OBJECTIME ®

C Target Module
5.2.1

Getting Started Guide
& Release Notice

Product Release: ObjecTime Developer 5.2.1 for C
Document Version: 1.0
Release Date: February 1999
Part Number: OT-R521-DOC810

ObjecTime Limited
340 March Road
Kanata, Ontario
Canada K2K 2E4

Printed in Canada

Important Notice

Copyright 1991-1999 ObjecTime Limited. All rights reserved.
Unpublished -- rights reserved under all Copyright laws including Copyright laws of the United States.
ObjecTime (and logo) is a registered trademark of ObjecTime Limited. Developer is a trademark of ObjecTime Limited.
The license management portion of this product is based on:
Elan License Manager © 1989-1999 Elan Computer Group, Inc. All rights reserved.
ObjecTime Limited (OTL) PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING ANY WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Information in this publication is subject to change from time to time without notice. Some states, provinces, or jurisdictions do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.
ObjecTime Limited (OTL) and its licensors retain ownership to the ObjecTime computer program and other computer programs offered by OTL (hereinafter collectively called "ObjecTime") and their documentation. **Use of ObjecTime is governed by the License Agreement associated with your purchase.**

Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraphs (a) through (d) of the Commercial Computer Software-Restricted Rights clause FAR 52.227-19 and its successors.
For units of the Department of Defense (DoD), the license for this software is subject to the "Restricted Rights" as that term is defined in the DFAR 252.227-7013 (c)(1)(ii), Rights in Technical Data and Computer Software and its successors.

The contractor/manufacturer is:

ObjecTime Limited
340 March Road
Kanata, Ontario
Canada, K2K 2E4

When acquired by the Government, commercial computer software and related documentation so legended shall be subject to the following:
(A) Title to and ownership of the software and documentation shall remain with the Contractor.
(B) User of the software and documentation shall be limited to the facility for which it is acquired.
(C) The Government shall not provide or otherwise make available the software or documentation, or any portion thereof, in any form, to any third party without the prior written approval of the Contractor. Third parties do not include prime contractors, subcontractors and agents of the Government who have the Government's permission to use the licensed software and documentation at the facility, and who have agreed to use the licensed software and documentation only in accordance with these restrictions. This provision does not limit the right of the Government to use software, documentation, or information therein, which the Government has or may obtain without restrictions.
(D) The Government shall have the right to use the computer software and documentation with the computer for which it is acquired at any other facility to which that computer may be transferred; to use the computer software and documentation with a backup computer when the primary computer is inoperative; to copy computer programs for safekeeping (archives) or backup purposes; and to modify the software and documentation or combine it with other software. Provided, that the unmodified portions shall remain subject to these restrictions.

COMMERCIAL COMPUTER SOFTWARE — RESTRICTED RIGHTS

(c) (1) The restricted computer software delivered under this contract may not be used, reproduced or disclosed by the Government except as provided in subparagraph(c)(2).
(c)(2) The restricted computer software may be —
(i) Used or copied for use in or with the computer or computers for which it was acquired, including use at any Government installation to which such computer or computers may be transferred;
(ii) Used or copied for use in or with backup computer if any computer for which it was acquired is inoperative;
(iii) Reproduced for safekeeping (archives) or backup purposes;
(iv) Modified, adapted, or combined with other computer software, provided that the modified, combined, or adapted portions of the derivative software incorporating any of the delivered, restricted computer software shall be subject to same restrictions set forth in this contract.
The following are trademarks or registered trademarks of their respective companies or organizations:
VxWorks, Tornado / Wind River Systems Inc. pSOS,pRISM,pRISM+ / Integrated Systems Inc. QNX / QNX Software Systems Ltd. LynxOS / Lynx Real Time Systems Inc. VRTX, MRI C++,Spectra / Microtec Inc. Green Hills C++ / Green Hills Software, Inc. Cygnus C++ / Cygnus Support. Watcom C++ / Sybase Inc. Elan License Manager / Elan Computer Group, Inc. OPEN LOOK, UNIX / UNIX System Laboratories, Inc. FrameMaker, FrameViewer, PostScript, Acrobat / Adobe Systems, Inc. Hewlett-Packard / Hewlett-Packard Company. SGI R3000, R4000, IRIX / Silicon Graphics Inc. AIX, IBM, PowerPC, RISC System/6000 / International Business Machines Corporation. WindowsNT, VisualC++,Visual Source Safe / Microsoft Corporation. Sun Microsystems, Sun Workstation, OpenWindows, Solaris, SunView, SPARC, SPARCstation / Sun Microsystems, Inc. X Window System, X11 / Massachusetts Institute of Technology. Smalltalk-80, ObjectWorks/Smalltalk / ParcPlace Systems, Inc. GNU / The Free Software Foundation. ClearCase, Purify /Pure Atria Corporation. Rational. Netscape, Netscape Navigator, and the Netscape N logo are registered trademarks of Netscape Communications Corporation in the United States and other countries. Microsoft, Windows, and Windows NT are either trademarks or registered trademarks of Microsoft Corporation. All other brand names are trademarks of their respective holders.

ObjecTime Support

Your opinions and suggestions are both welcome and vital to the evolution of ObjecTime Developer.

ObjecTime Support

ObjecTime Support Hotline: (613) 591-3400

ObjecTime Support E-mail: support@objectime.com

ObjecTime Sales

Sales Hotline outside the Ottawa area: 1-800-567-TIME

Sales Hotline within the Ottawa area: (613) 591-3831

Sales Email: sales@objectime.com

ObjecTime Limited

ObjecTime Fax: (613) 591-3784

Visit our Web Site: www.objectime.com

Table of Contents

Welcome to ObjecTime Developer for C 5.2.1 1

Introduction 1

What's new in Developer for C 5.2.1/5.2 2

Year 2000 Compliance. 4

Installation Information 5

Naming Conventions 6

Documentation Errata: 7

Supported Host & Reference Platforms 9

Changes in Developer 5.2.1/5.2 11

Model upgrade pre-5.2.1 to 5.2.1 11

5.2.1 API changes: 11

Serial line target observability 13

Limited support for 8.3 compilers (OTD 5.2.1) 15

OSE: An example port (OTD 5.2.1) 16

Issues with running earlier models 17

Naming changes and impact to user models 17

Application Programmer Interface (API) 18

Semantics 18

User Code changes necessary for new C Target Services Library 19

External debugging 20

Problems addressed in this release 20

General Information 21

Limits 21

C TargetRTS Services Library Limits 21

Special Notes and Reminders 21

Compliance Models 22

User Guide — Differences When Using the C Target Services Library 23

Troubleshooting 27
Compilation problems — Windows NT 27

Developer for C Directory Contents 29

Known Limitations/Restrictions 31
Inclusion Paths 32
Solaris Multi-threaded 32
Relay Ports 32
Target Observability 32
Tornado 32
Known Problem Information 32

Tornado Integration on NT 33
Using Tornado on Windows NT 33
Environment setup 33
Overview of the ObjecTime/Tornado integration tools 38
What Tornado Needs 39
What ObjecTime Needs 40
Tornado Integration 41
Configuration 41
Source Breakpoints 44
Source Breakpoint Hit 45
Help 47

Integrating Developer Studio on Windows NT 49
Overview. 49
Configuration 50
Source Breakpoints 54
Source Breakpoint Hit. 55

Chapter 1

Welcome to ObjecTime Developer for C 5.2.1

Introduction

The ObjecTime Developer for C 5.2.1 release provides general enhancements aimed at minimizing the memory requirements for the use of visual debugging on targets with constrained foot print requirements. This release also introduces visual debugging on target using serial port access as a complement to the current TCP based visual debugging on target. ObjecTime Developer for C 5.2.1 also introduces support for the Enea OSE 3.1 platform.

Please see the *ObjecTime Developer 5.2.1 Getting Started Guide and Release Notice* for information about new toolset-related features in ObjecTime Developer 5.2.1, and the following sections for information related to C language-specific features introduced in this release.

This chapter provides an introduction to using ObjecTime Developer for C 5.2.1. There are four main areas:

- What's new in ObjecTime Developer for C 5.2.1/5.2
- Year 2000 Compliance
- Installation Information
- Naming Conventions
- Documentation Errata

What's new in Developer for C 5.2.1/5.2

The following list highlights some of the key new features in **ObjecTime Developer 5.2.1**.

- **Serial Port Access**: This release adds serial communication support to the target observability feature of ObjecTime Developer 5.2.1. Currently, the serial support is only available for the C Target Services Library. See "Model upgrade pre-5.2.1 to 5.2.1" on page 11, Chapter 3, Changes in Developer 5.2.1/5.2.
- **C Target Services Library internal architectural improvements:** There have been numerous enhancements to the C Target Services Library between the 5.2 version and the 5.2.1 version. These enhancements include:
  - The porting process has been improved such that functions that need to be overridden during a port have been put into separate files
  - Improvements to system-wide timers and to per-thread timers
  - Thread synchronization algorithms have been improved
  - Code has been modularized to improve readability
  - Code has been re-written to minimize global lookups
  - Support for Serial Line Target Observability
  - 8.3 filename compliance
  - Problems reported in 5.2 have been fixed
  - Example ports to new platforms have been added
  - Use of large footprint function calls have been reduce.
  - Footprint reduction and speed improvements
- **OSE: An example port:** OSE separately creates then runs it's threads, rather than having one function that does both. See "OSE: An example port (OTD 5.2.1)" on page 16, Chapter 3, Changes in Developer 5.2.1/5.2 for further information.
- **Limited support for 8.3 compilers**: The code generation process appends extensions such as _Actor, _Data, _Protocol and _Package to generated header and implementation files. This can cause problems if you are using a compiler which insists on 8.3 filenames. Some limited support for this kind of compilation does exist. See "Limited support for 8.3 compilers (OTD 5.2.1)" on page 15 for more information.
- **Developer WebPublisher**: The Developer WebPublisher 5.2.1 optional product is available for use with all three of the ObjecTime Developer 5.2.1 product packages. Please see the ObjecTime Developer 5.2 *User Guide*, Web Model Publisher, Chapter 27, and the enclosed product information sheet for details about Developer WebPublisher's capabilities.
- **Developer TestScope:** The Developer TestScope 5.2.1 optional product is available for use with all three of the ObjecTime Developer 5.2.1 product packages. Developer TestScope extends ObjecTime Developer's design-automation capabilities to model debug and test. Please see the product information sheet enclosed with the ObjecTime Developer 5.2.1 documentation for details about Developer TestScope's capabilities.

The following list highlights some of the key new features in **ObjecTime Developer 5.2**.

- **C actor toolset integration**: C actors are now fully integrated into the toolset, with appropriate modeling enforcement where needed.
- **Target Observability for C**: Target Observability of a running executable from within the toolset has been added. This includes support for port injection and message tracing.
- **Timestamp driven compilation**: ObjecTime Developer for C now uses industry standard timestamp driven compilation. Only modified or affected-by classes are recompiled.
- **Generated code persistence**: Code generation only takes place when classes are changed. As well, generated files and compile outputs are never deleted, only overwritten.
- **External code generation and compilation**: Code generation and compilation can be performed outside the toolset via make. This is useful in situations where a large model needs to be recompiled. Users can now trigger this activity outside the toolset so they may continue to work within the toolset unhindered.
- **Use of source file pairs**: The generated C code is now in the form of one .h and one .c file created for each actor and protocol class.
- **Reuse of build products**: Reuse is now at the class level versus package level. Designer sessions can pick up loadbuild results during compile. For ClearCase users, this is accomplished by using derived objects made visible in user view. GNU make users can do so by setting the Load Build Paths property in Update Properties Editor, and others can do so by manually copying loadbuild results.
- **Internal model dependencies**: Automatically generated. You can also add dependencies for situations where you require access to the public interface of an ObjecTime component in another ObjecTime component.
- **External compile dependencies**: Changes to user-specified dependencies, explicitly created through external .h inclusions, are now appropriately considered during the compilation phase, resulting in incremental recompilation as necessary.
- **New toolset host support**: IBM AIX, Silicon Graphics IRIX, NT and SunOS are officially supported as host and target development environments.
- **Tornado integration on NT**: WindRiver Tornado has been added as a target development platform. Refer to Appendix C of this guide for information on integrating and using Tornado.
- **C Target Services Libraries**: These are re-compiled libraries which are linked during each update compile, rather than re-compiling every time an update is compiled.
- **Code-generation pattern**: The new code-generation pattern is smaller and more efficient.
- **Debugger**: The new debugger for Developer for C is similar to the C++ debugger.

Year 2000 Compliance

Complete Year 2000 testing has been performed by ObjecTime Limited, including correct handling of leap year calculations. ObjecTime Developer 5.2.1 is year 2000 compliant. The ObjecTime Developer class libraries will function correctly across the year 2000 boundary with one clarification. The RPL Date class allows year to be specified as either a two digit (interpreted as 2000 - 2050 if the entered year is less than 51, or interpreted as 1951 - 1999, if the entered year is greater than or equal to 51) or a four digit (relative to the start of the Roman calendar) number. It is recommended that existing models be converted to use the four digit year format.

