage and Dynamic Memory allocations, and also to obtain real time software execution trace data.

This document describes how these two products may be configured to work together so that CodeTEST instrumentation may be automatically included in the code generated by Rhapsody. Using a sample application that comes with Rhapsody, a worked example will be followed through for code generated by Rhapsody for a Windows NT host, and then instrumented for use with the CodeTEST Native analysis environment.

CodeTEST Native uses a software data collection method to enable code analysis to be performed on the same Windows NT host environment on which Rhapsody is running. Although not specifically addressed in this application note, Rhapsody can also be configured to work with CodeTEST's Software-In-Circuit (SWIC) and Hardware-In-Circuit (HWIC) data sources.

Rhapsody in C and Rhapsody in C++ can be configured to work with CodeTEST.

1.2 Scope

This application note addresses both the Windows NT and Windows 2000 host environments. Screen snapshots are included for both CodeTEST 2.2 and 3.0. The following environment has been used:

1) CodeTEST version 2.2 and version 3.0:
    a. Windows NT and 2000 Native
2) Rhapsody in C++ version 3.0.1 (build 9533)
3) Microsoft Visual Studio version 6.0
4) Windows NT and 2000

Whilst every effort has been made to capture all of the configuration changes that are required to make these two products work together, there is no guarantee that the examples described here will work with different versions of either tool, with different compilers or in a different host environment.

NOTE: This application note assumes that CodeTEST, Rhapsody and the build environment are already installed and configured.

1.3 Definitions and Acronyms

1) In the interests of simplifying the text, throughout this document the path to the CodeTEST installation directory has been described using the AMC_HOME reference. For example, if CodeTEST has been installed in the following directory:

G:\tools\amc\CodeTEST

The AMC_HOME reference is set to the following:

AMC_HOME = G:\tools\amc\CodeTEST

Whenever the AMC_HOME reference appears, it is used to represent the whole installation path. This means that if the following appears:

'change directory to AMC_HOME\bin'

Then what is actually meant is:

'change directory to G:\tools\amc\CodeTEST\bin'

2) In the same manner as in 1) above, the path to the Rhapsody installation directory has been described using the OMROOT reference. For example, if Rhapsody has been installed in directory:

G:\tools\ilogix\Rhapsody

The OMROOT reference is set to the following:

OMROOT = G:\tools\amc\ Rhapsody

Whenever the OMROOT reference appears, it is used to represent the whole installation path.

3) To denote text that appears in a text file, or to represent code samples, the following font will be used:

```
this represents an entry in a text file
```

1.4 Related Documents

- CodeTEST user's guide
- CodeTEST instrumentation manual
- Getting Started Guide with CodeTEST Native
- Rhapsody user's guide
- Rhapsody RTOS Adapter Guide
- Rhapsody on-line help

1.5 Author

Any comments about this document should be directed to:

Nat Hillary

Metrowerks

Voice: (425) 487-5985
email: nath@metrowerks.com

2. Process overview

Because Rhapsody generates its own makefiles at the same time as generating code, the process for getting CodeTEST and Rhapsody to work together is straightforward, and involves three main steps:

- The Rhapsody property settings must be updated to generate makefiles that call the CodeTEST compiler driver rather than your usual compiler driver.
- The Rhapsody code generation configuration must be updated to include the appropriate CodeTEST run time library and also the new code generation environment property.
- Set the AMC_TARGET environment variable appropriately to tell CodeTEST exactly which build environment you are using.

This application note uses the Dishwasher sample code provided with Rhapsody as an example to walk through this process in detail. This Application Note addresses the CodeTEST Native data collection method only, using the Microsoft Visual Studio compilation environment.

3. Using code generated by Rhapsody with CodeTEST Native

CodeTEST Native provides the ability to perform Code Coverage Analysis, Memory Analysis and Software Execution Trace on a host platform. This example looks at how the code generated from Rhapsody may be automatically instrumented for analysis by CodeTEST Native.

In this example, the Rhapsody siteC++.prp file is updated to enable code to be generated with a makefile that calls the CodeTEST compiler driver (which in turn calls the CodeTEST instrumenter), rather than the normal Microsoft compiler driver.

Next, the Rhapsody project is updated to include the CodeTEST Native run time libraries. These libraries contain the implementations of a number of functions that are placed into the code, by the instrumenter, during the build process.

Before compiling the code, either from within Rhapsody or from a DOS prompt, the AMC_TARGET environment variable must be set to tell the CodeTEST compiler driver what build environment is being used.

Finally, after successful instrumentation, CodeTEST may then be used to analyze your executing code. Note that this analysis may occur whether you are using Rhapsody animation, or not.

3.1 Modifying the siteC++.prp file

Rhapsody has a number of tool and project properties that affect aspects of a model, ranging from the appearance of graphics in the diagram editors to code generation and many other subjects. The Rhapsody tools, such as the code generator, reference these properties heavily.

Default properties are assigned in the factory and site default files, factory.prp and site.prp, respectively. In addition, there are language specific default files, factoryC++.prp and siteC++.prp, that define properties specifically for Rhapsody in C++. These files are located in the $OMROOT\share\Properties directory and provide a way to tune project properties on an individual or site-wide basis without recompiling Rhapsody.

This application note describes a modification to the siteC++.prp file to update the default Microsoft build environment already defined in the factoryC++.prp file. New properties are added to the siteC++.prp file to allow the user to control CodeTEST instrumentation. With CodeTEST instrumentation enabled, the Rhapsody generated makefile overrides the standard compiler driver and replaces it with the CodeTEST compiler driver. With this change, the CodeTEST instrumentation process occurs as part of the normal compilation process.

3.1.1 Updating the "Microsoft" code generation environment

1) In the OMROOT\share\Properties directory, open the factoryC++.prp file using a text editor.
2) Open the siteC++.prp file. The default file will contain a single 'end' statement, a single line comment, and nothing else.
3) In the siteC++.prp file, create a `Subject CPP_CG` section as follows:

```
Subject CPP_CG

end
```

4) In the factoryC++.prp file, search for the metaclass `Microsoft` within the subject `CPP_CG`, and the property `MakeFileContent` within the metaclass `Microsoft`. The `MakeFileContent` property defines a makefile template that Rhapsody will use when generating code. Copy the entire contents of this property into the sitec++.prp file:

```
Subject CPP_CG

      Metaclass Microsoft
            Property MakeFileContent MultiLine "
            <contents of property>
            "
      end
```

5) Add the `CodeTESTUsage`, `CodeTESTCoverageLevel`, and `CodeTESTCompileSwitches` properties to the metaclass `Microsoft`:

```
Subject CPP_CG

      Metaclass Microsoft
            Property CodeTESTUsage Enum "No,Yes" "No"
            Property CodeTESTCoverageLevel Enum "None,Statement
Coverage,Decision Coverage,MC/DC" "Statement Coverage"
            Property CodeTESTCompileSwitches MultiLine "-CTv -CTkeep -CTtag-
allocator"
            Property MakeFileContent MultiLine "
            <contents of property>
            "
      end
```

6) The next step is to update the `Microsoft` environment's makefile template to call the CodeTEST compiler driver.
7) It is necessary to define the CPP macro within this file to point to the CodeTEST compiler driver. By default, the CPP macro points to the Microsoft compiler driver. In the `Target type (Debug/Release)` section of the makefile template, add the following:

```
!IF \"$(CodeTESTUsage)\" == \"Yes\"
```

```
!IF \"$(CodeTESTCoverageLevel)\" == \"Statement Coverage\"
CPP=ctcxx.exe $(CodeTESTCompileSwitches) -CTtag-level=SC
!ELSEIF \"$(CodeTESTCoverageLevel)\" == \"Decision Coverage\"
CPP=ctcxx.exe $(CodeTESTCompileSwitches) -CTtag-level=DC
!ELSEIF \"$(CodeTESTCoverageLevel)\" == \"MC/DC\"
CPP=ctcxx.exe $(CodeTESTCompileSwitches) -CTtag-level=MCDC
!ELSE
CPP=ctcxx.exe $(CodeTESTCompileSwitches)
!ENDIF
!ENDIF
```

This conditional allows the Rhapsody user to turn CodeTEST instrumentation on or off by setting the `CodeTESTUsage` property. For example, if `CodeTESTUsage` is set to `Yes`, and the `CodeTESTCompileSwitches` property is set to `Statement Coverage`, the following CPP macro is generated in the makefile.

```
CPP = ctcxx -CTv -CTkeep -CTtag-allocator -CTtag-level=SC
```

This line calls the CodeTEST C++ compiler driver (`ctcxx`) with the following switches:

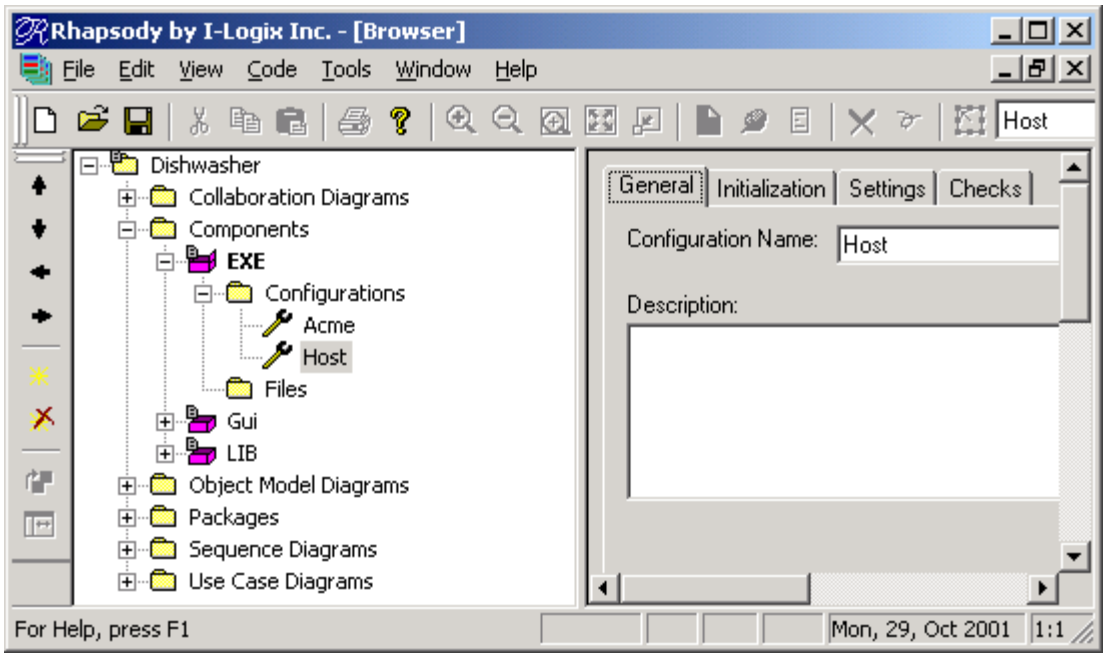
`-CTv`	- verbose mode
`-CTkeep`	- retain intermediate build files
`-CTtag-level=SC`	- instrument for Statement Coverage
`-CTtag-allocator`	- include memory instrumentation

NOTE: The above list has been provided for completeness only. The complete range of instrumentation options that CodeTEST provides may be found in the CodeTEST instrumenter Reference Manual. These additional instrumentation options may be specified using the `CodeTESTCompileSwitches` property, through the Rhapsody property editor or the siteC++.prp file.

