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-DOC809

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Unpublished -- rights reserved under all Copyright laws including Copyright laws of the United States.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 © 1989-1998 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.

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

Chapter 1

Welcome to ObjecTime Developer for C++ 5.2.1/5.2

Introduction

The **ObjecTime Developer for C++ 5.2.1** release builds on the capabilities introduced in the previous releases to meet your large project scalability needs. You can now streamline your loadbuild efforts by building executables directly from the ClearCase CM system, and maximize your team's productivity by using advanced ClearCase features such as wink-in, and synchronization with views. This release also introduces support for Visual C++ 6.0 for the Windows NT platform, as well as support for Lynx 3.0, and Enea OSE 3.1.

ObjecTime Developer for C++ 5.2.1 also introduces support for several new and updated target platforms, including Lynx 3.0 and OSE 3.1.

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 five 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 highlights some of the key new features available in **ObjecTime Developer 5.2.1**:

- **ClearCase Support Enhancement**: C++ users of ClearCase can take advantage of the following when generating for the target:
  - **External builds**: ClearMake can now build directly from the ClearCase CM system, eliminating the need to bring up the toolset for compilation of artifacts already in the library.
  - **Wink-in**: ClearCase wink-in capabilities can now be used to achieve optimal reuse of the loadbuild artifacts.

  See "ClearCase Support Enhancements" on page 79 of the ObjecTime Developer 5.2.1 (Base) *Getting Started Guide and Release Notice*.
- **New target platforms**: You can now use ObjecTime Developer C++ with Lynx OS 3.0.1 as well as Enea OSE 3.1. For full details see "Supported Reference Platforms" on page 9.
- **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" on page 14 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 *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 ObjecTime Developer 5.2.1 for details about Developer TestScope's capabilities.

The following highlights some of the key new features made available with **ObjecTime Developer 5.2**:

- **Timestamp driven compilation**: ObjecTime Developer for C++ currently 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 in now in the form of one .h and one .cpp file created for each actor and protocol class. For data classes, this is done at the package level. That is, all data classes in a package are contained in one .h and .cpp source file pair.
- **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 de-

rived 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.

- **Improved Compilation performanc**e: Compilation performance for large models has been improved. Depending on the model specifics, an average compilation performance improvement of 30% is typical on a full model compilation. These improvements vary widely and a performance improvement of 300% has been realized on an extremely large test model.
- **Internal model dependencies**: These are 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.
- **On-target message sequence charts (MSCs)**: Target Observability has been enhanced. Users can now use on-target tracing and MSCs to carry out substantially the same activities on target as they are able to do in simulation.
- **Timer enhancement**: The Target Services Library now supports true relative timers. An API is provided to inform ObjecTime of any time adjustments.
- **All overrides supported**: Actor reference, port, SPP, and binding replication factors can now be overridden in subclasses. When coupled with 5.1.1 changes to support overriding of an actor reference class, this brings the functionality to the same level as simulation mode. This offers you new opportunities for reuse of modeling artifacts.
- **Actor references and actor importation enhancements**: The frame service now uses linked lists to manage the free list for both optional and imported references, as well as a separate linked list to manage the import list. The net result is improved performance for all frame service primitives.
- **Frame service enhancements**: The frame service has been enhanced:
  - supports ROOM semantics when using the frame service in a multi-threaded environment.
  - data structures of frame service have been enhanced so that finding a free element takes a fixed amount of time. A number of options are provided to manage time and space trade-offs.
- **C++ Tornado integration on Windows NT**: This has been enhanced. Please refer to the *C++ Language Guide* Appendix D for details.
- **New target Suppor**t: Developer for C++ 5.2 supports Chorus ClassiX and Enea OSE target platforms.
- **Annotated Diffs**: An annotated interface diff listing is available for the C++ Target Services Library (TargetRTS). The diff can be obtained from the ObjecTime Customer Support restricted-access website.

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.

The Target Run-Time Services Library provides only limited functionality with respect to time of day representations. The only internal representations are as follows:

1 **RTTimespec** — A specification of the number of seconds and nano-seconds from some arbitrary point in the past. The "zero" value for RTTimespec varies from one Operating System to another. The two most common "zero" values are:

   a) The boot time of the board (Most Real-Time Operating Systems)

   b) Midnight January 1, 1970 (Most flavors of UNIX)

   c) Midnight January 1, 1980 (Most Microsoft products)

   Note: The seconds field is a signed 32 bit number which can represent 68.1 years past the "zero" date.

2 **Clock Ticks** — Some of the time manipulation for the Timing service is done in terms of clock ticks. These ticks are generally represented as an unsigned number of ticks since system boot.