Note: For further details on ObjecTime Limited's Year 2000 Compliance Policy please visit:

http://www.objectime.com/otl/about/y2k.html

The license keys used by the License Manager are year 2000 compliant, with the exception of the License Manager log file, which lists only the two last digits of the year.

It is recommended that you review the Year 2000 compliance policies and statements from the vendors of your operating system, development tools and configuration management software.

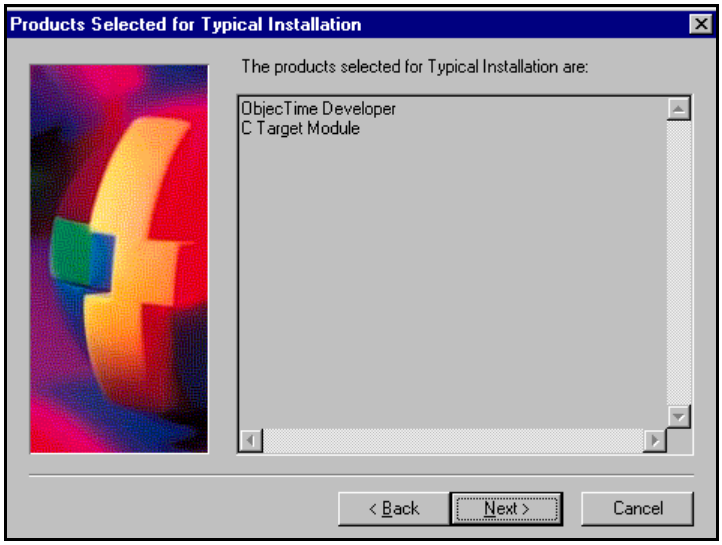
Installation Information

The C target module can be installed either as an upgrade to the ObjecTime Developer 5.2.1 base, or as a complete product package which includes the base. In either scenario, the installation instructions are similar and the *ObjecTime Developer 5.2.1 Getting Started and Release Notice* provides complete details on the installation instructions.

The only difference between the base ObjecTime Developer installation and the ObjecTime Developer for C installation will be the number of packages that are to be installed, along with your installation keys. The following figures represent the differences between the base install and an ObjecTime Developer for C install for the two different platforms — UNIX and Windows NT.

> **Note:** The default printer requirement is, at minimum, a UNIX or Windows NT compatible printer. We recommend a PostScript™ printer.

Figure 1 On Windows NT



If installing on a UNIX system, a typical install would appear as in the shell script that follows:

Figure 2 On Unix

```
Reading the setup directory...
Press T<ENTER> for Typical Installation,
or C<ENTER> for Custom Installation: t<ENTER>
The products selected for Typical Installation are:
ObjecTime Developer
C Target Module.
```

Naming Conventions

In **ObjecTime Developer for C 5.2** the C TargetRTS was renamed to the **C Target Services Library** to better reflect the implementation of the C TargetRTS and to clearly identify the different libraries. Throughout the documentation set, the terms cRSL, cRTS, C Target or C TargetRTS refer to the C Target Services Library.

Documentation Errata:

Note the following updates to the information contained in the **ObjecTime Developer *C Language Guide***. Please read and note the following changes to your documentation set.

Add the following to Chapter 5, in the Reserved Names list, page 54:

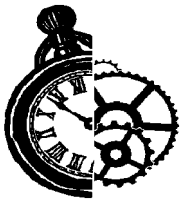
`getId`.

Note: `getId` cannot be used as a signal name.

Add the following to Chapter 2, in the Macros that require 'msg' or '_actor' section, page 28:

See "Run-Time Actor Instance Identification" on page 30 for more information.

Chapter 2



Supported Host & Reference Platforms

The following table shows the supported host platforms for **ObjecTime Developer for C 5.2.1**.

5.2.1 Host Platforms

Toolset Host	Simulation Services Library Name
Solaris 2.6	SUN5.sparc-gnu-2.8.1
Windows NT 4.0	NT40.x86-VisualC++-5.0, -VisualC++6.0
HPUX 10.20	HPUX10.hppa-gnu-2.8.1
AIX 4.2.1	AIX4.ppc-gnu-2.8.1
IRIX 6.2	IRIX6.r4400-gnu-2.8.1
Sun OS 4.1.3	SUN4.sparc-gnu-2.8.1

Reference ports are ports delivered as part of the standard ObjecTime Developer for C product. These ports are fully tested by ObjecTime, and are covered by standard ObjecTime support. A reference port can be used to facilitate a port to your environment of choice.

A reference port is based on the following specifics:

- OS version
- Compiler version
- Processor type

If you are using a line-up other than the one tested by ObjecTime and listed in this guide, standard support will cover problems encountered by you only to the extent that the problem is reproducible on the line-up listed in this guide.

5.2.1 Reference Platforms & Ports

Toolset Host	Target Services Library Name	Target Services Library
AIX 4.2.1 (PowerPC)	AIX4S.ppc-gnu-2.8.1	Supplied
HPUX 10.20	HPUX10S.hppa-gnu-2.8.1	Supplied
IRIX 6.2	IRIX6S.r4400-gnu-2.8.1	Supplied
Solaris 2.6	SUN5S.sparc-gnu-2.8.1 SUN5T.sparc-gnu-2.8.1	Supplied
SunOS 4.1.3	SUN4S.sparc-gnu-2.8.1	Supplied
Windows NT 4.0	NT40S.x86-VisualC++-5.0 NT40T.x86-VisualC++-5.0 NT40S.x86-VisualC++-6.0 NT40T.x86-VisualC++-6.0	Supplied
Solaris 2.6 Windows NT 4.0	TORNADO101S.ppc-cygnus-2.7.2-960126 TORNADO101T.ppc-cygnus-2.7.2-960126 (Tornado 1.0.1: cygnus tools for VxWorks 5.3.1 on PowerPC)	Generate
Solaris 2.6 Windows NT 4.0	OSE31T.ppc603-Diab-4.1a (Diab 4.1a, SDS 7.1.1 tools for OSE 3.1 for PowerPC)	Generate

Note: The following applies to the Tables in this chapter:

- **S** = Single-threaded
- **T** = Multi-threaded

  Simulation Services Libraries don't have an 'S' or a 'T' thread indicator in their names.
- **Supplied** = Simulation Services Libraries and Target Services Libraries are supplied as part of the ObjecTime Developer for C installation.
- **Generate** = not supplied as part of the ObjecTime Developer for C installation, but can be generated from source code that is supplied.

Chapter 3

Changes in Developer 5.2.1/5.2

This section is of particular interest to customers who have used previous releases of ObjecTime Developer. It describes new features that are available for ObjecTime Developer for C (version 5.2.1/5.2) that were not available with ObjecTime Developer for C (version 5.1).

Model upgrade pre-5.2.1 to 5.2.1

Due to enhancements introduced in 5.2.1, a minor conversion effort will be required to bring your pre-5.2.1 models to 5.2.1. Specifically, changes have been introduced in the area of timers and some API changes. In many cases, the only actions that you will have to do are:

1 open the C_Timers update and

2 drag and drop the new timer actors into your model update.

If, however, you have coded your own timer actors, or used functions that were meant for timer design in your model, you will have to update some function calls.

It is also recommended that integrated timers be redesigned into a single actor, and that actor be dragged and dropped into every thread requiring timing services.

5.2.1 API changes:

RSLPortEnqueue

```
RSLMessage *

RSLPortEnqueue( RSLActorIndex RSLExecutingActor , RSLPortIndex
portOffset ,

   RSLSignalIndex signal , RSLMessagePriority priority , void
*data )
```

is now:

```
RSLMessage *
```

```
RSLPortEnqueue( RSLActorIndex      RSLExecutingActor ,
      RSLPortIndex       portOffset ,
      RSLSignalIndex     signal ,
      RSLMessagePriority priority ,
      void *             data ,
      RSLActorIndex      FromActor )
```

The function was changed to include the sending actor as a parameter to calculate whether the message is an intra-thread send or an inter-thread send. If both actors are on the same thread, then it is an intra-thread send.

RSLRegisterTimerServices

```
void

RSLRegisterTimerServices( RSLActorIndex RSLExecutingActor ,
          RSLPortIndex port , void *cv ,
          void( *sigfunc )( void * ) )
```

is now:

```
void

RSLRegisterTimerServices( RSLActorIndex RSLExecutingActor ,
           RSLPortIndex port ,
          void( *sigfunc )( void * ) )
```

The cv parameter was eliminated, as it was unnecessary. The pointer to the instance data for the actor is stored, and passed back to associated functions. The CV, if necessary can be a part of the Instance Data (ESVs)

RSLRegisterMessageSignallingInterface

```
void

RSLRegisterMessageSignallingInterface( RSLActorIndex actor ,
void *cv ,
```

```
                    void( *sigfunc )( void * ) )
```

is now:

```
void

RSLRegisterMessageSignallingInterface( RSLActorIndex actor ,

                void( *sigfunc )( void * ) )
```

The cv parameter was eliminated, as it was unnecessary. The pointer to the instance data for the actor is stored, and passed back to associated functions. The CV, if necessary, can be a part of the Instance Data (ESVs).

RSLGetFirstTimeout / RSLTimerReference

```
RSLGetFirstTimeout( RSLActorIndex RSLExecutingActor )
```

is now:

```
void

RSLGetFirstTimeout( RSLActorIndex RSLExecutingActor ,

        RSLTimerReference * ThisTimer     )
```

The RSLTimerReference return parameter has been moved to being a parameter. This has been done for several reasons. Some compilers don't accept a struct as a return parameter. Second, returning a struct implies that the struct must exist in the stack in the calling function and the called function, so this eliminates one stack instance.

RSLGetMyThread - new function: RSLThreadIndex

```
RSLGetMyThread( RSLActorIndex actor )
```

This function has been added in case a timer design needed to know on which thread a certain actor was running.

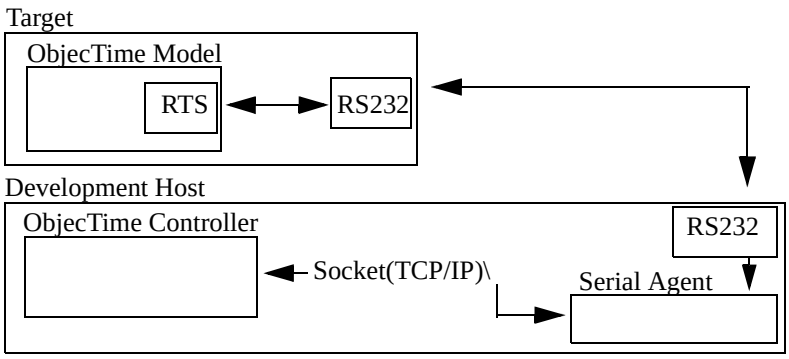
Serial line target observability

ObjecTime Developer for C 5.2.1 supports Serial IO in the C Target Services Library. When using serial line target observability, note that additional arguments to configure the serial port are required when starting the ObjecTime model. See "Command line options" on page 15.

Serial Agent application and source code are added to the ObjecTime Developer 5.2.1 suite of tools. The Serial Agent acts as a proxy for the target, shuffling target observability communications between the ObjecTime Controller and the serial port which is connected to the target.

Serial line target observability permits ObjecTime Developer's target observability features, without a target TCP/IP stack or network hardware. Instead target observability communications are sent via RS232 serial line to the development host machine, running the Serial Agent proxy application. The Serial Agent then relays the target observability communications to the ObjecTime Controller. The following figure illustrates the components of the serial line target observability system.

Figure 3 Serial line target observability system



System requirements

Serial Line target observability has the following requirements or restrictions:

- a target with a RS232 serial port.
- a port of the Target Services Library with serial line target observability enabled.
- a development host with a RS232 serial port and TCP/IP stack.
- the Serial Agent proxy, which is provided for all ObjecTime Developer platforms.

Note: Currently, only the C Target Services Library supports Serial Line Target Observability.