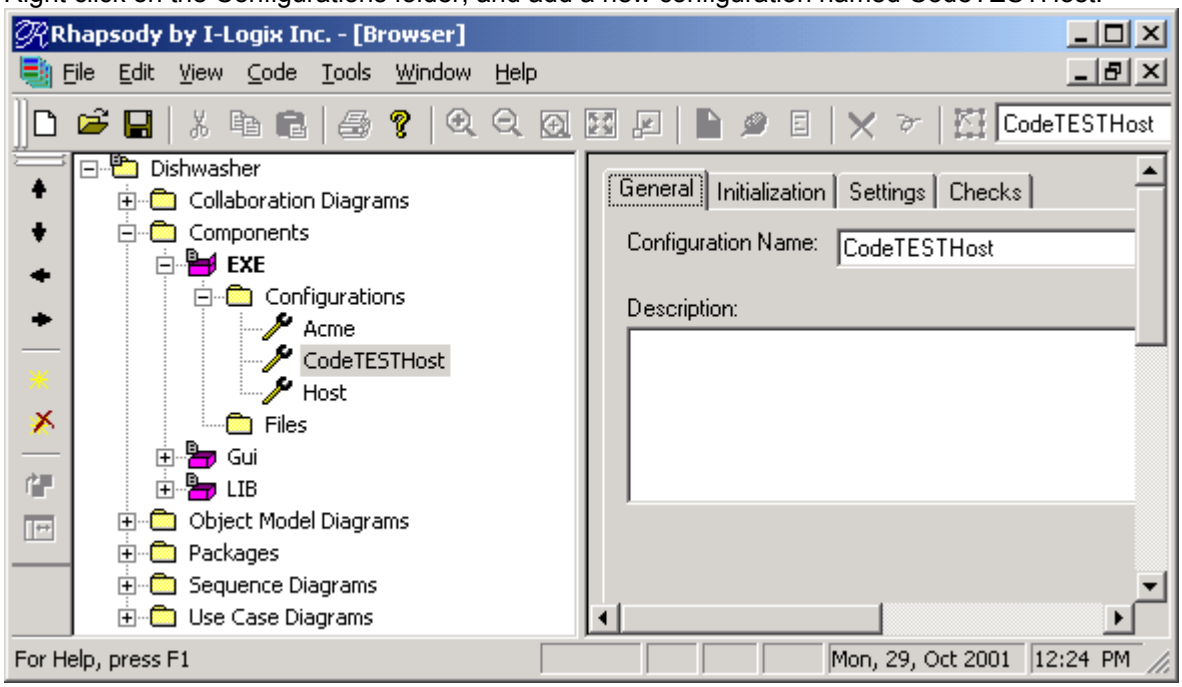
8) Save the contents of the siteC++.prp file before continuing.

3.1.2 Using the updated Microsoft code generation environment with sample Rhapsody project Dishwasher

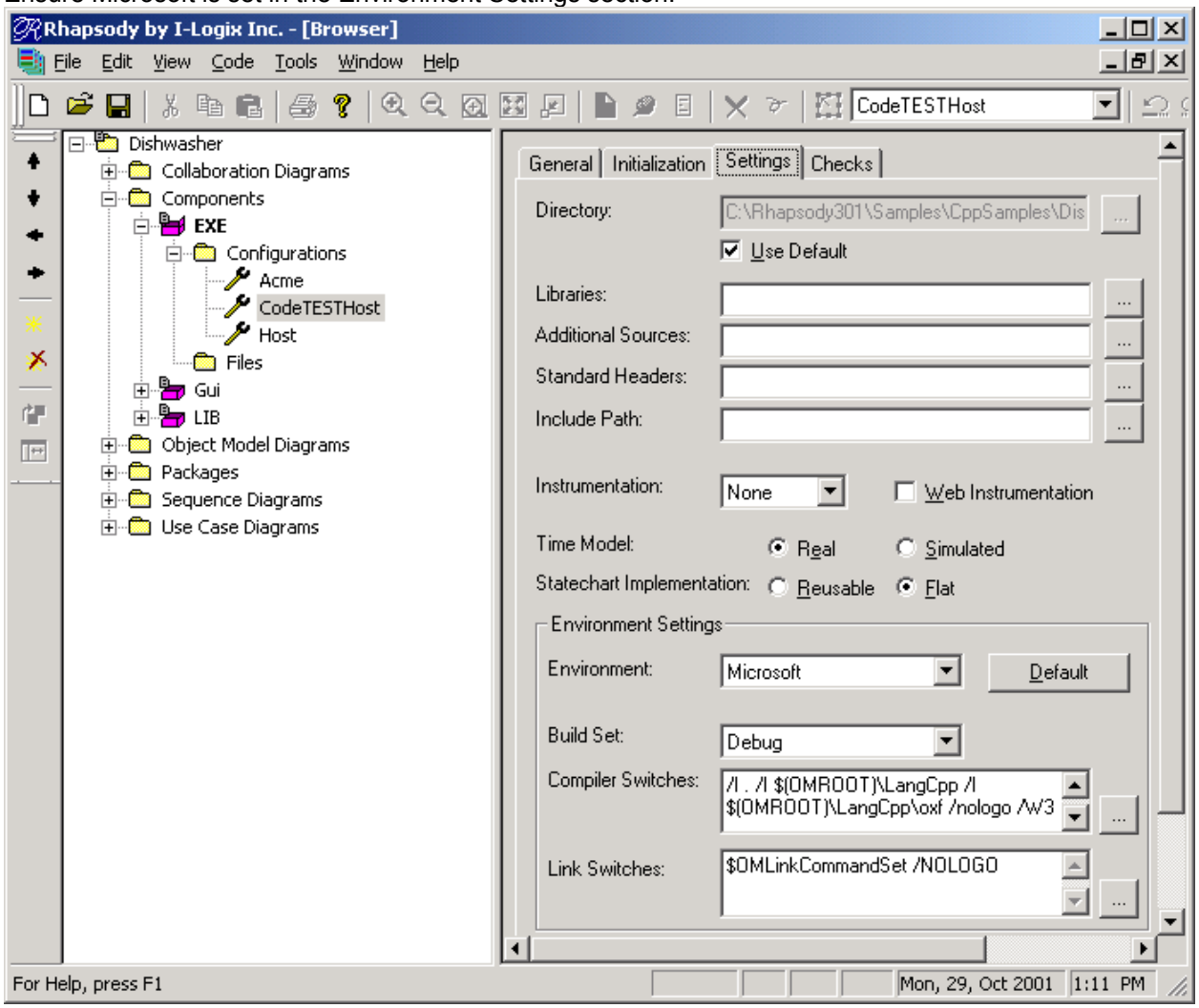
1) Start Rhapsody and open the sample project Dishwasher, found in the OMROOT\samples\CppSamples\Dishwasher directory.
2) From the Rhapsody project browser, select the Components tab and open up the list of configurations as follows:



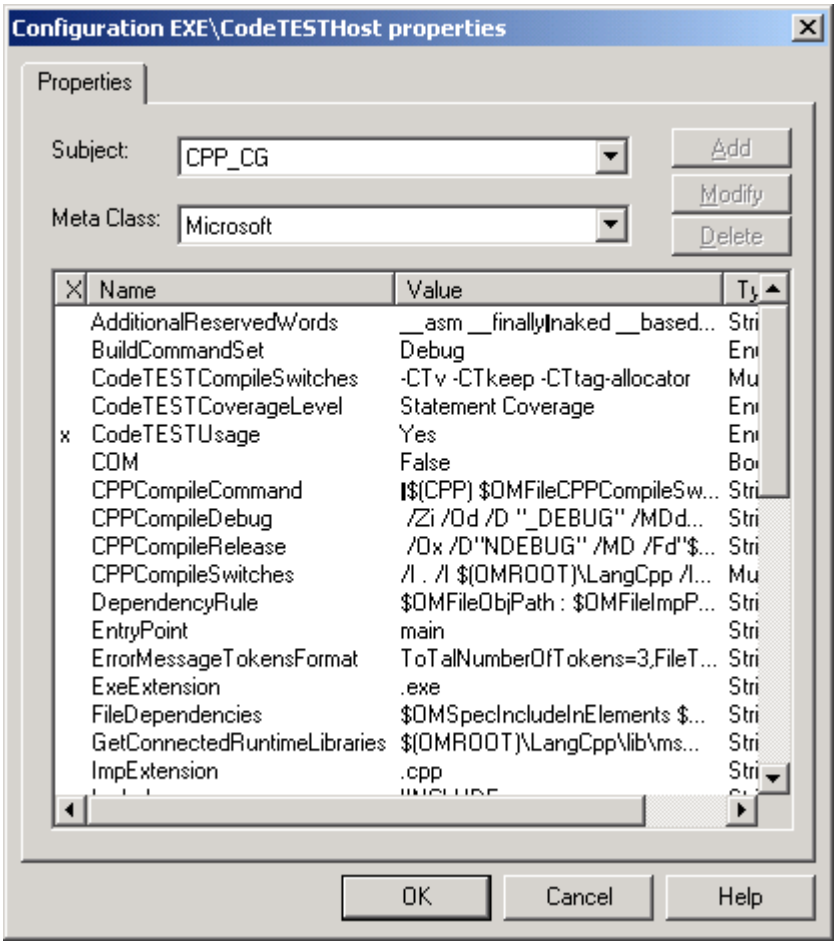
3) Right click on the Configurations folder, and add a new configuration named CodeTESTHost:



4) With the `CodeTESTHost` configuration active, select the Settings tab from the right hand pane. Ensure Microsoft is set in the Environment Settings section:

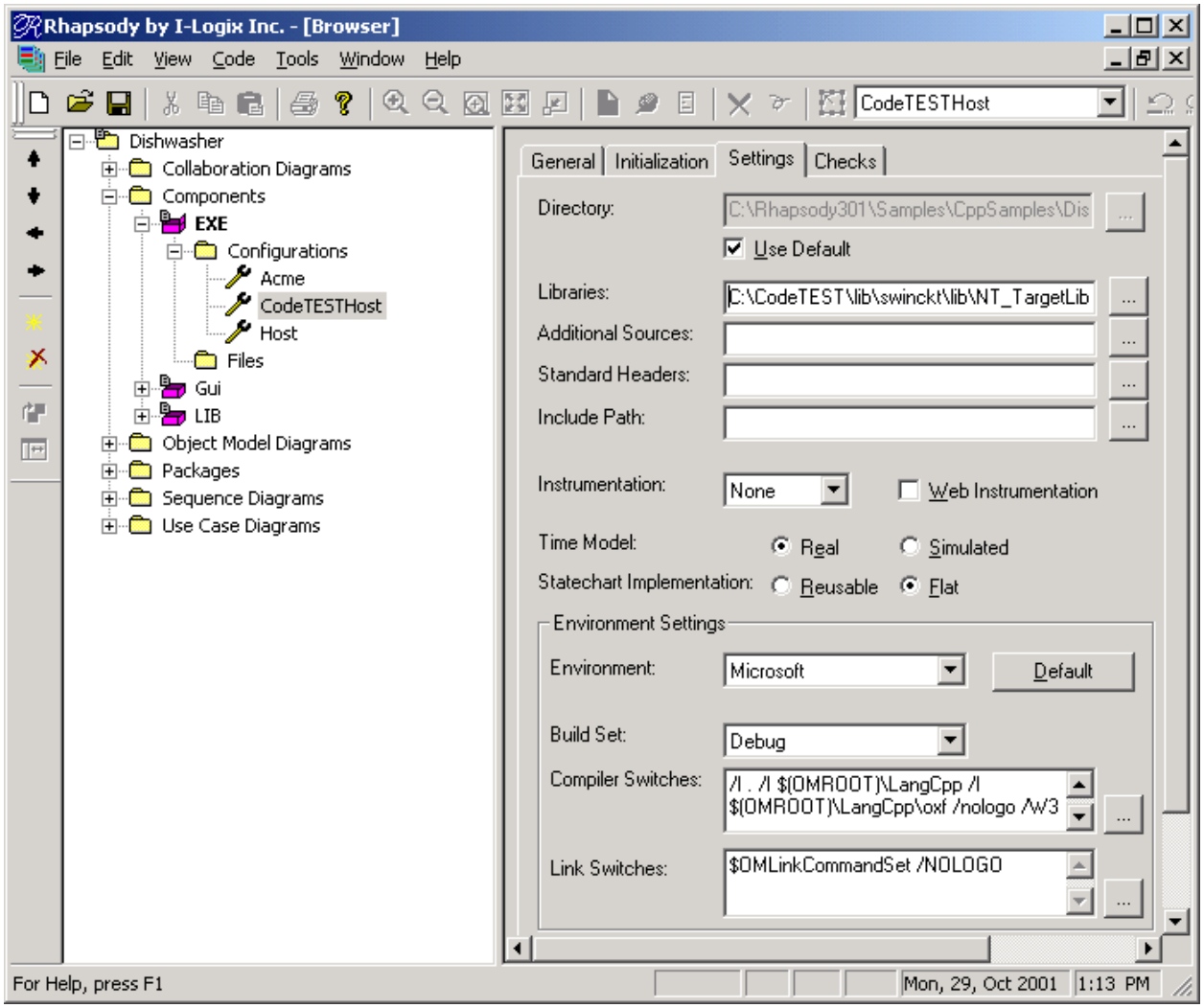


5) Select the `CodeTESTHost` configuration, right click, and select 'Properties' from the resulting popup menu. Set the `CodeTESTHost` configuration's `CPP_CG::Microsoft::CodeTestUsage` property to `Yes` to automatically instrument for CodeTEST analysis.

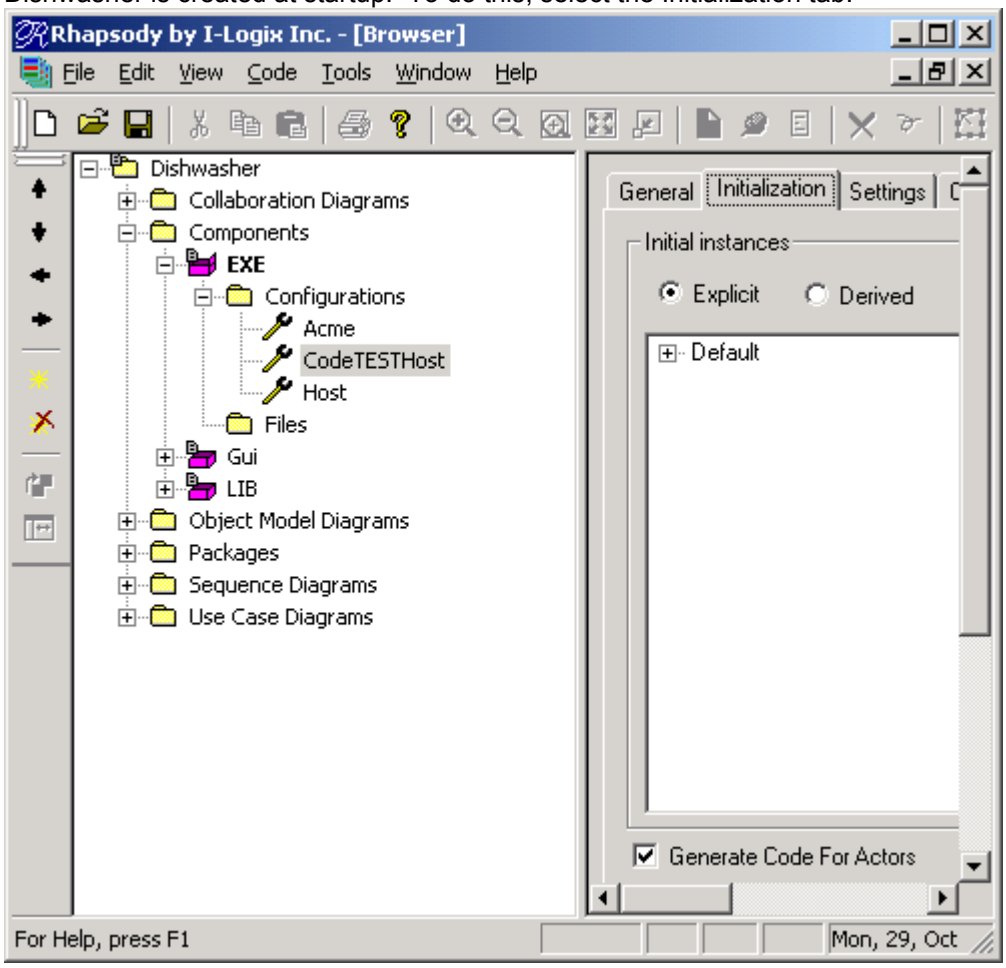


6) Before this code will compile correctly, it is necessary to include the correct CodeTEST Native libraries. In the Libraries text box, enter the following (Note that this is best done by selecting the ellipsis button (…) and making an entry in the resulting editing window):

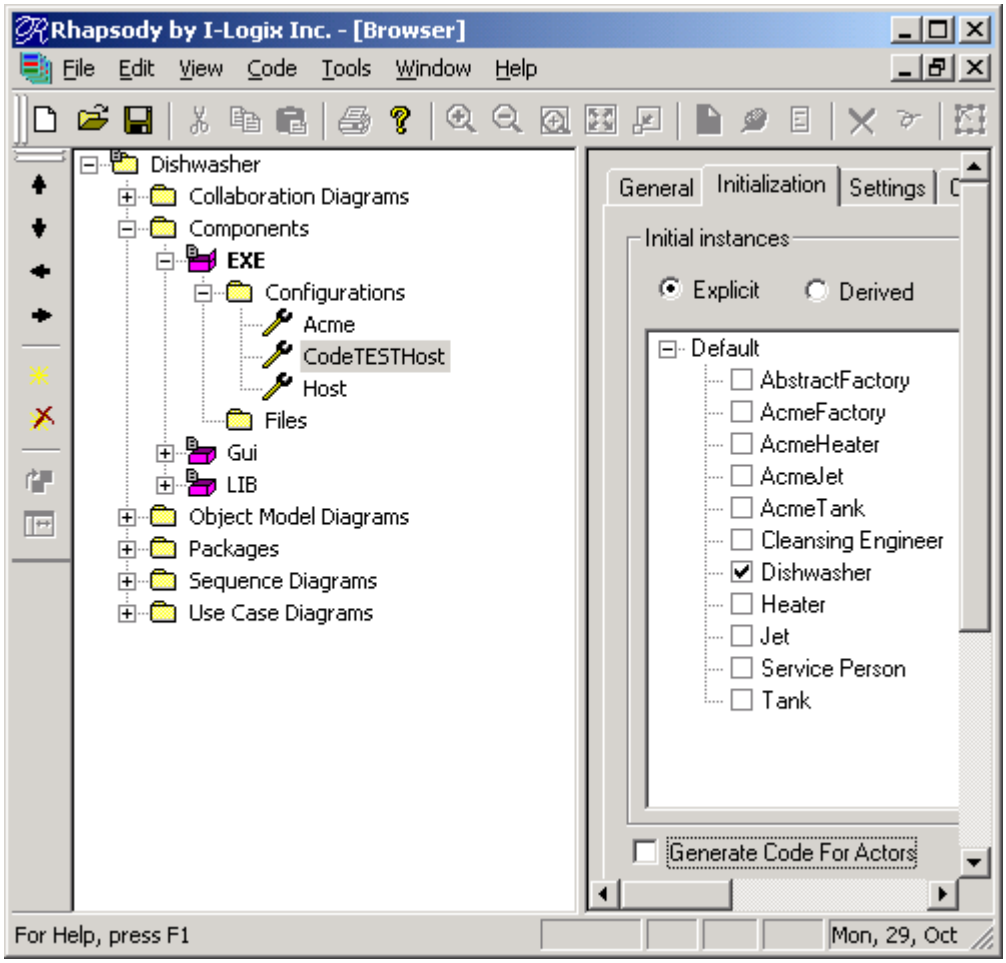
```
C:\CodeTEST\lib\swinckt\lib\NT_TargetLibMD.lib;
C:\CodeTEST\lib\rtos\winnt4\mvc_Release\ct_MemMD.obj
```



7) Before this code generation configuration can be used, it will be necessary to ensure that Rhapsody does not generate code for the actors in the system, and that an object of type Dishwasher is created at startup. To do this, select the Initialization tab:



8) Deselect the 'Generate Code for Actors' box, expand the list of initial instances and select Dishwasher:

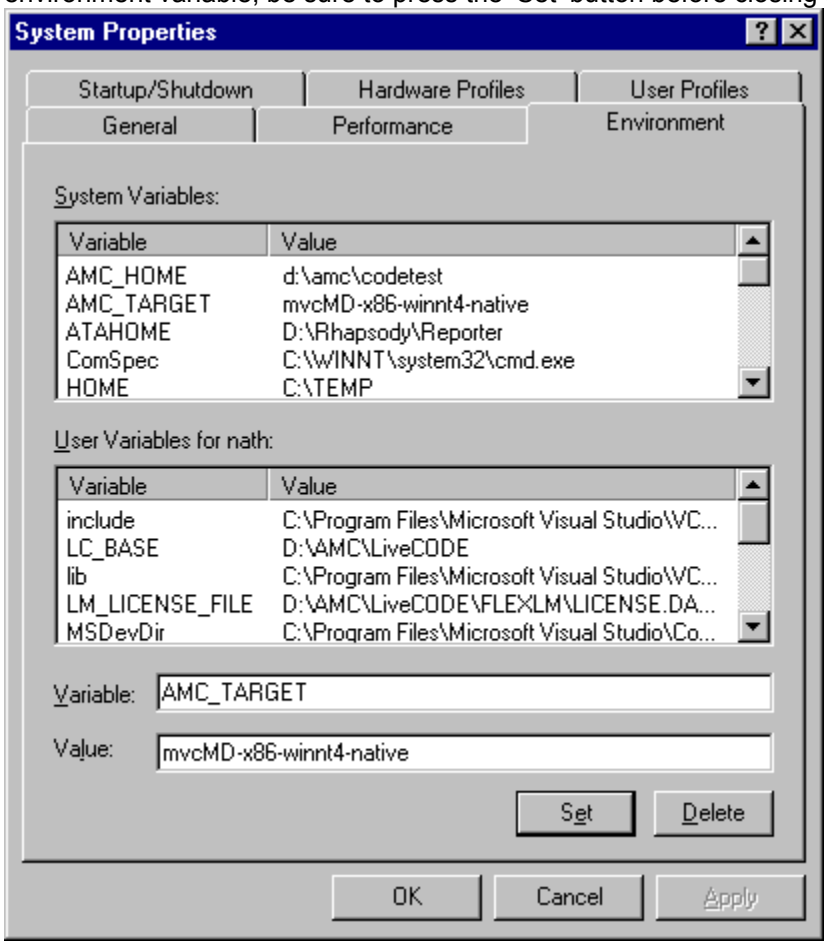


9) Before generating code, it is necessary to tell Rhapsody to use this code generation configuration. Right click on the CodeTESTHost configuration and select the 'Set as Active Configuration' option.