In cases 1a and 2 there are no association with calendar date, and as such no problems with year 2000. In cases 1b) and 1c), the Operating System is responsible for providing the "start" or "zero" value for time. As time passes there are no additional requirements on date calculation, also leaving no problems with the year 2000.

There is however one supported target platform (PSOS) that uses a mixture of clock ticks (for RTSyncObject::timedwait) and real calendar date/time for the underlying acquisition of the current RTTimespec (in the method RTTimespec::getclock). This platform explicitly does date calculations and has been tested to ensure proper operation over the Year 2000 boundary. For further information on the implementation of this getclock() method see the Target Run-Time Services Library source code file "$OBJECTIME_HOME/C++/TargetRTS/src/target/PSOS2/RTTimespec/getclock.cc".

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.

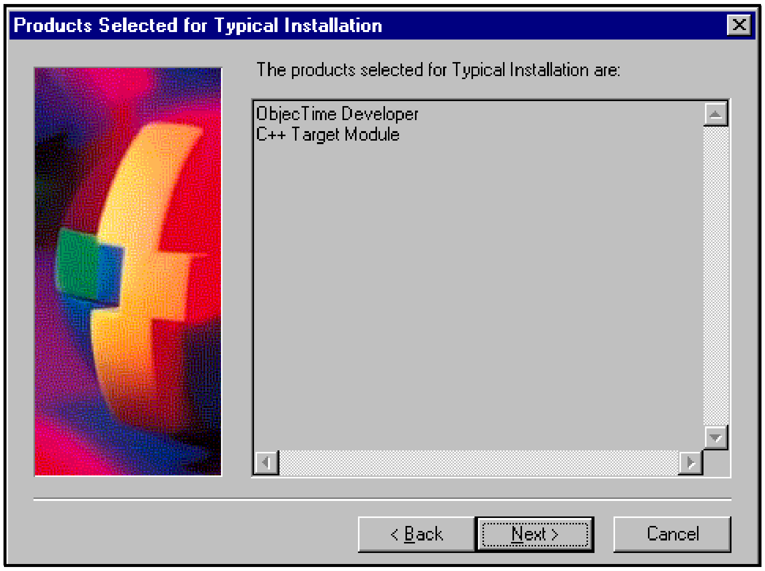
Installation Information

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.

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

The only difference between the base ObjecTime Developer installation and the ObjecTime Developer for C++ installation will be the number of packages which 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.

Figure 1 On Windows NT



Products Selected for Typical Installation

The products selected for Typical Installation are:

ObjecTime Developer
C++ Target Module

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 5.2,** the TargetRTS was renamed to C++ Target Services Library to better reflect the implementation of the TargetRTS and to clearly identify the different libraries.

Where the context is clear, C++ Target or C++ TargetRTS may be used to refer to the C++ Target Services Library. While the default library name is still TargetRTS in the RTS Versions Browser, this can be set to any legal directory name when customizing the C++ Target Service Library.

Documentation Errata:

Note the following updates to the information contained in the **ObjecTime Developer 5.2** documentation. Please read and note the following changes in your documentation set.

***C++ Language Guide*, Using C++ Data Classes, Chapter 3, page 49, in the third paragraph, the last sentence should read as follows:**

To integrate your external type, fill out the definition in the Data Class Editor. (PR 8446)

***C++ Target Guide*, Design Components, Chapter 3, page 30, under RTIOController, the following information completes this section:**

The initialization of the External Layer Service of the TargetRTS requires that TCP/IP interface already be completely initialized. If the TCP/IP interface is not initialized, then the External Layer Service initialization will detect this and therefore it will not complete initialization. The specific error detected and reported is a bind() failure. Situations where this can occur is when the target TCP/IP is initialized using the RARP protocol and the response from RARP does not complete before the TargetRTS application starts or, more specifically, the External Layer Service initialization. (PR 5900)

***C++ Language Guide*, Using C++ in Actor Classes, Chapter 2, page 27, the 'Macros Used in Actor Code Segments', the POINTER_OF section should read as follows:**

POINTER_OF

The POINTER_OF macro can be used for coercing an RTPointer object into a pointer of the class specified.