Platforms supported

The Serial Agent is supported for all of the ObjecTime Developer development platforms (AIX4, HPUX10, IRIX6, NT4, SUN4, SUN5). The following C Target Services Library target platforms are supported:

AIX4S.ppc-gnu-2.8.1

HPUX10S.hppa-gnu-2.8.1

IRIX6S.r4400-gnu-2.8.1

NT40T.x86-VisualC++-6.0

SUN4S.sparc-gnu-2.8.1

SUN5T.sparc-gnu-2.8.1

Command line options

With the addition of serial support in the C Target Services Library, some additional command line options are required to start the ObjecTime model with target observability. The standard options, '`-connect=`' and '`-name=`', are still required.

```
-connect=<port>@<host>
```

Where <`port`> is the TCP/IP port of the ObjecTime Controller, and <`host`> is the TCP/IP address or host name of the computer on which the Controller is executing. The port number when connecting to the Controller with the C Target Services Library is Master Port + 1. The Master Port value can be found from the External Access menu item in ObjecTime Developer 5.2.1.

```
-name=<name>
```

Where <`name`> is the name the target will use when connecting to ObjecTime Developer 5.2.1. The correct <`name`> is shown in the title bar of the RTS Control Panel.

```
-serial=<dev>
```

Where <`dev`> is the file name of the serial device. On UNIX systems <`dev`> is something like '`/dev/tty1`'. On Windows NT <`dev`> should be something like 'COM1'.

```
-baud=<baud>
```

This optional argument sets the serial port baud rate to <baud>. If this option is not specified, the baud rate will default to 9600.

The Serial Agent also requires the setting of command line options. The syntax for invoking Serial Agent is as follows:

serialagent `<dev> [<baud>]`

Where <`dev`> is the name of the serial device, and <`baud`> specifies what baud rate will be used. As with the target, the baud rate is optional and defaults to 9600 if not specified.

Limited support for 8.3 compilers (OTD 5.2.1)

The following applies to C, C++ Target and C++ Simulation equally.

The code generation process appends extensions such as _Actor, _Data, _Protocol and _Package to generated header and implementation files. This can cause problems if you are using a compiler which insists on 8.3 filenames. Some limited support for this kind of compilation does exist.

It should be noted that the file system cannot be restricted to 8.3 filenames. Similarly the Make executable must be able to support non-8.3 Makefile fragments (such as Foo_Package.mk).

To activate 8.3 compiler support, set the environment variable OBJECTIME_8DOT3 to a non-zero value, restart the session and regenerate all.

Restrictions:

- The file system must still support non-8.3 filenames
- The Make executable must be able to support non-8.3 Makefile fragments (such as Foo_Package.mk)
- Obviously all classes and packages must be a maximum of 8 characters long.
- All packages and all classes now belong to a common name-space. The toolset checks that all package names are unique and all class names are unique, but does not check for package names conflicting with class names.
- The toolset does not check for case-insensitive uniqueness. If your compiler requires 8.3 filenames, it is likely not case-sensitive.
- The link objects list file (ALL_OBJS.olist) is not 8.3, but its contents are. If the linker does not like the name ALL_OBJS.olist, you must write a link.pl script to rename it and use the renamed file.
- An 8.3 compiler likely insists that all include paths must also be 8.3. The path to the RTS Home directory (typically "$(OBJECTIME_HOME)") must be composed of 8.3 sub-directories. The Services Library name must be 8.3, for example, "TargetRTS" -> "target.rts".
- The Library name must be no more than 3 characters, for example, "VisualC++-5.0" -> "vc5". The Target Platform name must be no more than 7 characters long, for example, "TORNADO101" -> "tornado". This is to accommodate the linker which refers to $(PLATFORM)$(THREADED_FLAG).$(LIBRARY_NAME) as a subdirectory, which likely must be 8.3.

OSE: An example port (OTD 5.2.1)

The following is included here to illustrate changes required for a recent port. Please note that ports vary and your port may be different.

OSE separately creates then runs it's threads, rather than having one function that does both. It was necessary to override the CRSL/RTCrUsTh.c and the CRSL/RTSrUsTh.c to call the appropriate functions.

OSE required the use of a signalling mechanism.

- It was therefore necessary to re-write the thread sending functions in `CRSL/RTPrtSnd.c` and create entirely new functions in `CRSL/RTPrtSig.c`.
- A performance enhancement was added to the Target Services Library for the OSE 3.1 platform. The interthread messaging now uses the native signal message passing mechanism of OSE.

- `CRSL/RTPrtSnd.c` was overridden to call the new functions instead of the standard semaphore posting functions, should `RSLSIGNALLING==RSLTRUE` be activated
- `CRSL/RTThrRun.c` was overridden to not check for external events, since the signalling mechanism can put the events into the internal queues in a thread safe manner. It was also recoded so that a function `RSL_retrieveEvent` was called instead, to check if there were any new signalling events.
- These functions were put in `CRSL/RTThrSig.c`
- The standard input for OSE was different, so `DEBUG/debugio.c` was overridden to read in new input from the debugger command line.
- `FUNCTION/RTfflush.c` was recoded as an empty function stub, since OSE does not provide the function call for the `fflush` function.
- `INITSTOP/TGTinit.c` was overridden since OSE has special functions that need to be called upon system startup, like the error handler, and the signal buffer creation functions.
- `MAIN/main.c` was overridden since you must code your own Task startup routine, along with creating your initial thread.
- `TCP/` The functions in the TCP directory were modified to call the OSE specific tcp function calls.
- `THREAD/RTThread.c` was overridden to call the OSE specific thread function calls.

Issues with running earlier models

ObjecTime Developer for C 5.2 no longer supports the environmental variables OBJECTIME_CRTS_HOME and OBJECTIME_CRTS_TARG. Before attempting to run earlier models, users must ensure that they have set-up their compilation environment properly (and, if necessary, generated an appropriate C Target Services Library. Earlier releases of ObjecTime Developer for C required that the C Target Services Library code be compiled *every* time an update was compiled). The standard release is delivered with a standard set of target libraries for various target platforms, so depending upon the desired target, it may be possible to work with one of the supplied libraries. See the *C Language Guide* for information on how to compile C Target Services Libraries and updates in ObjecTime Developer for C 5.2.1.

Subsequent sections document several other enhancements to ObjecTime Developer for C 5.2.1. Although it is possible that an earlier update will run without modification in ObjecTime Developer for C 5.2.1, it is likely that some modifications to customer updates will be required, particularly if the model is multi-threaded or uses built-in RSL types directly.

Naming changes and impact to user models

In **ObjecTime Developer for C 5.2**, the names of all run-time library support interfaces and variables have been prefixed with ‘RSL’, which is an initialism meaning “Run-time Services Library”. The prefix “cRTS” is no longer used. Also, all RSL base types have been defined via typedef (that is, struct keyword is no longer required). Since access to most RSL services is via ROOM interface macros, for example, ROOM_PortSend, most of these changes are hidden from users.

The RSL function “cRTSRegisterMainloop” has been renamed to “RSLRegisterExternalInterface”, to more properly reflect its intended operation. See “Run-Time Services” on page 159 of the *C Language Guide* for more information on the C Target Services Library service routines.

The name of the thread map specification function has been changed from ‘cRTSThreadMap’ to ‘RSLThreadMap’. See the next sections for a discussion on semantic changes.

For header file compatibility reasons, it is required that the keyword ***timeout*** not be specified in any user-defined code; instead, the macro RSL_Timeout() should be used (thus, at compilation time, the correct keyword ‘timeout’ or ‘RSLTIMEOUT’ will be generated. You must, however, continue to use ‘timeout’ within the toolset for associating events with transitions).

Application Programmer Interface (API)

In **ObjecTime Developer 5.2**, a new Run-time Services Library function ‘RSLRegisterSignallingInterface’ was introduced, which allows local (per-thread) timer implementations to register a special function which the C Target Services Library will invoke when an inter-thread message is delivered to that thread. Thus, it is now possible to efficiently implement local timers with ObjecTime inter-thread messaging (via actor port bindings). An example of this type of interface has been provided. See “C_TornadoQueuesWithTimers” on page 234 of the *C Language Guide* for further information.

The macro ROOM_Signal is still supported in ObjecTime Developer (for compatibility reasons with earlier releases), but it is no longer required for the specification of signal names. In this release, all signal names are globally defined to be equivalent across all protocol classes (thus allowing switching of signals across compatible protocol classes). This does continue to imply, however, that all signal names must be unique symbols in the global compilation space (for example, avoid the use of signal names that are C/C++ keywords, or are likely to be defined in system header files).

Users no longer have a configuration option of disabling/enabling message priority levels, as message priority is now *always* enabled. Also, a new priority level (for *internal* C Target Services Library use to support system/debugger messaging) has been added.

Semantics

In **ObjecTime Developer 5.2**, the name of the thread map specification function has been changed from ‘cRTSThreadMap’ to ‘RSLThreadMap’. In addition, since compilation settings are now configured in the toolset (where physical thread priorities/stack-sizes are specified), these keywords are no longer specified in the RSLThreadMap. Also, support for static registration of the external interface routine (previously called IPCMainLoop) has also been eliminated, as each actor that requires those services should now register their routines dynamically (via the RSLRegisterExternalInterface C Target Services Library function). See the *C Language Guide 5.2* for additional details.

This release also supports a simple and efficient memory allocator, which is particularly effective for target environments which allocate large blocks of memory, even if a small block of memory was requested. See the *C Language Guide 5.2* for details on how to enable this feature. Note that these routines are invoked by the core RSL technology during initialization, and only address memory allocation

performed by the C Target Services Library during initialization (although they may be used at an application level, the user should be aware that there is no ability to “free” memory, once allocated using this mechanism).

All C Target Services Library error messages are generated via a ‘RSLERROR’ call, thus providing users with the ability to more easily identify possible RSL errors and generate alternate handling routines (for example, system logs, and so forth).

The default entry point for the RSL is now called rtsMain, and is encapsulated in a separate file called ‘main.c’. See the *C Language Guide 5.2* and *Porting Guide* for details on how to change the entry point, if required.

Users now have the option of supporting locally-supported thread timers in conjunction with ObjecTime controlled inter-thread message communication (inter-thread actor port bindings), via a new API function ‘RSLRegisterSignallingInterface’. See the *C Language Guide 5.2* for details and examples of how to use this new multi-threaded capability.

User Code changes necessary for new C Target Services Library

The following check list should be used to ensure that all necessary user code changes are made to use the **ObjecTime Developer 5.2** C Target Services Library (see the *C Language Guide 5.2* and/or *Porting Guide* (available from ObjecTime Support), as required, for additional details):

- Check to ensure that the proper compilation settings are created in the toolset configuration browsers (since OBJECTIME_CRTS_HOME and OBJECTIME_CRTS_TARG are no longer supported);
- Check for the string “cRTS” in your updates; if found, theses are likely candidates for modification to newly named “RSL” routines (these are likely to be either C Target Services Library API routines or new RSL types);
- Verify that the default C Target Services Library configuration is suitable for your target. (See “RTConfig.h” on page 199 of the *C Language Guid*e 5.2 and the *Porting Guide* for additional information);
- Check to see if the routine “cRTSRegisterMainloop” was used. If so, change it to “RSLRegisterExternalInterface”;
- Check to see if ‘timeout’ has been specified as the name of the timeout signal in any user-specified code; if so, replace with RSL_Timeout();
- Consider the configuration of the C Target Services Library efficient memory allocation routines;
- Consider eliminating the use of the ROOM_Signal macro interface, as it is no longer required in OTDC 5.2. Also, check to see if any message switching (from one actor port to another) is performed, and if so, it is possible that this code may be simplified;
- Consider changing the definition of RSLError in the core C Target Services Library code to something that may generate an appropriate log/error stream in your target environment;
- Verify that the executable entry point “rtsMain” is suitable for your executable; if not, see the *Porting Guide* for information on how to change the default name.

- Ensure that no signal names match any variable names. The namespace for signal and variable names is the same.

In addition, if the application is multi-threaded,

- Change ‘cRTSThreadMap’ to ‘RSLThreadMap’, and eliminate any un-supported thread specifications, which are now either configured in the toolset, or dynamically registered by the application. See the *C Language Guide* for further information on how to create logical and physical threads and mappings and details on the RSLThreadMap function;
- If supporting local (per-thread) timer implementations, consider using inter-thread port bindings, in conjunction with the “RSLRegisterSignallingInterface” routine.

External debugging

To switch between the Simulation Services Library and the C Target Services Library it is recommended that you employ the LINE mode, which has the same syntax for both. This still entails that they override the `.objectime.debugger.commands` file in your session directory, but at least it is common to both.

The `.objectime.debugger.commands` file for xxgdb under C Target Services Library is as follows:

```
attach "attach %d"
bline  "break %s:%d"
bfunc  "break %s%s"
cont   "cont"
dir    "dir %s"
debug  "xxgdb -command=%s %s"
mode   "line"
```

This allows Target Observability to run with External Debugging.

Problems addressed in this release

For a complete list of problems which have been addressed in this release, please refer to the ObjecTime web site at:

`http://www.objectime.com/support/restricted-dir/index.html.`

You will be prompted to enter your assigned ObjecTime user name and password to gain access.