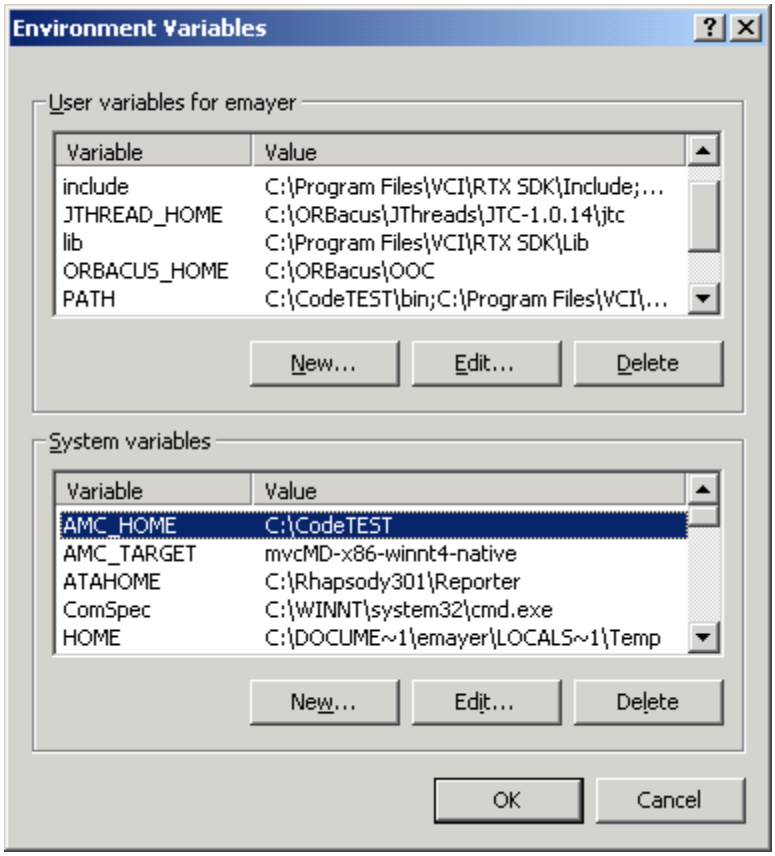
10) Code can now be generated and instrumented as part of the build process. Before code may be instrumented by CodeTEST, however, the AMC_TARGET environment variable must be set for the Microsoft Visual Studio compilation environment. Pull up the system environment editor by right clicking on the 'My Computer' icon on the desktop and selecting the Properties option.

11) Select the 'Environment' tab from the resulting dialog box, and ensure that the AMC_TARGET variable is set to `mvcMD-x86-winnt4-native.` I

The System Properties window for Windows NT follows. If you have to change the value of this environment variable, be sure to press the 'Set' button before closing the window:



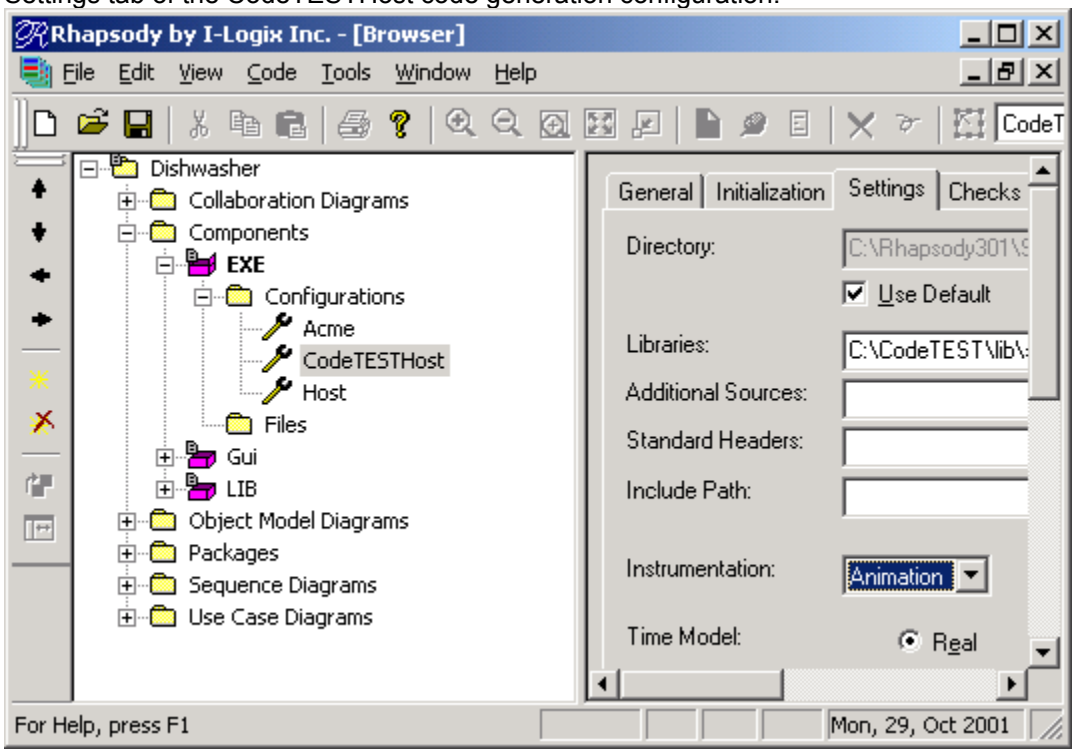
The System Properties window for Windows 2000 follows.



12) In order to generate, compile and run code, select Code->Generate/Make/Run from the Rhapsody menu. In the Rhapsody output window, you will see the code being generated and compiled, and finally the executable will be started as a console application.

NOTE: If the AMC_TARGET environment variable has had to be changed, then Rhapsody must be restarted before this step.

13) If Rhapsody visual debugging is required, select Animation from the Instrumentation list in the Settings tab of the CodeTESTHost code generation configuration:

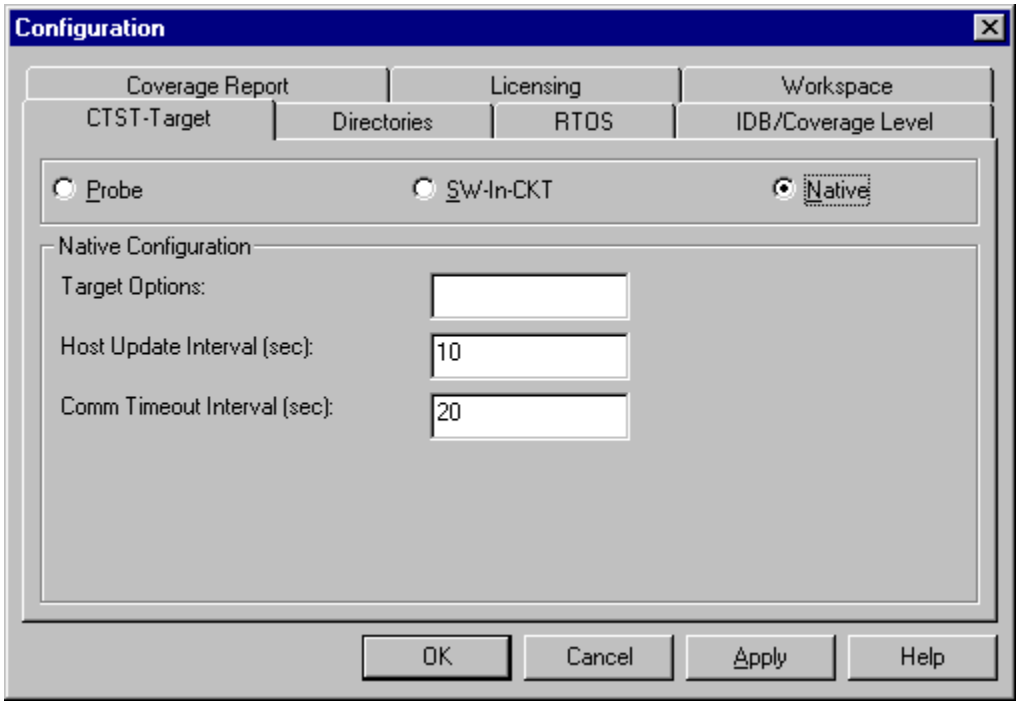


NOTE: If you change the setting of the Instrumentation tab between code generations, the resulting IDB MUST be deleted before code may be instrumented.