Chapter 4

General Information

Limits

C TargetRTS Services Library Limits

When compiling for the C Target Services Library there may be only:

- 65534 actor references
- 65534 ports and SAPs
- 65534 threads
- 65534 port references per actor
- 65534 actor classes
- 65534 states in each actor
- 65534 port classes
- 65534 bytes of extended state variable space per actor instance

Special Notes and Reminders

- For an example of how to compile a C model for the Simulation Services Library and for the C Target Services Library, please see “The compilation process” on page 49 of the *C Language Guide*.
- Once an actor has been compiled, modifications to the replication factor of the actor itself or of any ports may cause a recompile of the complete model. We recommend specifying the replication factors as early as possible or editing them in a batched fashion.

Also note that for unspecified replication factors (replication factor = *), if you change the root class of a system, an actor’s previous replication factor will be statically copied over to the new system, and will not take on an intended new value unless explicitly compiled. In this situation, we recommend regenerating the entire model to ensure that the intended replication factor is applied.

Compliance Models

ObjecTime has included a number of models on the distribution media which allow you to test your target port against an ObjecTime developed compliance test suite. This compliance test suite has been used extensively at ObjecTime to verify the reference ports which are shipped with the ObjecTime Developer for C product package and you should take advantage of these models to validate your target port.

The following models can be found in the C Model Examples directory:

- **C_HelloWorld**: This is a simple model helping to confirm the proper configuration, compilation and execution of a model.
- **C_Multithreads**: This model verifies the proper operation of threads for multi-threaded targets.
- **C_General**: This model tests the proper operation of the general modeling features available with ObjecTime Developer which are States, Choice-points, Messages, Functions, Variables, State History, Multiple Port Bindings, SAPs and SPPs and Inheritance Mechanisms. A Raw Speed test is also included in this model to permit the measurement of performance.
- **C_Timers**: This model verifies the correct operation of timers executing on a development platform.
- **C_TornadoQueuesWithTimers**: This model is an example of integrated timers and message queues implementation for the Tornado target.
- **C_FiveStatesA and C_FiveStatesATornado**: These models confirm the operation of the target observability feature.

Additional information can be found in Appendix C - Compliance Suite & Examples in the *C Language Guide*.

User Guide — Differences When Using the C Target Services Library

Not all facilities described in the *User Guide* are supported in the C Target Services Library. Below are the section title and page number for sections or chapters in the *User Guide* where there's a known difference with the C Target Services Library. This is not necessarily a comprehensive list. The following list also includes a brief description of what the difference is, or what other document to refer to for additional information.

For detailed information on some key differences, please see "Differences between SimulationRTS and C TargetRTS" on page 189 of the *C Language Guide*.

- **Package List p. 98**

  "External Data packages" are not supported with the C Target Services Library.

- **Data class list p. 105**

  "Data classes" are not supported with the C Target Services Library.

- **Creating a new data class p. 121**

  "Data classes" are not supported with the C Target Services Library.

- **Listing the system-defined data classes p. 122**

  "Data classes" are not supported with the C Target Services Library. Only the standard Fundamental Types for the C Language are supported with the C Target Services Library.

- **Differences Tests p. 145**

  The various items having to do with "data classes" are not supported with the C Target Services Library.

- **Multiple containment p. 158**

  "Multiple Containment" is not supported with the C Target Services Library.

- **Rules for creating an equivalence p. 160**

  "Equivalences" are not supported with the C Target Services Library.

- **Dynamic Structure p. 161**

  "Dynamic Structure" is not supported with the C Target Services Library.

- **Optional actors p. 161**

  "Optional actors" are not supported with the C Target Services Library.

- **Imported actors p. 161**

  "Imported actors" are not supported with the C Target Services Library.

- **Combining Replication, Dynamic Structure and Multiple Containment p. 163**

  "Dynamic Structure" and "Multiple Containment" are not supported with the C Target Services Library.

- **Optional replicated actors p. 164**

  "Optional replicated actors" are not supported with the C Target Services Library.

- **Imported replicated actors p. 165**

  “Imported replicated actors” are not supported with the C Target Services Library.
- **Including optional and imported actors in equivalences p. 167**

  “optional and imported actors” are not supported with the C Target Services Library.
- **Layered networking communication p. 169**

  “Layered networking communication” is not supported with the C Target Services Library.
- **Actor reference symbols p. 179**

  “Imported actors” and “Optional actors” are not supported with the C Target Services Library.
- **Actor Reference Properties Editor p. 186**

  The “Fixed, Optional, Imported” item is not supported with the C Target Services Library.All actors are Fixed.
- **Actor reference list item menu p. 192**

  The various items having to do with “Equivalences” are not supported with the C Target Services Library.
- **Creating an equivalence (multiple containment) p. 198**

  “Equivalences” and “Multiple Containment” are not supported with the C Target Services Library.
- **Making an imported actor p. 198**

  “Imported actors” are not supported with the C Target Services Library.
- **Data Class p. 202**

  “Data Classes” are not supported with the C Target Services Library.
- **Signal list p. 203**

  Signal “Data Classes” are not supported with the C Target Services Library.
- **Log Service p. 223**

  The “Log Service” is not supported with the C Target Services Library.
- **Frame Service p. 223**

  The “Frame Service” is not supported with the C Target Services Library.
- **Basic Communication Service p. 225**

  “Synchronous communication” is not supported with the C Target Services Library.
- **Replying to messages p. 226**

  This is not supported with the C Target Services Library.
- **Name registration and binding p. 227**

  “Name registration” is manual rather than automatic in the C Target Services Library.
- **Simulation Communication Service p. 230**

  The “Simulation Communication Service” is not supported with the C Target Services Library.
- **Timing Service p. 231**

  The “Timing Service” is implemented quite differently in the C Target Services Library. Please see the C Language Guide for details.

- **Exception Service p. 233**

  The “Exception Service” is not supported with the C Target Services Library.

- **Variables Window p. 252**

  The information about “Data Classes” is not supported with the C Target Services Library.

- **Data Modeling p. 273**

  The information in this chapter is not supported with the C Target Services Library.

- **Data Class Browser & Editor p. 283**

  The information in this chapter is not supported with the C Target Services Library.

- **MSC Editor p. 337**

  Design-time “MSCs” are a useful way of specifying requirements for any system. Run-time MSCs can be captured from within the Simulation Services Library but are not supported with the C Target Services Library.

- **MSC Tool Palette p. 340**

  The “Creation Tool” and “Stop Tool” are not supported with models that will run in the C Target Services Library.

- **Creations/Destructions p. 342**

  “Creations/Destructions” are not supported with models that will run in the C Target Services Library.

- **Message Sequence Charts p. 349**

  “Message Sequence Charts” are only available when running C models in the Simulation RTS.

Chapter 5

Troubleshooting

This section lists common problems and errors encountered when installing and running ObjecTime Developer. With the description of the problem is the suggested course of action required to overcome the problem. Refer to the *ObjecTime Developer 5.2.1 Getting Started Guide & Release Notice* for additional troubleshooting information.

- **Recompiling (after deleting a Load-Build Path)**

  When you compile an update with a load-build path and then delete the load-build path and recompile, a make error occurs, since the .dep files do not have paths hard-coded in them.

  In order to remove previous compilation results you should manually remove all previous compilation results after all load-build changes. From your Update Root Directory, the Unix shell command is:

```
rm -R LF build C++ Makefile
```

Compilation problems — Windows NT

- **Compile fails on valid C model for Simulation Services Library with Microsoft Visual C++**

  If Microsoft Visual C++ is installed at the default location in \Program Files\DevStudio, which contains spaces in the path name, then ObjecTime will not find it. ObjecTime Developer 5.2.1 does not support embedded spaces in the directory names. Reinstall Microsoft Visual C++ in the directory without spaces (for example, \DevStudio)

  The INCLUDE and LIB environment variables may not be properly set. Start “ObjecTime Developer Command Prompt” from “ObjecTime Developer 5.2.1” group in the Start Menu and run the “set” command. Ensure, that your compiler binaries are on the path and INCLUDE and LIB environment variables are set. (for example, they could be set for the user, who installed Microsoft Visual C++, but not set for another user). Set the environment variables. Refer to the Microsoft Visual C++ documentation for further details.

- **Error loading Actor (“could not spawn process”)**

  If the executable (Actor.exe) is stored on an NFS server then the NFS client must be configured to have execute permission set.

Appendix A

Developer for C Directory Contents

After installation of the main ObjecTime files has been completed, the directory structure should be as follows. Please ensure that the <INSTALL> directory and all its files are readable, and not writable, by all users of ObjecTime. The `Developer5.2.1` directory and its sub-directories contain all the individual files that comprise the particular release. Some of the files and directories included by the C Target Module are as follows:

- `<INSTALL>/Developer5.2.1`

  This is the top level directory.

- `C`

  This directory contains all of the source code, makefiles and other files required by the C Target Services Library (cRSL).

- `Help/C`

  This directory contains C documentation files used for the Online Help System.

- `ModelExamples/C`

  This directory contains ObjecTime C models referenced at various locations in the documentation.

- `Training/BasicTutorials/BasicTutorialC`

  This directory contains ObjecTime C model examples for Basic Tutorial.

Appendix B

Known Limitations/Restrictions

- You cannot delete files when they are currently open or being observed in the Explorer. This impacts how ObjecTime code generation and possibly other subsystems work. An error message is returned if the update/C directory is opened.
- In the Simulation Services Library, the ROOM_SAPRegister() and ROOM_SPPRegister() macros always return non-zero. In the C Target Services Library, the return code indicates only that the registration was performed, and not that a SAP/SPP pair was actually bound.
- The environment variable OBJECTIME_MAX_VFORKS no longer has any effect.
- Layer SPPs in C actors must be specified with a maximum replication factor, equal to the maximum number of SAPs bound to it. An *unspecified* replication factor in this case equates to “0”.
- Changing permissions, deleting files, or using symbolic links to ObjecTime C generated directories or files can cause ambiguous errors.
- System header files such as stdio.h are not automatically included. Due to this, expressions for Choice Point and Transition Guard conditions should return an int (for example, 0 or 1). To return TRUE or FALSE, user must explicitly include the header files containing the definition of TRUE and FALSE.
- There is a difference in behavior between the Simulation and Target services library for isTimerValid. The target system reported correctly false when the timer was invalid. Simulation reported true in this case. (PR 7468)
- Models using timer actors may report the following warning at the end of the compilation - “warning: passing arg 2 of 'RSLRegisterExternalInterface' from incompatible pointer type”. Please ignore this warning message. (PR 7936)
- To trace on a replicated port within the C TargetRTS Debugger you must follow the outlined procedure.

    - List the different ports/saps/spp with "info <actor>"
    - The port identifier number required for the trace command is the base number given by the info command plus the index of the replicated port or SPP you wish to trace (port = Base_Port + port/SPP Index)
    - For example if you want to trace ids[5] of an actor which lists the following ports after the info command:

      ```
      End ports:
      ```

```
0: ThePort
1: MyPort
2: ids[13]
```

- To trace the fifth occurrence of ids, you would have to enter the following command:

```
trace 1 end 7 (Base_Port is 2 and port/SPP Index is 5)
```

Inclusion Paths

- Use absolute inclusion paths (as opposed to relative inclusion paths) as the results from using relative inclusion paths can be inconsistent and in some cases will simply not work.

Solaris Multi-threaded

- Solaris threads support priorities but the priority specified in the cRSLThreadMap for solaris multi-threaded is ignored, that is, it is not applied to the created thread. Although the meanings of priorities with respect to other processes, and bound/unbound LWP is somewhat confusing, the priority should be applied. Solaris does not support thread priorities. Thread priorities set in the toolset are ignored in Solaris threads (Solaris only supports thread priorities with POSIX threads). (PR 5371)

Relay Ports

- The C Target Services Library Debugger Info command does not list the Relay Port information. For relay ports, N/A will be displayed, as opposed to the actual value. (PR 7866)

Target Observability

- The limit on the number of actor instances that can be displayed in the Run-Time System panel is approximately 1,000 actors. If the model to be run on target has more than this number of actor instances within the first two levels of actor hierarchy, then the model will not be loadable with Target Observability.

Tornado

- When you reset a Tornado board, the target operating system is also reloaded. This means that all applications will be reloaded whether or not they are directly related to your ObjecTime session. A reset can occur by pressing the Reset button in the Target Services Library, or by exiting from the Target Services Library. ObjecTime recommends using manual mode to prevent the automatic reset if extensive board setup is required. (PR1649)

Known Problem Information

For a complete list of known problems in this release, please refer to the ObjecTime web site at:

`http://www.objectime.com/support/restricted-dir/index.html.`

You will be prompted to enter your assigned ObjecTime user name and password to gain access.

Appendix C

Tornado Integration on NT

This appendix is added for reference and outlines detailed information on the integration of Tornado with ObjecTime Developer. Since the syntax is the same with regards to Tornado integration when designing an ObjecTime Developer model using either the C or C++ language, the references to the C++ language can be disregarded, and in their place, substitute the C language settings.

Using Tornado on Windows NT

This section provides an introduction to using Tornado on Windows NT, including

- a discussion of the environment setup, along with some tests to prove that all the pieces are working
- a description of the configuration used at ObjecTime Limited
- an overview of the tools used with Tornado

Environment setup

Assumptions

The instructions in this report assume that Tornado and ObjecTime are installed in their default locations, which are `C:\Tornado` and `C:\ObjecTime\Developer5.2.1` respectively. It also assumes that ObjecTime is working properly. If you have installed either package in a different location, you will have to adapt the instructions in this report accordingly. As well, this report assumes you are developing for a Motorola 68040 platform. Of course, Tornado VxWorks also has tools for the x86, and PowerPC architectures. Since all of the instructions here work equally well for those targets, it should be a simple matter to adapt them as appropriate.

Setting Your Environment Variables

WindRiver's *Installation Guide* suggests that there is no need to set any environment variables when you install Tornado on NT[1]. However, ObjecTime Developer needs to know enough about your Tornado installation to properly invoke the make utility and the Tornado cross-compiler to compile your model. ObjecTime Developer also needs to know the location of the target server registry in order for Target Observability to work. For ObjecTime to work correctly with the WindRiver development tools, the following NT environment variables must be properly set:

1. *WindRiver Products Installation Guide, Tornado 1.0.1*, 1997. p. 42.

```
WIND_BASE=C:\Tornado
WIND_HOST_TYPE=x86-win32
GCC_EXEC_PREFIX=%WIND_BASE%\host\%WIND_HOST_TYPE%\lib\gcc-lib\
PATH=.;C:\WINNT\system32;C:\WINNT;%WIND_BASE%\host\%WIND_HOST_TYPE%\bin
WIND_REGISTRY=<host_name>
```

The first three variables can be found in the `torVars.bat` batch file located in the `C:\Tornado\host\x86-win32\bin` directory. The meaning of these variables is as follows:

`WIND_BASE` —installation directory for Tornado

`WIND_HOST_TYPE` —name of host type

`GCC_EXEC_PREFIX` —location of GNU compiler subprograms

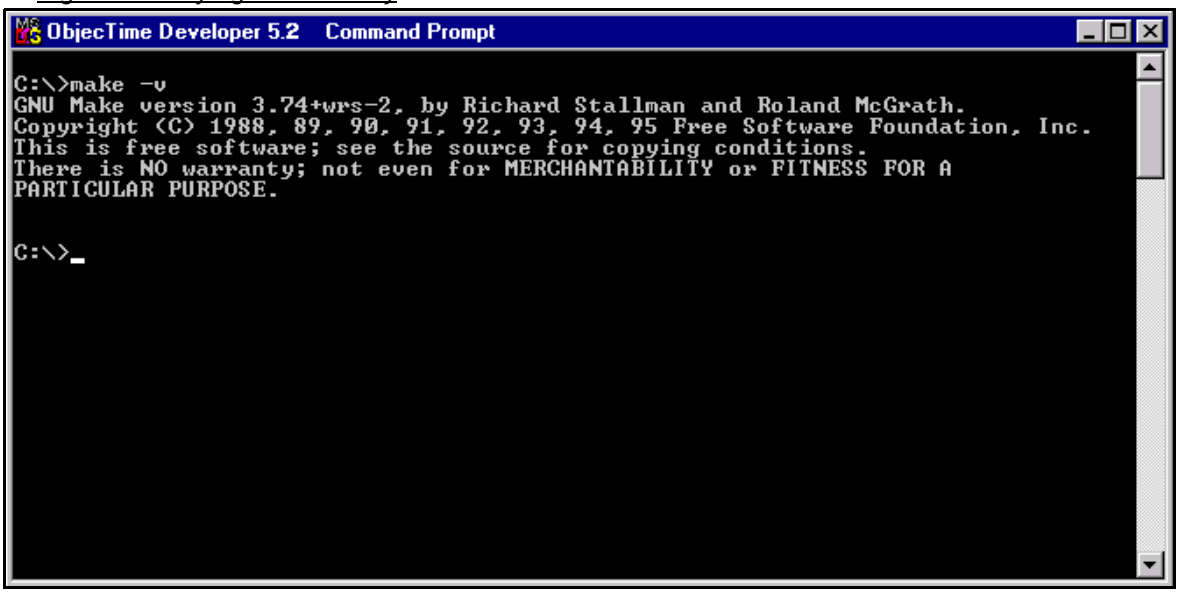
`WIND_REGISTRY` —name of the registry host

Note that the trailing backslash is important in the value of GCC_EXEC_PREFIX. The compiler driver, `cc68k`, does not treat the variable as a directory. Instead it uses the text literally as a prefix. If the backslash is missing, or if the value is incorrect, compilation will fail, usually with the preprocessor[2].

Of course, your PATH variable will contain additional information depending on other software packages installed on your machine. For the most part, the path should not be a concern. However, if you have installed other development utilities, such as another version of '`make`', there may be a conflict. The safest thing to do is open the ObjecTime Developer Command Prompt and verify that the make utility is the version shipped by WindRiver. From a Command Prompt type '`make -v`' and insure that the version is reported as '`3.74+wrs-2`'. Your version number might vary slightly. Figure 4 shows the technique. Note that it is recommended that you make use of the ObjecTime Developer Command Prompt because the correct path to the Developer 5.2.1 binaries is automatically appended to the PATH variable.

2. *User's Guide: Tornado 1.0* (Unix Version), 1995. p. 56.

Figure 4 Verifying ‘make’ utility



Finally, if you are running the Tornado registry on a machine other than your workstation, you will have to set the value of the WIND_REGISTRY environment variable to be the host name of the machine where the registry is running. This variable is required even if you have changed the value of the registry using Tornado’s *Registry&License* utility. Please refer to section 3.2 of the WindRiver *User’s Guide* for more details on setting up a target server registry.

Testing the Cross Compiler

This section presents a method for establishing that the components of your tool-web are functioning properly. The point to bear in mind throughout this section is that compiling your ObjecTime model is no different than compiling any other source code. If you have not successfully compiled any source code at all using the Tornado cross-compiler for your intended target, then you will have to resolve these problems before attempting to compile your ObjecTime model. As discussed, ObjecTime expects the `make` utility to be able to invoke the proper cross-compiler for your target platform. If the compiler and related tools are not available from the command prompt, or if they are not functioning properly, then there is nothing that the toolset can do except tell you the compile failed. Please refer to your WindRiver documentation, or contact the WindRiver support group to resolve any compiler problems you encounter.

For the purposes of the following test, it is assumed that you have set your environment variables properly and that you are using the ObjecTime Developer Command Prompt. First, save the code below into some directory on your working drive. For this example, let us assume that the file is called `C:\mydir\hello.c`

```
#include <stdio.h>
void rtsMain()
{
```

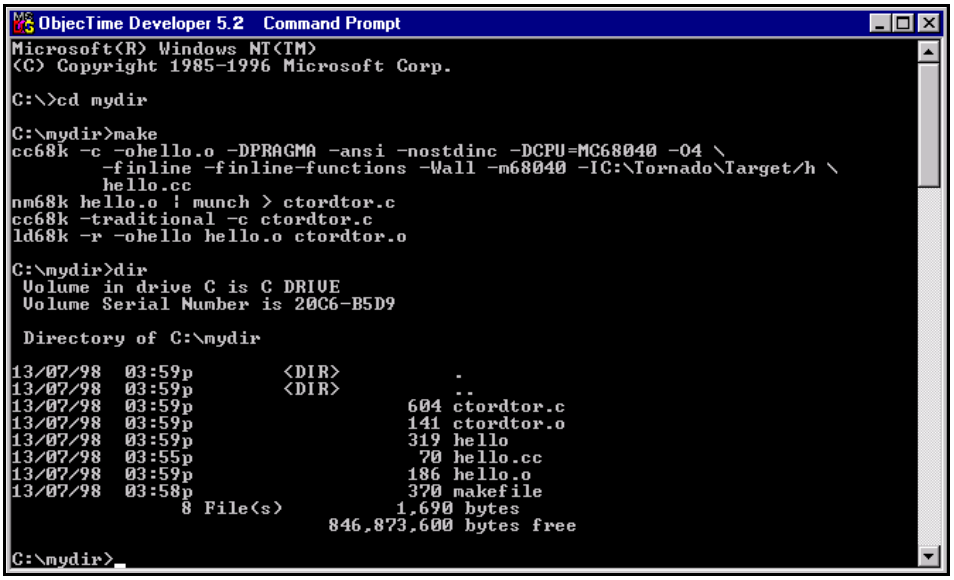
```
printf('Hello World\n');
}
```

Next, save the `makefile` below in your working directory. For this example, the file is called `C:\mydir\makefile`

```
hello : hello.o ctordtor.o
	ld68k -r -ohello hello.o ctordtor.o

hello.o : hello.c
	cc68k -c -ohello.o -DPRAGMA -ansi -nostdinc \
	-DCPU=MC68040 –O4 -finline -finline-functions -Wall \
	-m68040 -IC:\Tornado\Target/h hello.c
```

From the ObjecTime Developer Command Prompt change directories to your working directory and type `'make'`. The figure below shows the expected results of building this "Hello World" program.



If these steps are successful, then you can be reasonably certain that your ObjecTime model will compile. ObjecTime Developer should be able to run the `make` utility, which in turn will invoke the necessary executables to compile your model for VxWorks using the ObjecTime generated `makefile`. If any of these steps failed, then you should contact your system administrator or WindRiver support contact to determine what is wrong. For an explanation of these steps, please see the next section.

Cross Platform Development

The sample `makefile` in the previous section was not explained in detail. However, it contained the exact dependency rules that ObjecTime will use to build your model into a VxWorks application module. This section will explain some of the compiler flags used in the `makefile`. If you have already

done cross-platform development in Tornado, then feel free to skip this section entirely. If, after this section, you still have some questions, please consult the appropriate WindRiver manuals for details. The *GNU Toolkit User's Guide* and *User's Guide* are good places to start your exploration of this subject.

Compiling an Application Module

The first step in building your model into an application module for VxWorks is to compile the ObjecTime generated source code into object code. To do so, the make utility invokes the proper cross compiler driver for your target. In this case, it is the **cc68k** compiler. The compiler flags used will differ from platform to platform. To compile your model for the 68040 target, the compiler invocation is as follows:

```
cc68k -c -ohello.o -DPRAGMA -ansi -nostdinc -DCPU=MC68040 \
    -O4 -finline -finline-functions -Wall -m68040 \
    -IC:\Tornado\Target/h hello.c
```

The explanation of the flags is found in Table 1.

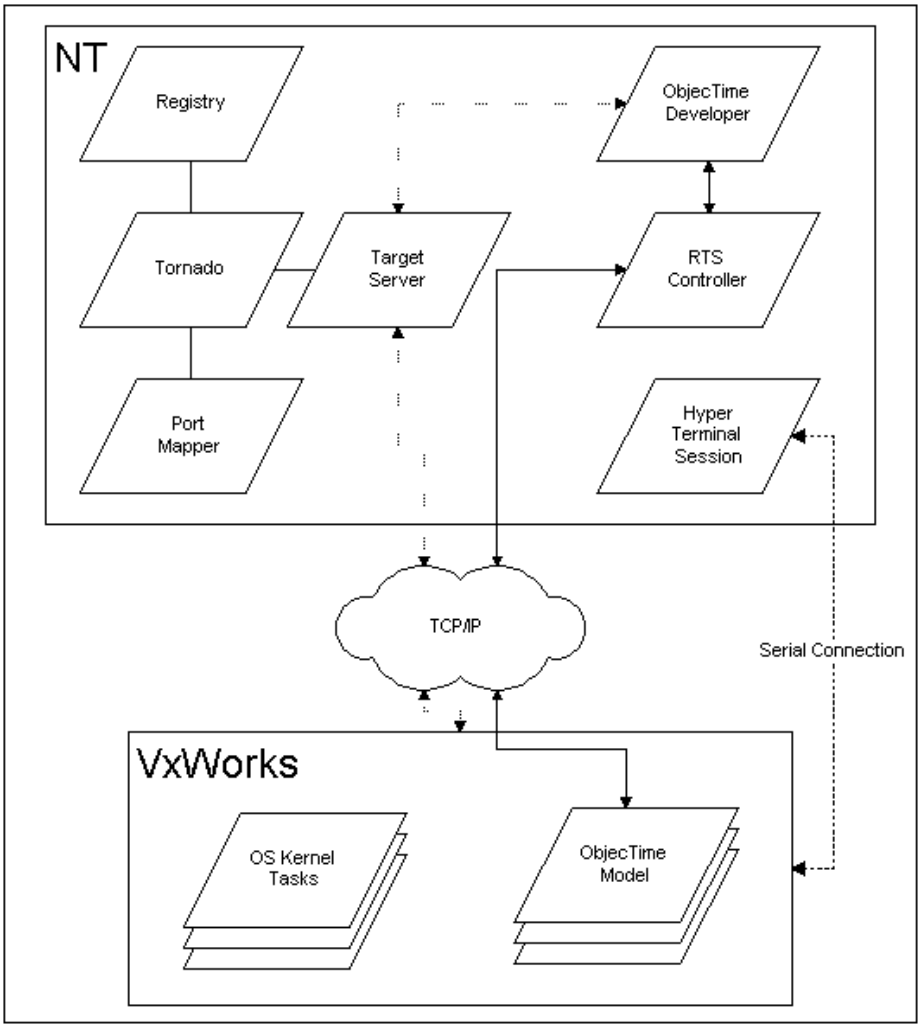
Table 1 Compiler Flags

Flag	Description
-c	Compile only; do not link for execution under the host. The output is an unlinked object module with the suffix ".o".
-o `file`	Place the output in file, in this case **hello.o** (not strictly necessary in our example).
-DPRAGMA	Define the symbol PRAGMA for the build.
-ansi	Support all ANSI standard C programs. This option turns off certain features of the GNU compiler that are incompatible with ANSI C.
-nostdinc	Do not search host-system header files; search only the directories specified with the -I flag and the current directory header files.
-I `path`	Include VxWorks header files.
-O4	Perform all optimizations.
-finline	Pay attention to the inline keyword.
-finline-functions	Integrate all simple functions into their callers.
-Wall	Turn on all compiler warnings.
-DCPU=`arch`	Define the CPU-type.
-m68040	Generate output for 68040. Inhibits use of 68881/82 instructions that have to be emulated on 68040.