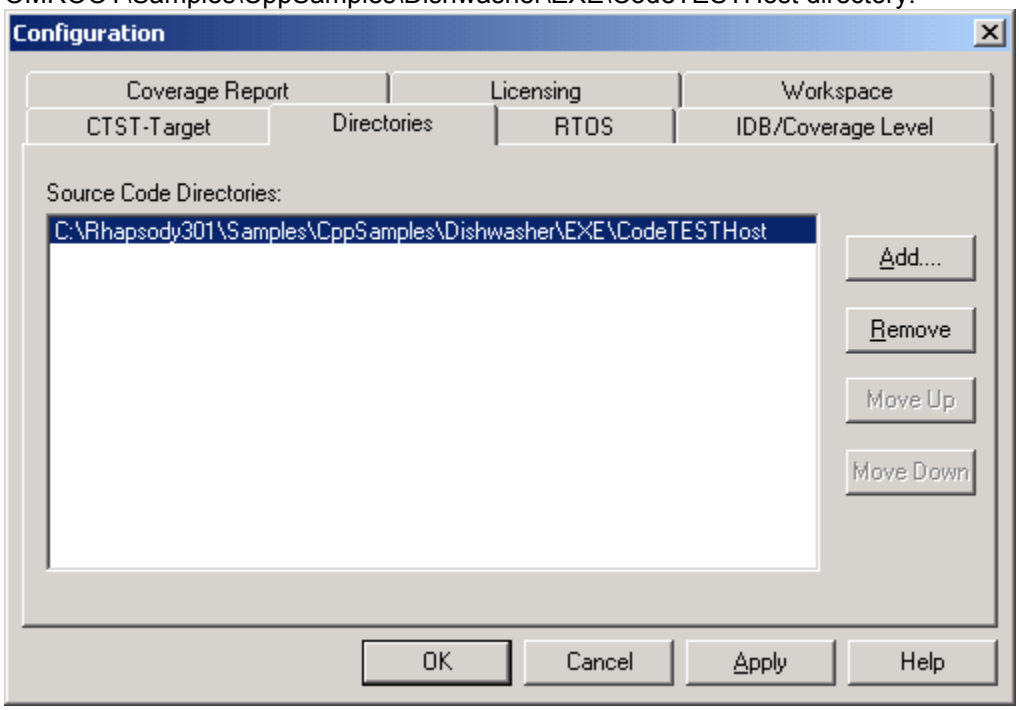
3.1.3 Making CodeTEST v2.2 measurements

1) Start the CodeTEST v2.2 host application
2) From the tools menu, select Configuration.

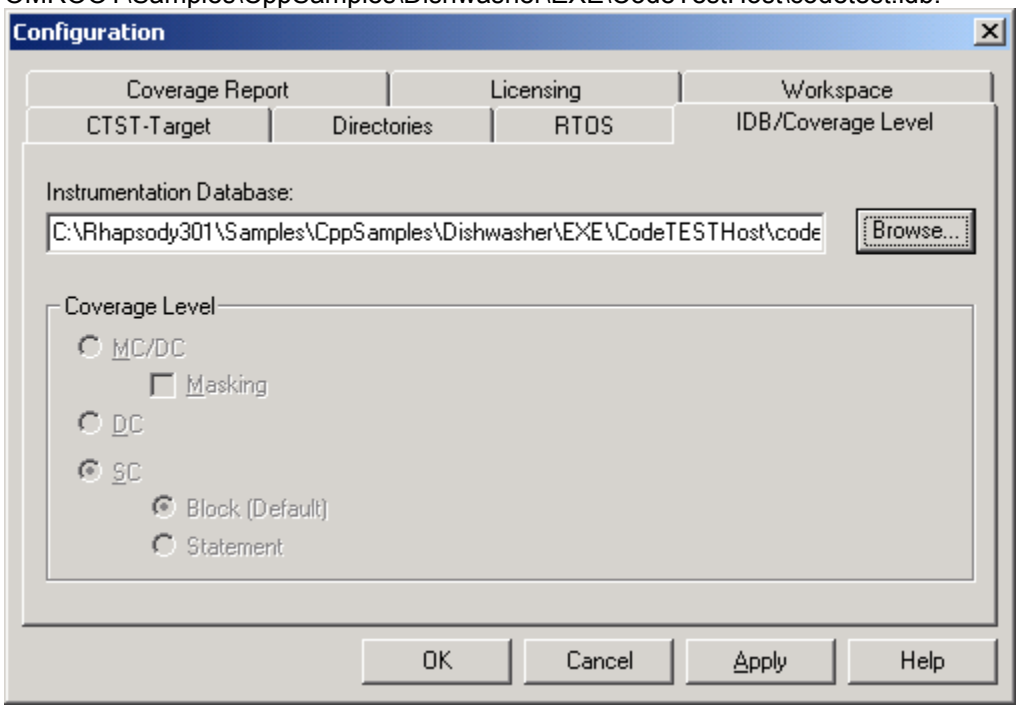
3) With the CTST-Target tab selected, select Native as the data collection type by selecting the Native radio button:



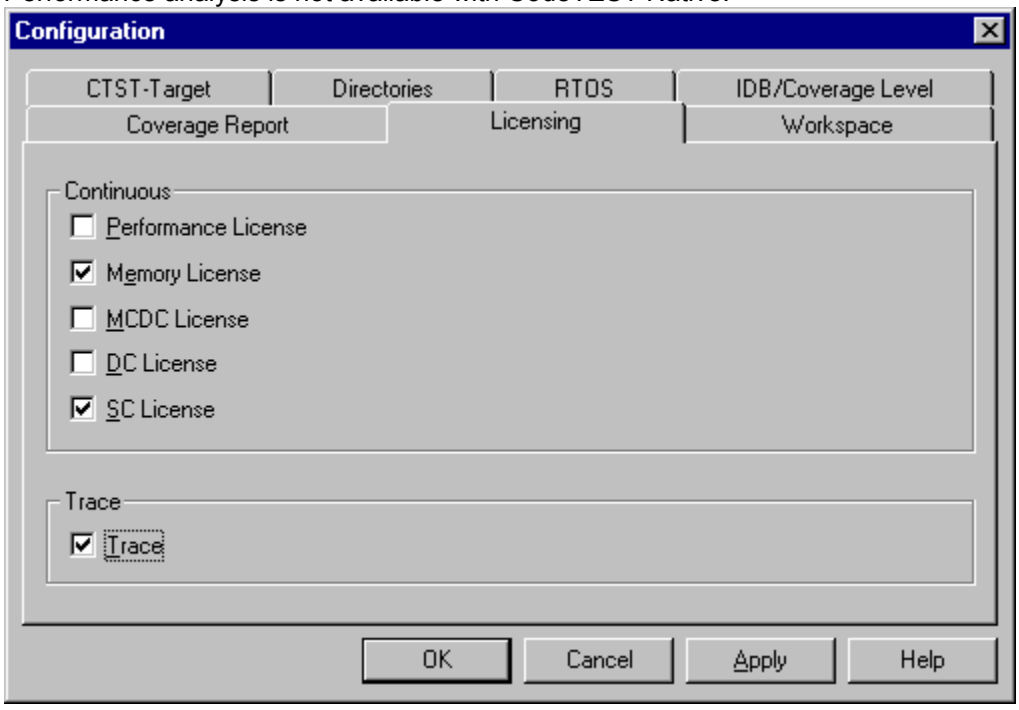
4) With the Directories tab selected, remove the current directories, and add the OMROOT\Samples\CppSamples\Dishwasher\EXE\CodeTESTHost directory:



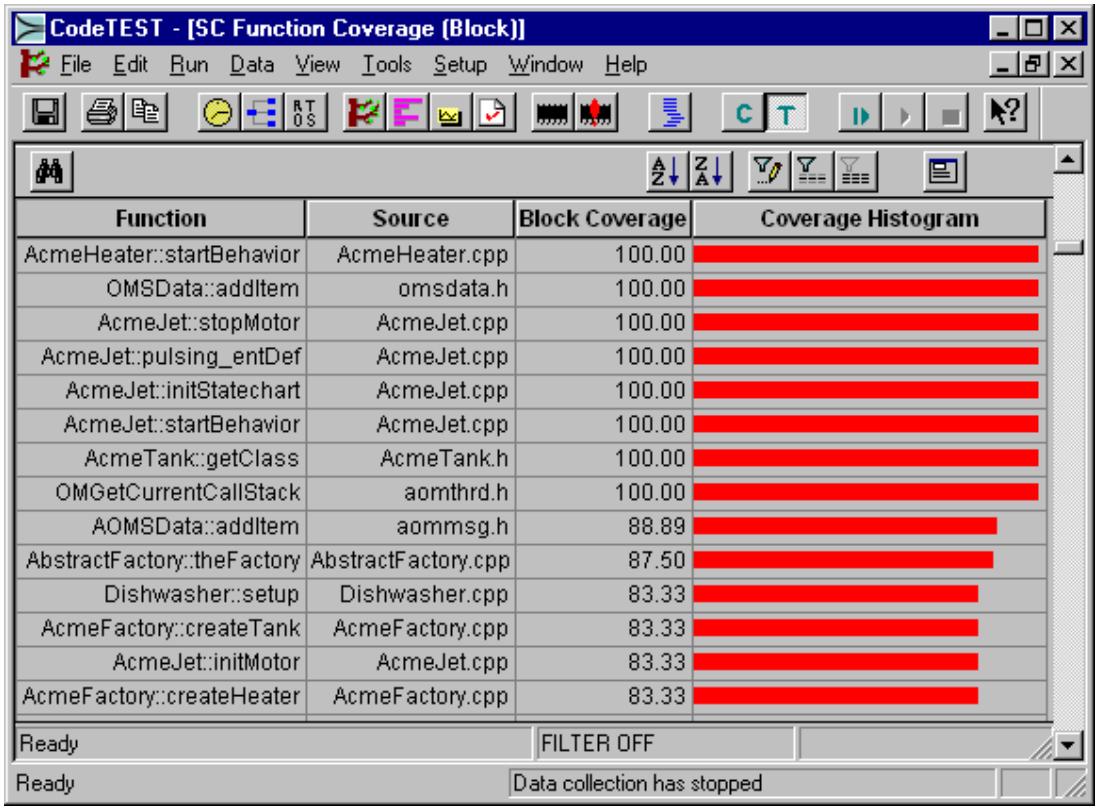
5) With the IDB/Coverage Level tab selected, set the Instrumentation Database to OMROOT\Samples\CppSamples\Dishwasher\EXE\CodeTestHost\codetest.idb:



6) With the licensing tab selected, select those tabs for which you have licenses. Note that Performance analysis is not available with CodeTEST Native:



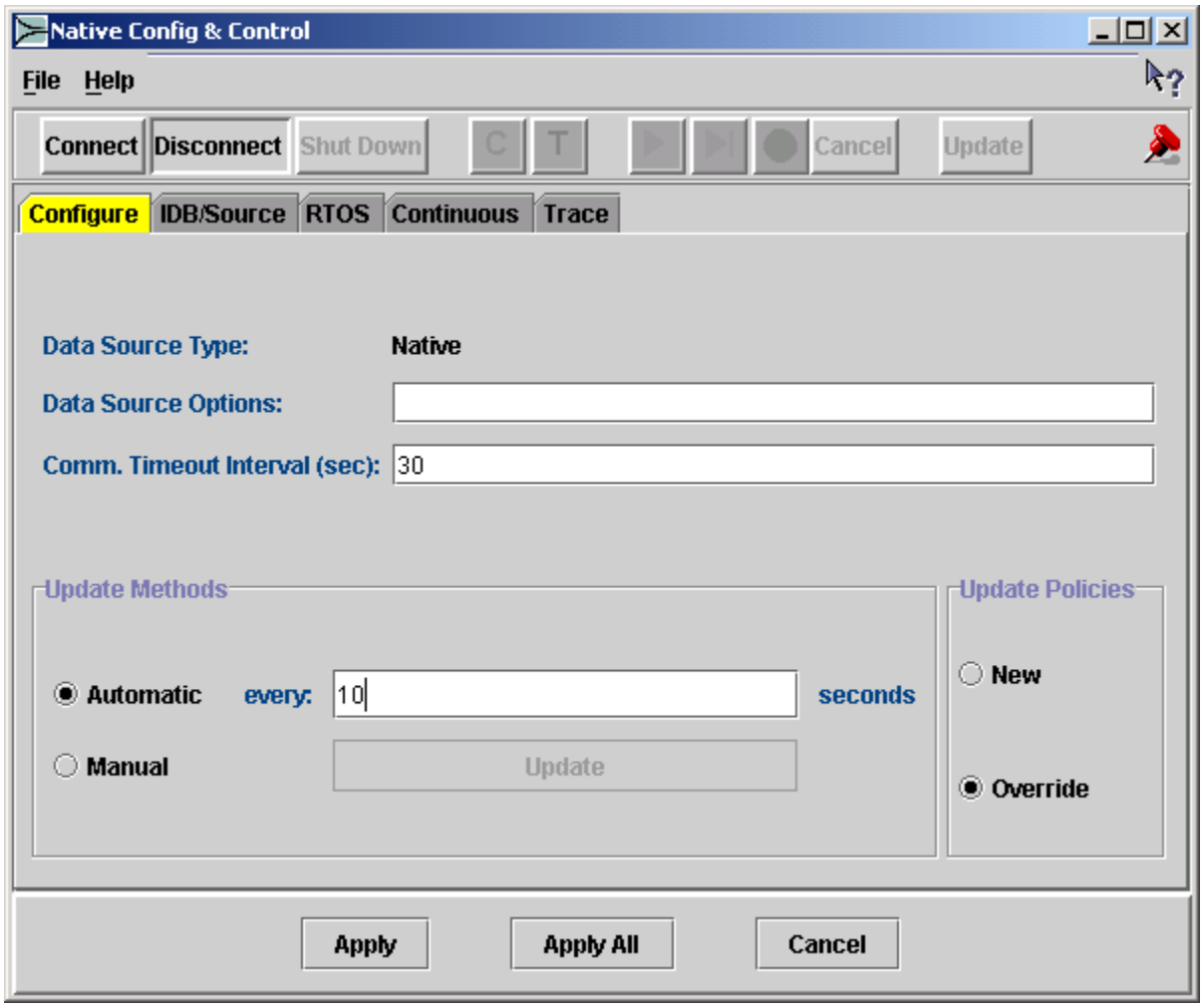
7) Once these changes have been made, press the OK button. CodeTEST measurements may now be made, e.g.:



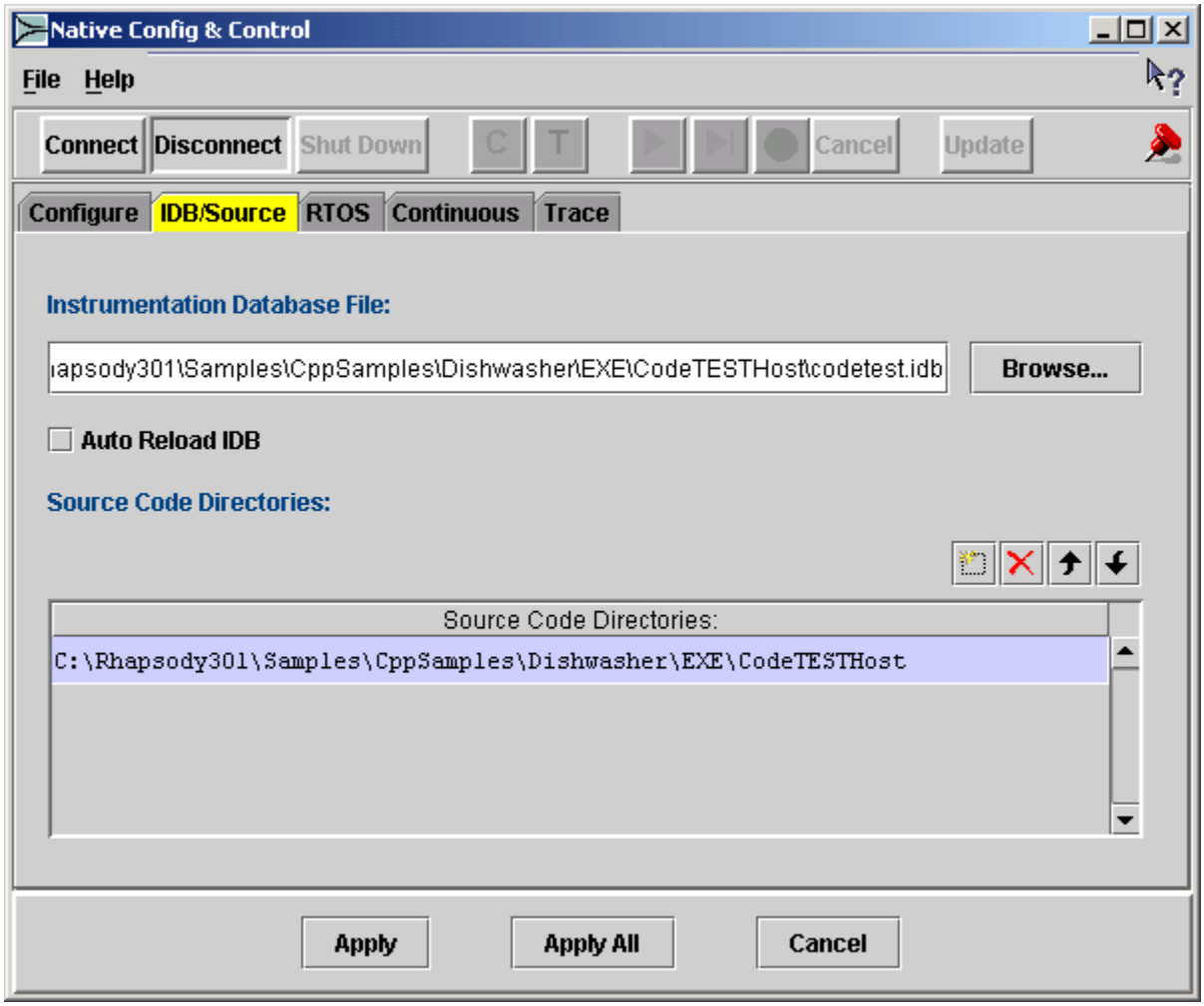
A detailed explanation of the CodeTEST analysis views and how to interpret the data presented will be left to the CodeTEST Users Guide.

3.1.4 Making CodeTEST v3.0 measurements

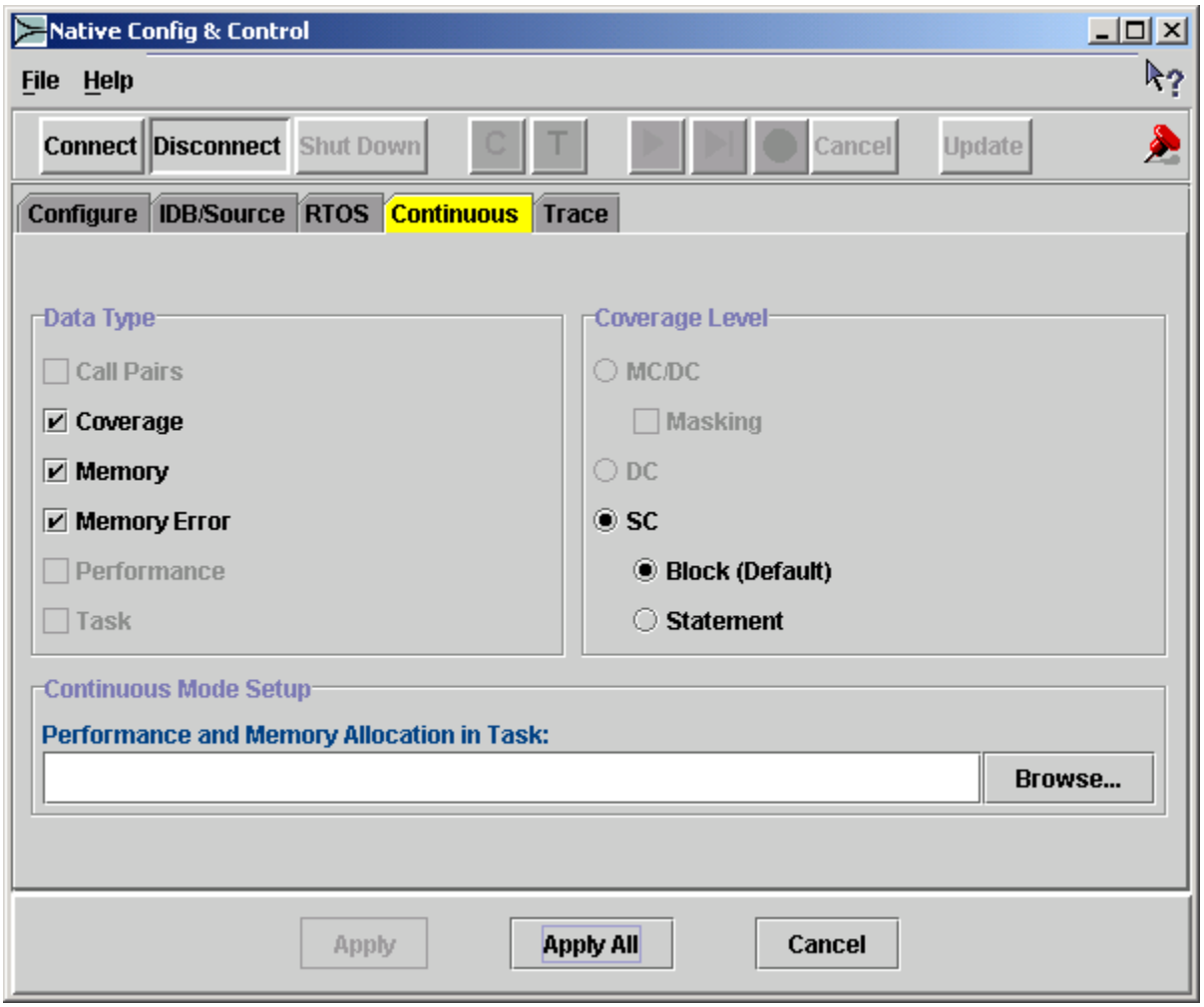
1) Start the CodeTEST v3.0 host application
2) From the DataSource menu, select Native… to get the Native Config & Control window.
3) With the Configure tab selected, set Update Methods to Automatic every 10 seconds. Also set the Communications Timeout Interval to 30 second.