Overview of the ObjecTime/Tornado integration tools

ObjecTime Developer and WindRiver's Tornado are closely integrated. In fact, the toolset can automatically download your model to a VxWorks target and spawn it. Further, one can watch the execution of a model through our Target Observability feature as it is running on the target. In order to implement this integration though, a number of helper applications are required. These applications will vary depending on the workstation OS and development environment you are working in. Figure 5 shows the helper applications that are required for the client's workstation and development environment. Please note that *all* of the Tornado components need to be running and functioning properly for ObjecTime Developer to automatically download and run models on a VxWorks target.

Figure 5 ObjecTime/Tornado tools



What Tornado Needs

To run Tornado on Windows NT, you must have a portmapper and target registry[3] running. Additionally, in most development environments, a serial connection between the developer's workstation and the target is also required. This is used to bootstrap the target board and to receive standard out and standard error. When working on Windows NT, the 'HyperTerminal' application is most often used to establish the serial connection. Finally, WindRiver hosts all development tools on the workstation to reduce

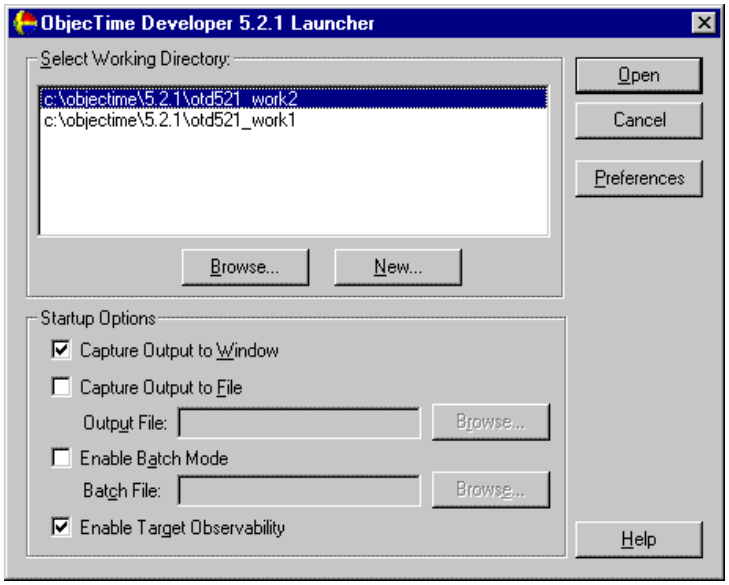
3. The registry does not have to be located on the developer's workstation. It can be running anywhere on the network, but only one registry can be used per session. Note too, that the WIND_REGISTRY variable must be properly configured if you are running the registry on another host.

the load on the VxWorks target. In order to accomplish this, a target server is required. The target server allows tools like the WindShell and CrossWinds debugger to run on the host workstation and communicate with the target.

What ObjecTime Needs

Like Tornado, ObjecTime employs a communication agent to enable the toolset to communicate with a model that is running as a stand-alone executable. This agent is called the 'rtsController'. You start the rtsController when you launch ObjecTime Developer with Target Observability enabled. Figure 6 shows how to start the rtsController when ObjecTime Developer is launched. When the ObjecTime model is loaded onto the VxWorks target and spawned, it will attempt to communicate with the rtsController running on the host workstation. As well, ObjecTime Developer needs to be able to communicate with the target board in order to download the model to the target. Thus, there has to be a target server running and the toolset needs to know the name of the target server.

Figure 6 Starting the RTSController



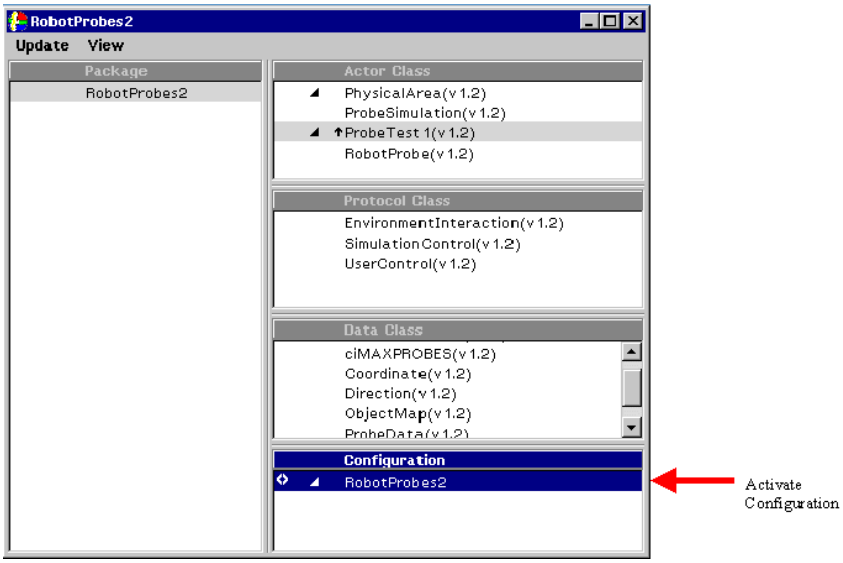
Tornado Integration

ObjecTime Developer 5.1.1 provided basic support for WindRiver Tornado integration on Windows NT. It provided users with Target Observability features such as loading, executing, monitoring and resetting models on a given Tornado target. In **ObjecTime Developer 5.2.1**, users are able to set and clear source line breakpoints. Users are notified when a breakpoint hit has been detected and given the appropriate steps to configure the Tornado debugger.

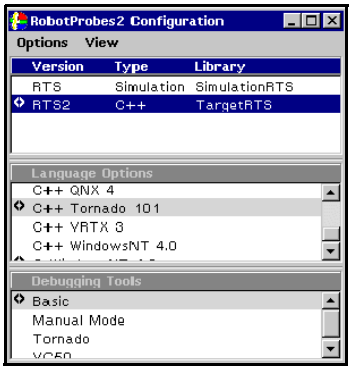
Configuration

The environment must be configured properly in order to use the Tornado source code breakpoint capability. To configure the environment, follow these steps:

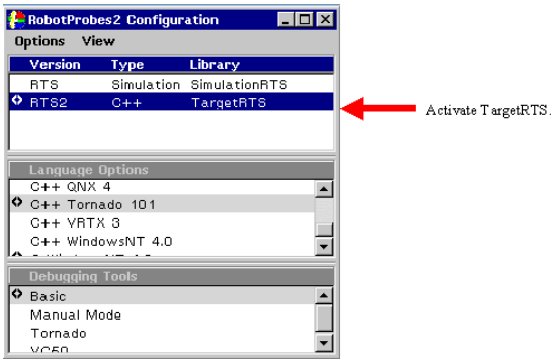
1 Activate the desired configuration from the Configuration menu of the Update Browser.



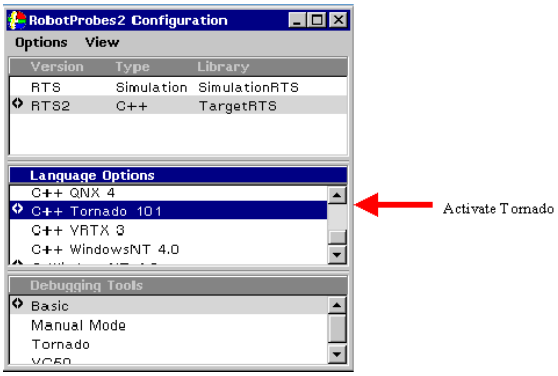
2 Open the Configuration Browser for the selected configuration.



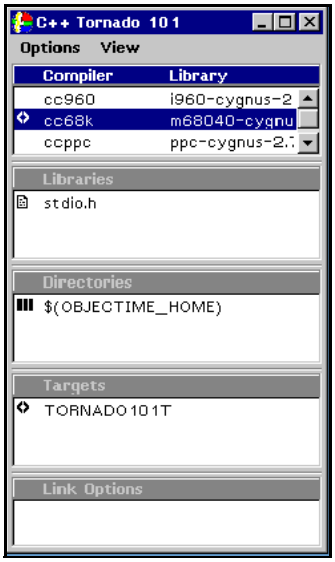
3 Activate the Target RTS entry in RTS Versions menu.



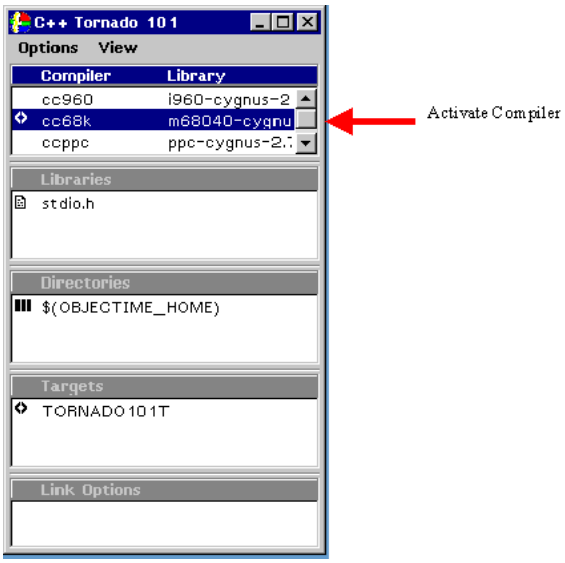
4 Activate the Tornado Language Option menu item.



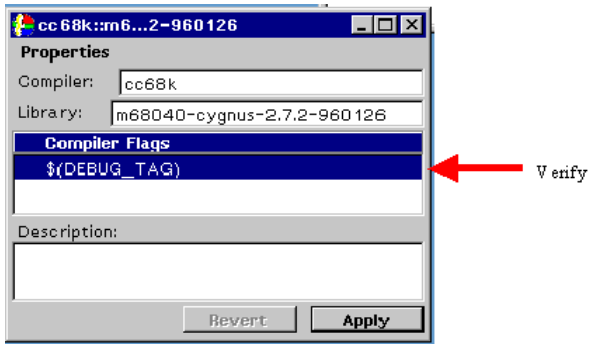
5 Open the Tornado Language Option Browser.



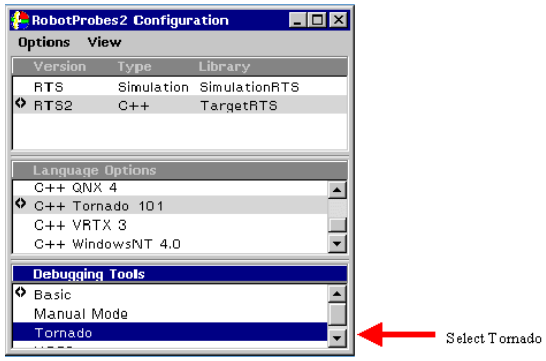
6 Activate the desired Tornado compilation settings for the target.



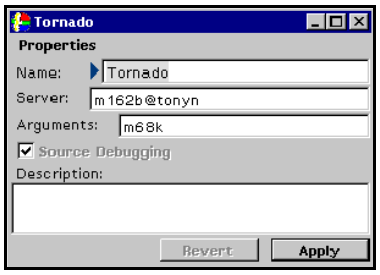
7 Open the Properties for the selected item in the Compiler menu and verify that the debug flag is set for compilation in the Compiler Flags.



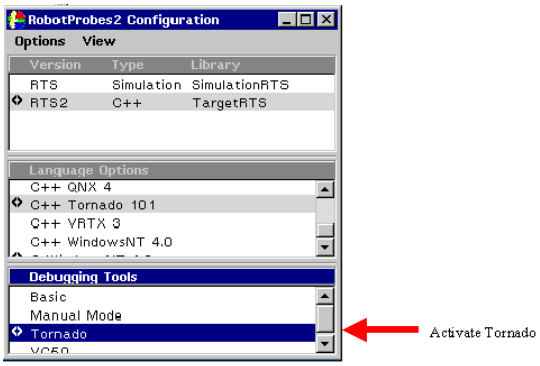
8 Go back to the selected Configuration Browser and select Tornado from the Debugging Tools menu.



9 Open the property editor for the Tornado entry; specify the desired Tornado target server and the target processor in the Server and Arguments fields, respectively.

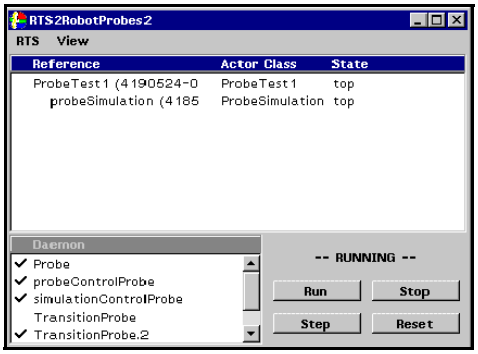


10 Activate the Tornado entry in the Debugging Tools menu of the Configuration Browser.

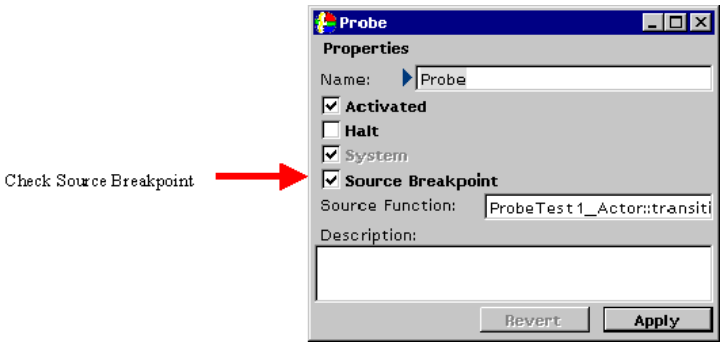


Source Breakpoints

ObjecTime Developer provides access to source line breakpoints through Transition and State Daemons. To set a source breakpoint, Target Observability must be running. Invoke the Load item from the Compile menu of the Update Browser.



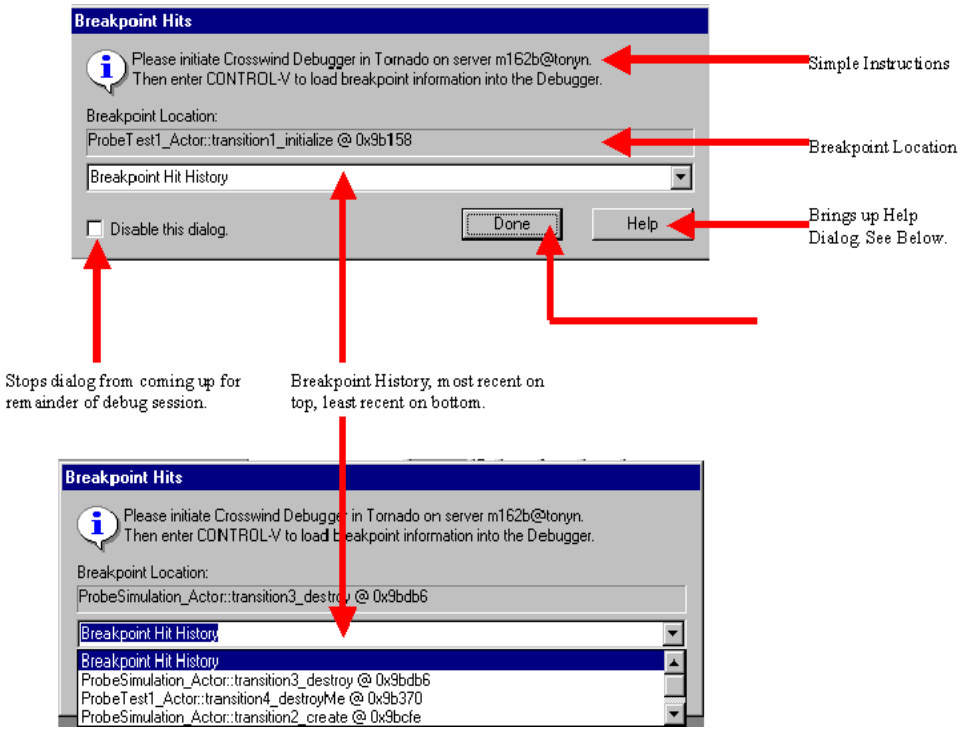
Daemons can now be created and activated. To set a source line breakpoint, open the Properties Editor for the daemon of interest. Check the “Source Breakpoint” check box and make sure the daemon is activated.



Note: Only Transition Daemons that are located at the start of a transition have the “Source Breakpoint” check box enabled. Similarly, only State Daemons with entry-code have the “Source Breakpoint” check box enabled.

Source Breakpoint Hit

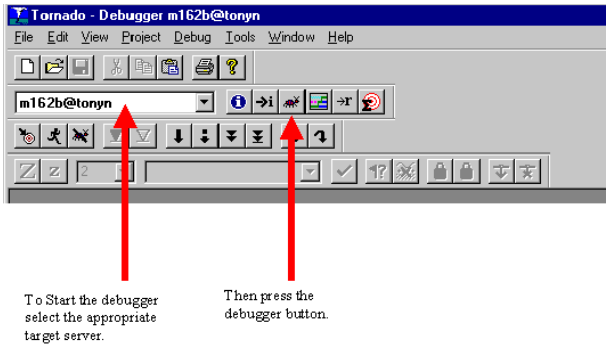
When ObjecTime Developer detects that a breakpoint hit, the following dialog is displayed.



The first time a breakpoint is hit, the user should follow the these steps:

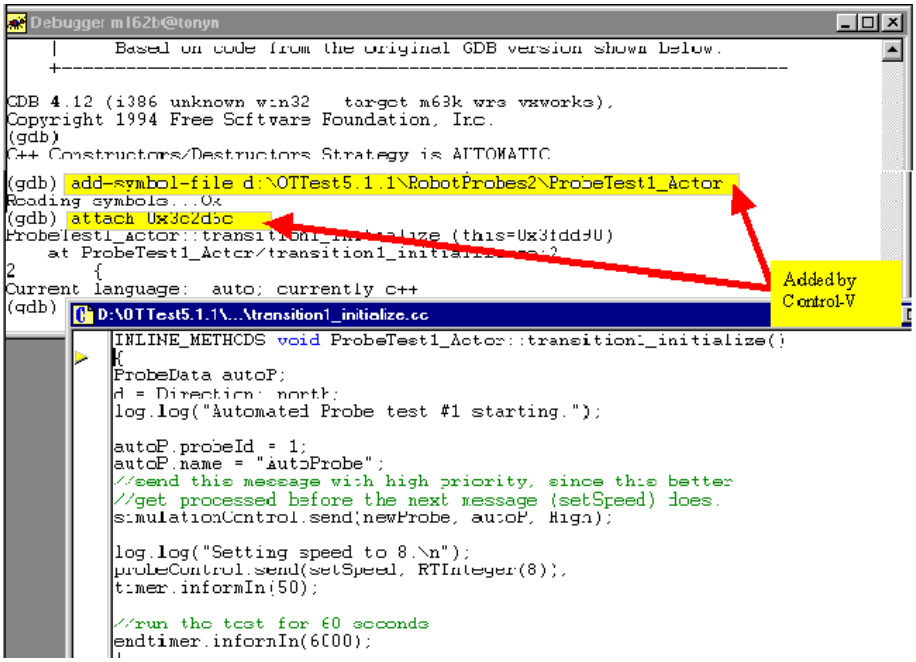
1 Bring up Tornado if not currently running.

2 Select the target server specified in the Breakpoint Hit dialog and invoke the debugger within Tornado.



3 Enter CONTROL-V (Windows paste operation).

The appropriate “add-symbol-file” and “attach” commands are copied into the Windows clipboard by ObjecTime Developer.



At this point, the debugger displays the file in which the breakpoint has occurred. Regular source line debugging can take place. The Breakpoint Hit dialog can remain opened or can be closed by clicking the Done button. The next time a breakpoint occurs, the dialog reappears if it was closed or the informa-

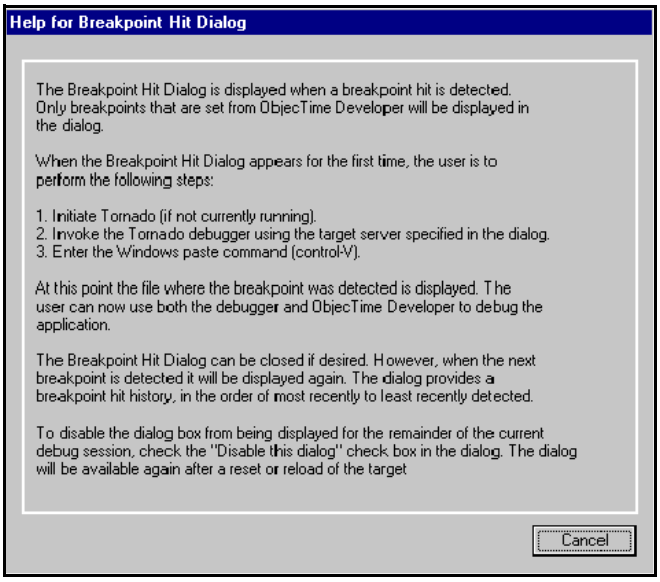
tion is updated if it was opened. The user can disable the breakpoint dialog from appearing during the current debug session by checking the “Disable the dialog” check box in the dialog and clicking Done. The dialog is re-enabled when the target is reset.

Note: Only breakpoints that are set via ObjecTime Developer are displayed in this dialog.

Once a breakpoint is hit, the user must use the Tornado debugger to continue model execution.

Help

Help information is available when the user clicks the Help button on the Breakpoint Hit Dialog. The help information is displayed in a simple dialog.



Appendix D

Integrating Developer Studio on Windows NT

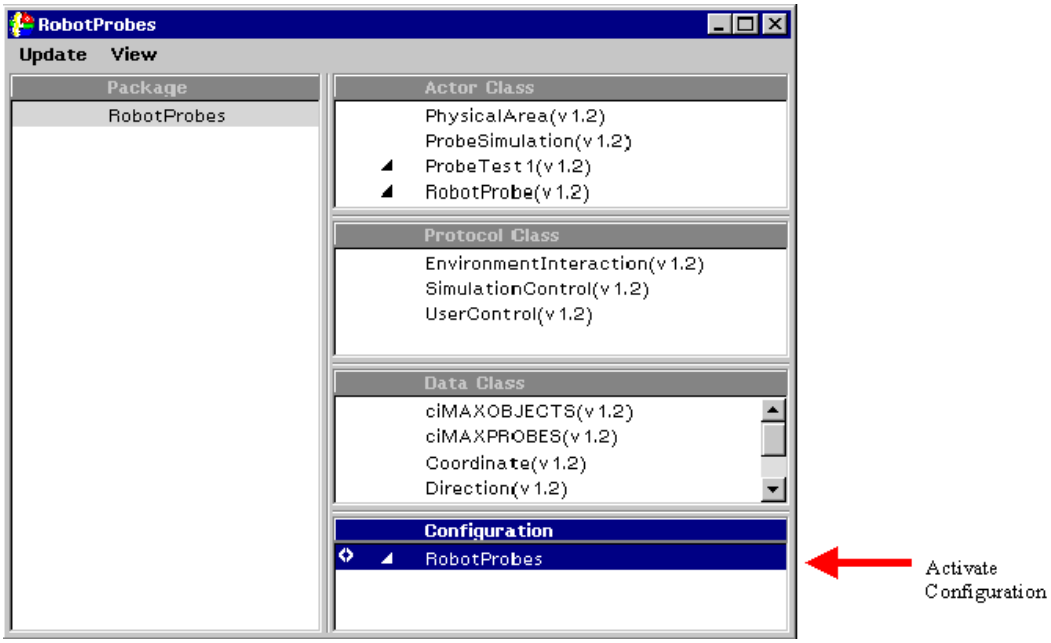
Overview

ObjecTime Developer 5.1.1 introduced support for Microsoft MSDEV integration. This document describes how a user is to configure **ObjecTime Developer 5.2.1** to use the MSDEV source code breakpoint capability. Since the syntax is the same with regards to Developer Studio integration when designing an ObjecTime Developer model using either the C or C++ language, the references to the C++ language can be disregarded and in their place, substitute the C language settings.