4) With the IDB/Source tab selected, set the Instrumentation Database to OMROOT\Samples\CppSamples\Dishwasher\EXE\CodeTESTHost\codetest.idb. Also, add the OMROOT\Samples\CppSamples\Dishwasher\EXE\CodeTESTHost directory as a Source Code Directory:

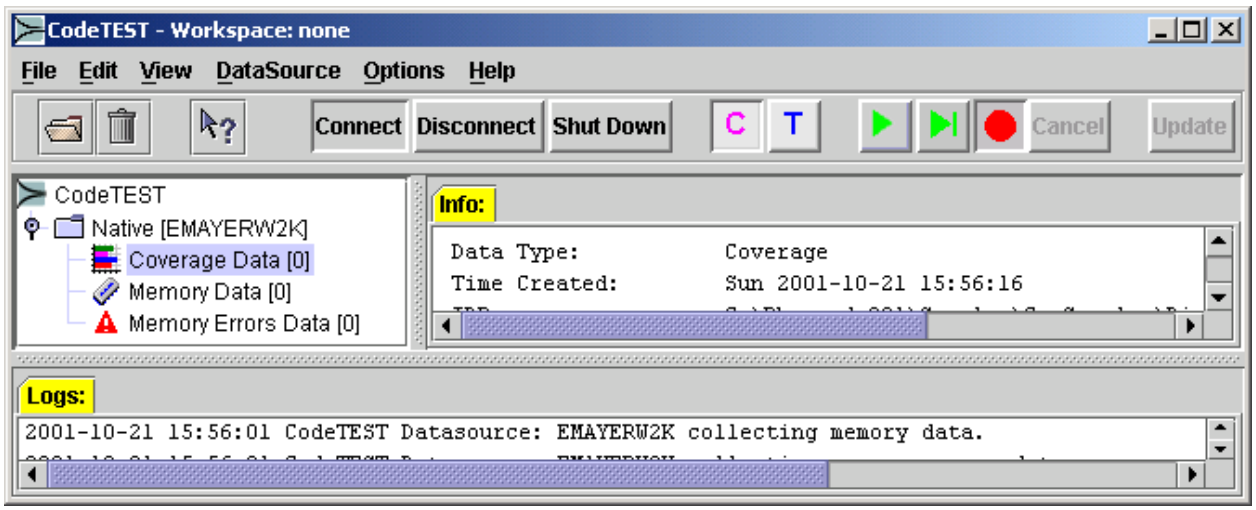


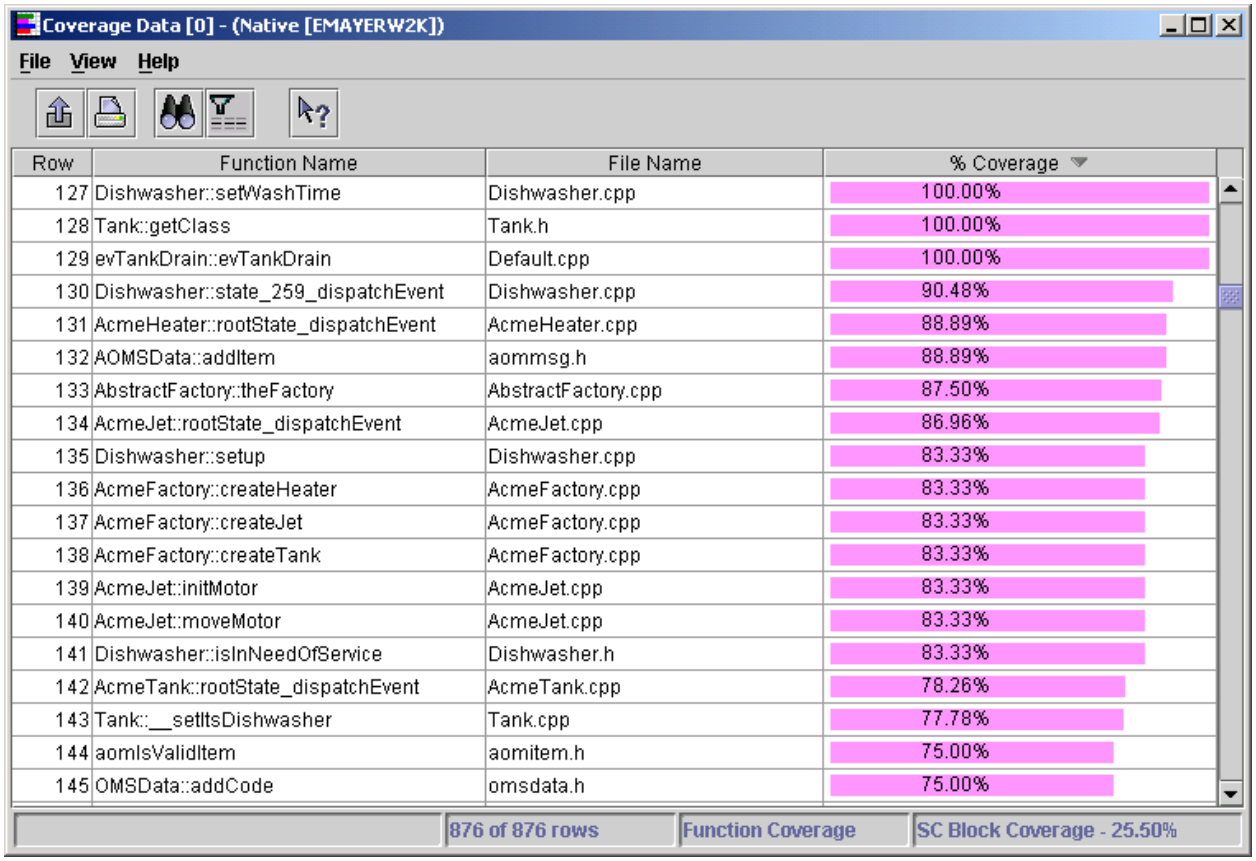
5) With the Continuous tab selected, ensure the Data Types and Coverage Levels for which you have licenses are selected. Note that Performance analysis is not available with CodeTEST Native:



6) Leave the Trace and RTOS tabs with their default settings. Once these changes have been made, press the Apply All button.

7) CodeTEST measurements may now be made. Once data is collected, the Workspace Pane will show the collected data sets:





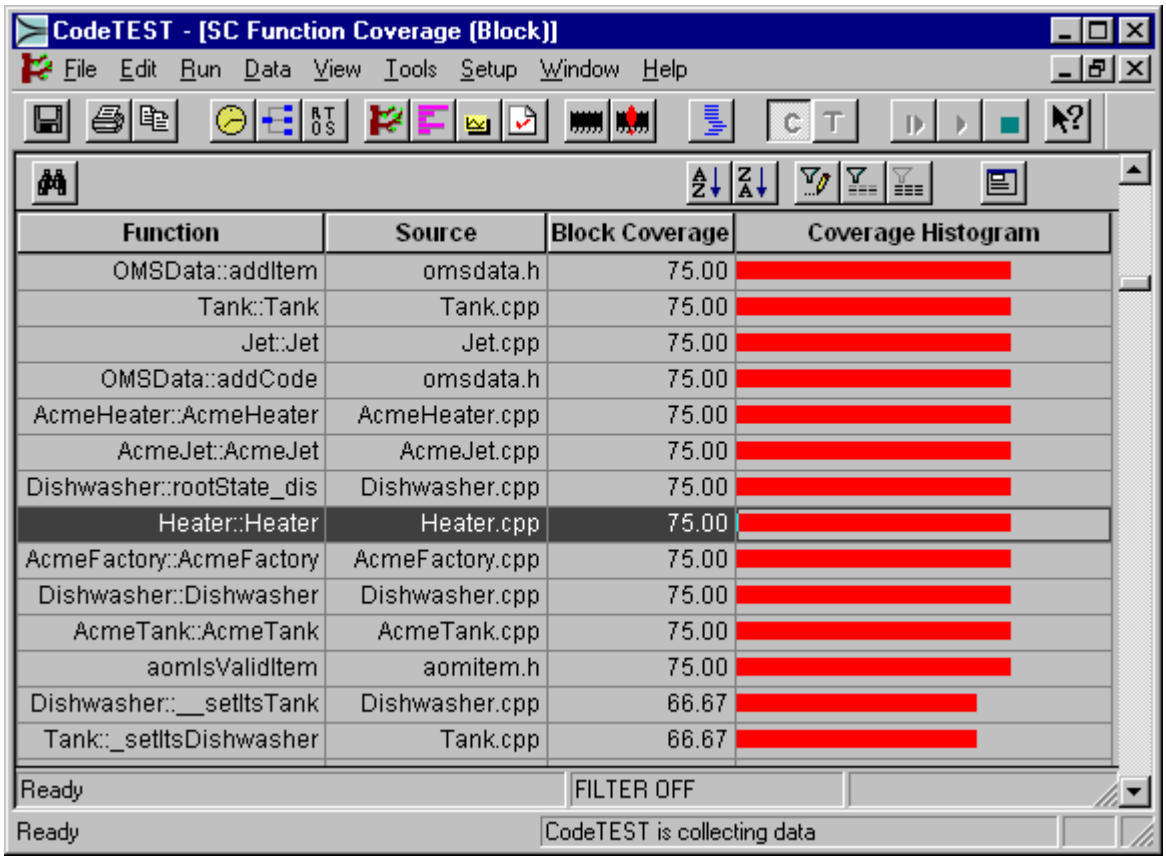
Row	Function Name	File Name	% Coverage
127	Dishwasher::setWashTime	Dishwasher.cpp	100.00%
128	Tank::getClass	Tank.h	100.00%
129	evTankDrain::evTankDrain	Default.cpp	100.00%
130	Dishwasher::state_259_dispatchEvent	Dishwasher.cpp	90.48%
131	AcmeHeater::rootState_dispatchEvent	AcmeHeater.cpp	88.89%
132	AOMSData::addItem	aommsg.h	88.89%
133	AbstractFactory::theFactory	AbstractFactory.cpp	87.50%
134	AcmeJet::rootState_dispatchEvent	AcmeJet.cpp	86.96%
135	Dishwasher::setup	Dishwasher.cpp	83.33%
136	AcmeFactory::createHeater	AcmeFactory.cpp	83.33%
137	AcmeFactory::createJet	AcmeFactory.cpp	83.33%
138	AcmeFactory::createTank	AcmeFactory.cpp	83.33%
139	AcmeJet::initMotor	AcmeJet.cpp	83.33%
140	AcmeJet::moveMotor	AcmeJet.cpp	83.33%
141	Dishwasher::isInNeedOfService	Dishwasher.h	83.33%
142	AcmeTank::rootState_dispatchEvent	AcmeTank.cpp	78.26%
143	Tank::__setItsDishwasher	Tank.cpp	77.78%
144	aomIsValidItem	aomitem.h	75.00%
145	OMSData::addCode	omsdata.h	75.00%

A detailed explanation of the CodeTEST analysis views and how to interpret the data presented will be left to the Getting Started Guide with CodeTEST Native document and the CodeTEST Users Guide.