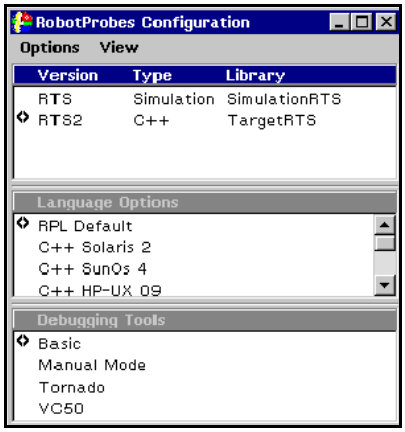
Configuration

The ObjecTime Developer environment must be configured properly in order to use the MSDEV source code breakpoint capability. To configure the environment, follow these steps:

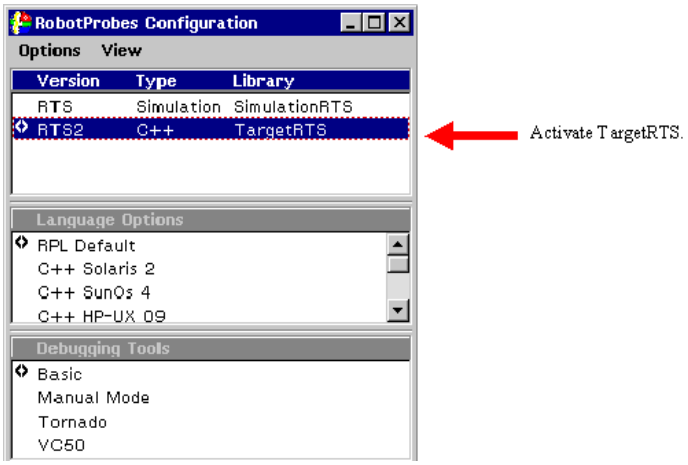
1 Activate the desired configuration from the Configuration menu of the Update Browser.



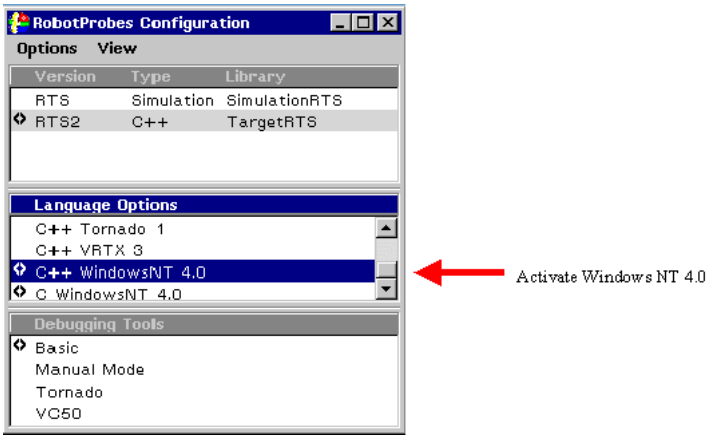
2 Open the Configuration Browser for the selected configuration.



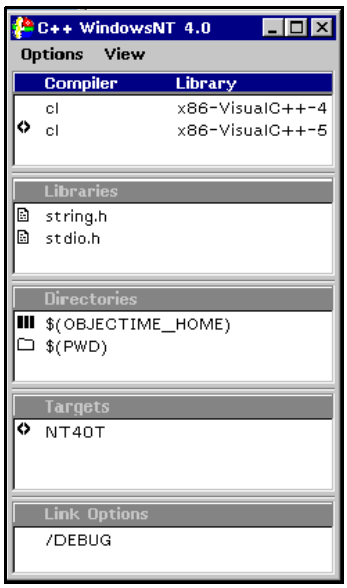
3 Activate the Target RTS entry in RTS Versions menu.



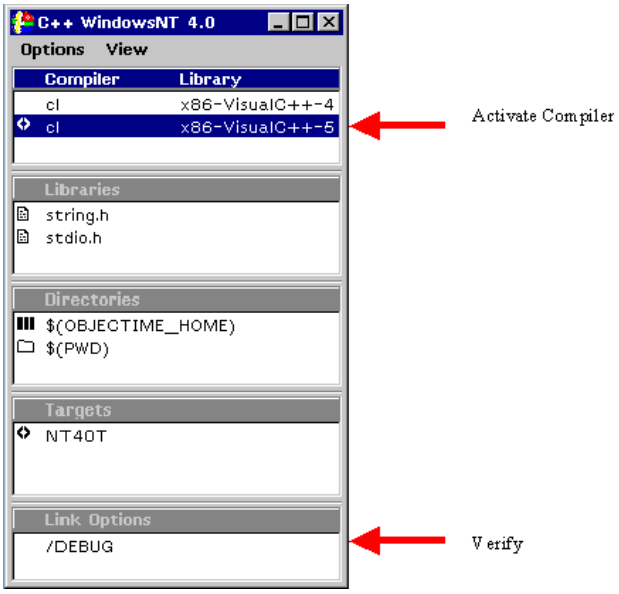
4 Activate the Windows NT C++ Language Option menu item.



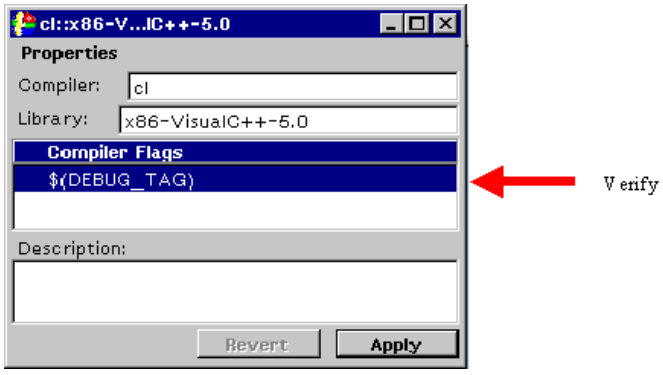
5 Open the Windows NT C++ Language Option Browser.



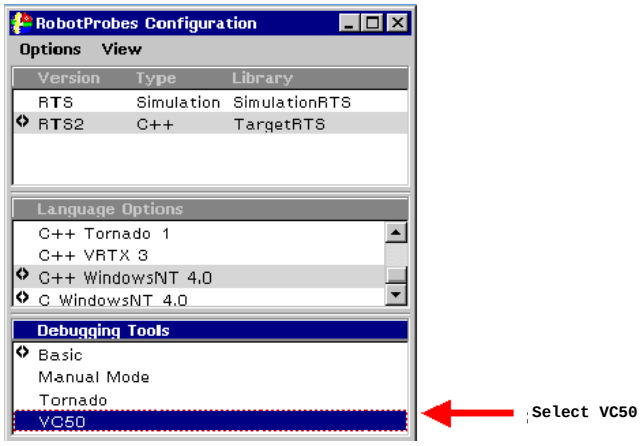
6 Activate the desired Visual C++ compilation settings for the target. Verify that the /DEBUG link option is set.



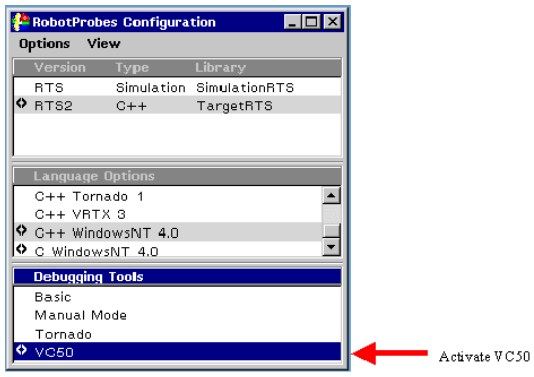
7 Open the Properties for the selected item in the Compiler menu. Verify that the debug flag is set for compilation in the Compiler Flags.



8 Go back to the selected Configuration Browser and select VC50 from the Debugging Tools menu.

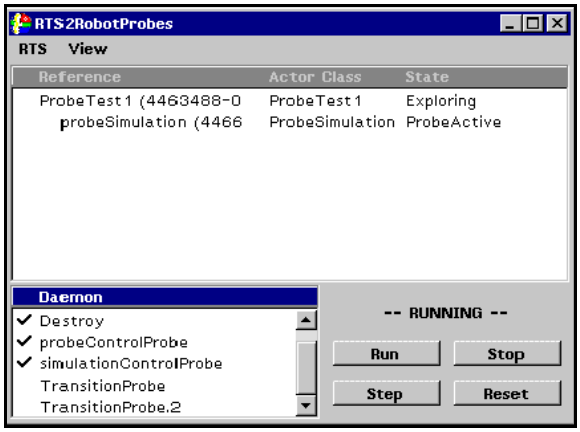


9 Activate the VC50 entry in the Debugging Tools menu of the Configuration browser.



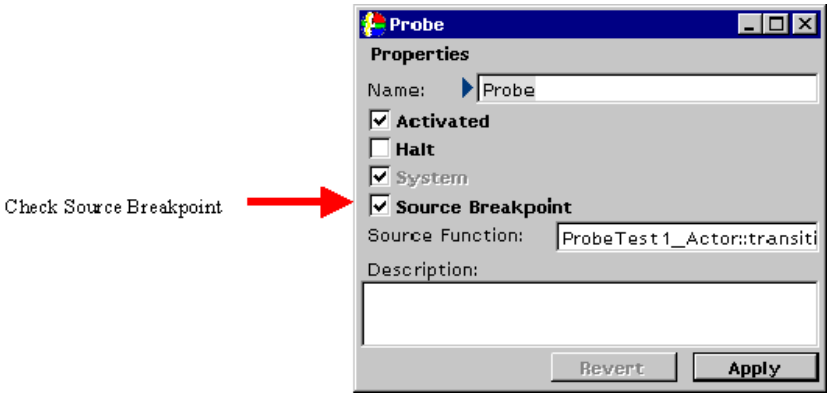
Source Breakpoints

ObjecTime Developer provides access to source line breakpoints through Transition and State Daemons. To set a source breakpoint, Target Observability must be running. Invoke the **Load** item from the Compile menu of the Update Browser.



Note: MSDEV is brought up on the first load for a given update and remains up until the VC50 debugger option is deactivated or the session has terminated.

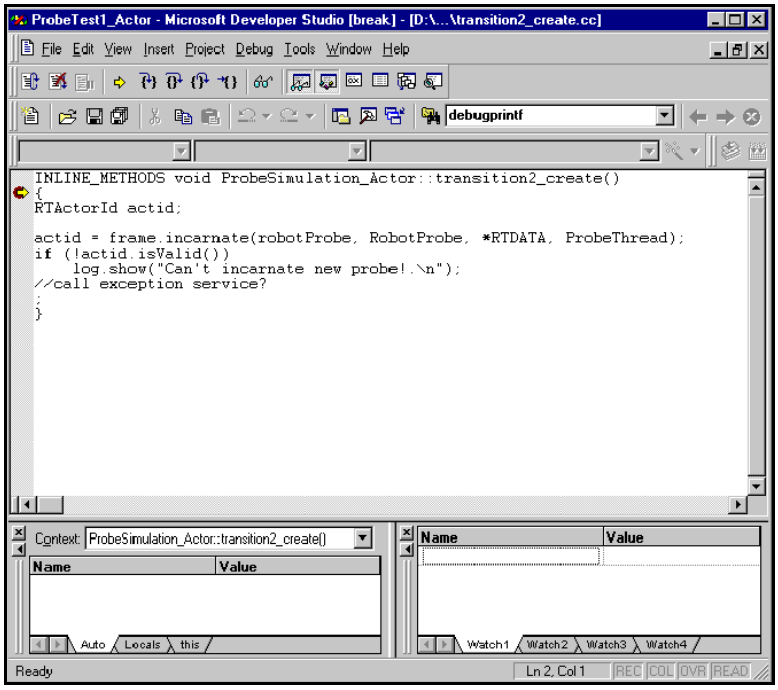
Daemons can now be created and activated. To set a source line breakpoint, open the Properties Editor for the daemon of interest. Check the “Source Breakpoint” check box and make sure the daemon is activated.



Note: Only Transition Daemons that are located at the start of a transition will have the “Source Breakpoint” check box enabled. Similarly, only State Daemons with entry-code will have the “Source Breakpoint” check box enabled.

Source Breakpoint Hit

When a source line breakpoint is hit, MSDEV will pop to the front and display the source code corresponding to the breakpoint. The user can now use MSDEV and ObjecTime Developer to debug their model. Note that once a breakpoint is hit, the user must use MSDEV to continue model execution.



OBJECTIME®
www.objectime.com

ObjecTime Limited
340 March Road
Kanata, Ontario
Canada K2K 2E4