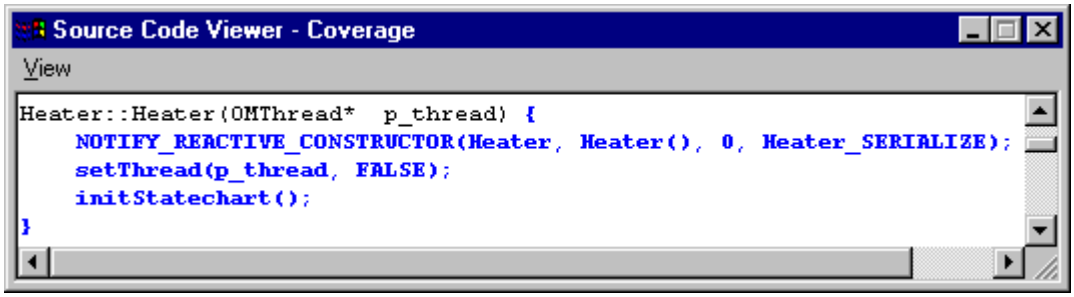
3.1.5 A note on coverage data collected when Rhapsody animation instrumentation is used

When Rhapsody animation instrumentation is used in conjunction with CodeTEST instrumentation, the resulting coverage information can appear misleading. This section describes how these measurements are accurately obtained and presented.

With some functions, most notably class constructors, the coverage information reported when code with Rhapsody animation instrumentation is run can appear misleading. For instance, in the CodeTEST v2.2 Function Coverage view below:



the constructor for class Heater is presented as having a Statement Coverage level of 75%. This means that for this function, 75% of the code statements have been executed. However, a view of the source code appears to have no decision points, suggesting 100% code coverage.



The reason for this has to do with how macros are expanded during pre-processing.

For the function Heater::Heater(), the animation instrumentation appears in the generated code as follows:

```
Heater::Heater(OMThread*  p_thread) {
    NOTIFY_REACTIVE_CONSTRUCTOR(Heater, Heater(), 0, Heater_SERIALIZE);
    setThread(p_thread, FALSE);
    initStatechart();
}
```

The NOTIFY_REACTIVE_CONSTRUCTOR() macro is the code that informs the Rhapsody host that this constructor has been executed. During the pre-processor stage of compilation, this macro expands to the following code:

```
OMMainThread::instance();
OMGetCurrentCallStack()->notify(this, omConstructorMethod );
AOMSMethod *aomsmethod;
if (withCallStackParameters(getClass()))
{
      aomsmethod = new AOMSMethod("Heater",0); ;;
}
else
      aomsmethod = new AOMSMethod("Heater()");
AOMEnterExit aomEnterExit(this, aomsmethod, omConstructorMethod );;
```

Notice that this single line of code has now been expanded into 10 lines of code with 2 decision points.

The CodeTEST instrumentation process involves first the preprocessing of each original source file to resolve any preprocessing directives (e.g., #include, #define, #if/#endif), followed by instrumentation via the CodeTEST instrumenter. This process is controlled by the CodeTEST compiler driver.

As the CodeTEST instrumentator is not run on this code until after the pre-processor stage of compilation, all macros are expanded. This means that with the basic heater constructor code above, which has only 3 code statements and no branch points, gets expanded to the following (i.e. 11 statements and 2 branch points):

```
Heater::Heater(OMThread*  p_thread) {
      OMMainThread::instance();
      OMGetCurrentCallStack()->notify(this, omConstructorMethod );
      AOMSMethod *aomsmethod;
      if (withCallStackParameters(getClass()))
      {
            aomsmethod = new AOMSMethod("Heater",0); ;;
      }
      else
            aomsmethod = new AOMSMethod("Heater()");
      AOMEnterExit aomEnterExit(this, aomsmethod, omConstructorMethod );;
      setThread(p_thread, FALSE);
      initStatechart();
```

```
}
```

Because the CodeTEST instrumenter is passed pre-processed code, it is this expanded version of the code that gets instrumented. The CodeTEST host, however, only sees the original source code, and this is where the discrepancy lies with the Function Coverage view.

NOTE: The CodeTEST Coverage Report will provide an accurate record of the coverage levels achieved.

APPENDIX 1 – SITEC++.PRP FILE EXAMPLE

This file was created from the factoryC++.prp file. The elements in the siteC++.prp file below that have changed from their factoryC++.prp file values have been highlighted using a gray background (e.g. this represents content that has changed, while this does not.)

<Beginning of file>

```
Subject CPP_CG
    Metaclass Microsoft
        Property CodeTESTUsage Enum "No,Yes" "No"
        Property CodeTESTCoverageLevel Enum "None,Statement Coverage,Decision Coverage,MC/DC"
"Statement Coverage"
        Property CodeTESTCompileSwitches MultiLine "-CTv -CTkeep -CTtag-allocator"
        Property MakeFileContent MultiLine "
############# Target type (Debug/Release) ##################
############################################################
CPPCompileDebug=$OMCPPCompileDebug
CPPCompileRelease=$OMCPPCompileRelease
LinkDebug=$OMLinkDebug
LinkRelease=$OMLinkRelease
BuildSet=$OMBuildSet
SUBSYSTEM=$OMSubSystem
COM=$OMCOM
RPFrameWorkDll=$OMRPFrameWorkDll

ConfigurationCPPCompileSwitches=$OMConfigurationCPPCompileSwitches

!IF \"$(RPFrameWorkDll)\" == \"True\"
ConfigurationCPPCompileSwitches=$(ConfigurationCPPCompileSwitches) /D \"FRAMEWORK_DLL\"
!ENDIF

!IF \"$(COM)\" == \"True\"
SUBSYSTEM=/SUBSYSTEM:windows
!ENDIF

!IF \"$(CodeTESTUsage)\" == \"Yes\"
!IF \"$(CodeTESTCoverageLevel)\" == \"Statement Coverage\"
CPP=ctcxx.exe $(CodeTESTCompileSwitches) -CTtag-level=SC
!ELSEIF \"$(CodeTESTCoverageLevel)\" == \"Decision Coverage\"
CPP=ctcxx.exe $(CodeTESTCompileSwitches) -CTtag-level=DC
!ELSEIF \"$(CodeTESTCoverageLevel)\" == \"MC/DC\"
CPP=ctcxx.exe $(CodeTESTCompileSwitches) -CTtag-level=MCDC
!ELSE
CPP=ctcxx.exe $(CodeTESTCompileSwitches)
!ENDIF
!ENDIF

################### Compilation flags ######################
############################################################
INCLUDE_QUALIFIER=/I
LIB_PREFIX=MS

################### Commands definition ####################
############################################################
LIB_CMD=link.exe -lib
LINK_CMD=link.exe
LIB_FLAGS=$OMConfigurationLinkSwitches
LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) /MACHINE:I386

############### Generated macros #################
##################################################
$OMContextMacros

######################### Predefined macros ##############################
##########################################################################
$(OBJS) : $(INST_LIBS) $(OXF_LIBS)

LIB_POSTFIX=
!IF \"$(BuildSet)\"==\"Release\"
```

```
LIB_POSTFIX=R
!ENDIF

!IF \"$(TARGET_TYPE)\" == \"Executable\"
LinkDebug=$(LinkDebug) /DEBUG
LinkRelease=$(LinkRelease) /OPT:NOREF
!ELSEIF \"$(TARGET_TYPE)\" == \"Library\"
LinkDebug=$(LinkDebug) /DEBUGTYPE:CV
!ENDIF


!IF \"$(INSTRUMENTATION)\" == \"Animation\"
INST_FLAGS=/D \"OMANIMATOR\"
INST_INCLUDES=/I $(OMROOT)\\LangCpp\\aom /I $(OMROOT)\\LangCpp\\tom
!IF \"$(RPFrameWorkDll)\" == \"True\"
INST_LIBS=
OXF_LIBS=$(OMROOT)\\LangCpp\\lib\\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX)$(LIB_EXT)
!ELSE
INST_LIBS= $(OMROOT)\\LangCpp\\lib\\$(LIB_PREFIX)aomanim$(LIB_POSTFIX)$(LIB_EXT)
OXF_LIBS=$(OMROOT)\\LangCpp\\lib\\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX)$(LIB_EXT)
$(OMROOT)\\LangCpp\\lib\\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT)
!ENDIF
SOCK_LIB=wsock32.lib

!ELSEIF \"$(INSTRUMENTATION)\" == \"Tracing\"
INST_FLAGS=/D \"OMTRACER\"
INST_INCLUDES=/I $(OMROOT)\\LangCpp\\aom /I $(OMROOT)\\LangCpp\\tom
!IF \"$(RPFrameWorkDll)\" == \"True\"
INST_LIBS=
OXF_LIBS= $(OMROOT)\\LangCpp\\lib\\$(LIB_PREFIX)oxftracedll$(LIB_POSTFIX)$(LIB_EXT)
!ELSE
INST_LIBS=$(OMROOT)\\LangCpp\\lib\\$(LIB_PREFIX)tomtrace$(LIB_POSTFIX)$(LIB_EXT)
$(OMROOT)\\LangCpp\\lib\\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX)$(LIB_EXT)
OXF_LIBS= $(OMROOT)\\LangCpp\\lib\\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX)$(LIB_EXT)
$(OMROOT)\\LangCpp\\lib\\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT)
!ENDIF
SOCK_LIB=wsock32.lib

!ELSEIF \"$(INSTRUMENTATION)\" == \"None\"
INST_FLAGS=
INST_INCLUDES=
INST_LIBS=
!IF \"$(RPFrameWorkDll)\" == \"True\"
OXF_LIBS=$(OMROOT)\\LangCpp\\lib\\$(LIB_PREFIX)oxfdll$(LIB_POSTFIX)$(LIB_EXT)
!ELSE
OXF_LIBS=$(OMROOT)\\LangCpp\\lib\\$(LIB_PREFIX)oxf$(LIB_POSTFIX)$(LIB_EXT)
!ENDIF
SOCK_LIB=

!ELSE
!ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified.
!ENDIF

################## Generated dependencies ########################
#################################################################
$OMContextDependencies

$(TARGET_MAIN)$(OBJ_EXT) : $(TARGET_MAIN)$(CPP_EXT) $(OBJS)
        $(CPP) $(ConfigurationCPPCompileSwitches) /Fo\"$OMFileObjPath\" $(TARGET_MAIN)$(CPP_EXT)

########################## Linking instructions ##############################
##############################################################################
$(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $(TARGET_MAIN)$(OBJ_EXT) $OMMakefileName
        @echo Linking $(TARGET_NAME)$(EXE_EXT)
        $(LINK_CMD)  $(TARGET_MAIN)$(OBJ_EXT) $(OBJS) $(ADDITIONAL_OBJS) \\
        $(LIBS) \\
        $(INST_LIBS) \\
        $(OXF_LIBS) \\
        $(SOCK_LIB) \\
        $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT)

$(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName
        @echo Building library $@
        $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS)
```

```
clean:
        @echo Cleanup
        $OMCleanOBJS
        if exist *$(OBJ_EXT) erase *$(OBJ_EXT)
        if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb
        if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT)
        if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk
        if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT)
"
        Property GetConnectedRuntimeLibraries String
"$(OMROOT)\\LangCpp\\lib\\msWebComponents.lib, $(OMROOT)\\lib\\msWebServices.lib, ws2_32.lib"
    end
end
```

<End of file>