Usage: If `ptrWrapper` is the `RTPointer` instance to be coerced, the POINTER_OF macro would be used as follows:

MyData* mydataPtr = POINTER_OF(MyData)(ptrWrapper);

***C++ Languge Guide*, Differences between SimulationRTS and TargetRTS, Chapter 14, page 307, the 'Indexing of replicated ports' section should read as follows:**

Indexing of replicated ports

The indexing of replicated port references starts at 0 in the Simulation and TargetRTS when using

the index operator '[]'. For example, portRef[0] is the first (or only) incarnation of the port. When using the `at()` function in the Simulation RTS, it returns a pointer to a port incarnation and always starts at index 1.

```
portRef[0] == portRef.at(1)
```

(PR 8919)

***C++ Language Guide*, Compiling C++ Models, Chapter 5, page 90, the ‘Reserved Names’ section should include:**

`getId`

(PR 8573)

***C++ Language Guide*, Makefiles, Chapter 11, page 194, in ‘Compiling and linking extra files with USER_OBJS’, change:**

the reference in the last section of code from:

```
RTUpdateCompile.mk
```

to

```
RTUpdate_Compile.mk.
```

***C++ Language Guide*, Makefiles, Chapter 11, page 194, delete:**

the section ‘Adding other compilation tasks to the compilation process’.

Supported Reference Platforms

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

5.2.1 Supported Platforms

Toolset Host	Target Library Name	Target Services Library
AIX 4.2.1 (PowerPC)	AIX4S.ppc-CSet-3.1.4 AIX4T.ppc-CSet-3.1.4	Supplied
	AIX4S.ppc-gnu-2.8.1 AIX4T.ppc-gnu-2.8.1	Supplied
HPUX 10.20	HPUX10S.hppa-gnu-2.8.1	Supplied
	HPUX10S.hppa-HPC++-10.11	Supplied
IRIX 6.2	IRIX6S.r4400-gnu-2.8.1	Supplied
	IRIX6S.r4400-ProDev-7.2	Supplied
Solaris 2.5.1 Solaris 2.6	SUN5S.sparc-gnu-2.8.1 SUN5T.sparc-gnu-2.8.1	Supplied
	SUN5S.sparc-SunC++-4.0.1 SUN5T.sparc-SunC++-4.0.1	Supplied
	SUN5S.sparc-SunC++-4.1 SUN5T.sparc-SunC++-4.1	Supplied
	SUN5S.sparc-SunC++-4.2 SUN5T.sparc-SunC++-4.2	Supplied
	SUN5S.sparc-Green-1.8.8 SUN5T.sparc-Green-1.8.8	Supplied

5.2.1 Supported Platforms

Toolset Host	Target Library Name	Target Services Library
SunOS 4.1.3	SUN4S.sparc-gnu-2.8.1	Supplied
	SUN4S.sparc-SunC++-4.0.1	Supplied
	SUN4S.sparc-Green-1.8.8	Supplied
WinNT 4.0	NT40T.x86-VisualC++-5.0[a]	Supplied
	NT40T.x86-VisualC++-6.0[a]	Supplied
Solaris 2.5.1 Solaris 2.6	LYNX30S.ppc-cygnus-2.7-97rl LYNX30T.ppc-cygnus-2.7-97r1 (cygnus tools for LynxOS 3.0.1 on PowerPC)	Generate
	PSOS2T.m68040-Green-1.8.7B (pRISM: Green Hills tools for pSOS 2.1.4 on 68K)	Generate
	PSOS2T.ppc603-Green-1.8.7C (pRISM: Green Hills tools for pSOS 2.1.3 on PowerPC)	Generate
	PSOS22T.m68040-Microtec-4.5G (Microtec tools for pSOS 2.2.0 on 68K)	Generate
	PSOS2T.ppc603-Diab-4.0b (pRISM+: Diab/SDS tools for pSOS 2.2.1 for PowerPC)	Generate
NC	QNX4S.x86-WC++-10.6 (Watcom tools for QNX4.24 on x86)	Generate
HPUX 10.2 0 Solaris 2.5.1 Solaris 2.6 WinNT 4.0	TORNADO101T.m68040-cygnus-2.7.2-960126 (Tornado 1.0.1: cygnus tools for VxWorks 5.3.1 on 68K)	Generate
	TORNADO101T.x86-cygnus-2.7.2-960126 (Tornado 1.0.1: cygnus tools for VxWorks 5.3.1 on x86)	Generate
	TORNADO101T.ppc-cygnus-2.7.2-960126 (Tornado 1.0.1: cygnus tools for VxWorks 5.3.1 on Pow-erPC)	Generate
	TORNADO101T.i960-cygnus-2.7.2-960126 (Tornado 1.0.1: cygnus tools for VxWorks 5.3.1 on Pow-erPC)	Generate

5.2.1 Supported Platforms

Toolset Host	Target Library Name	Target Services Library
NC	UNIXWARE212S.x86-SDK-2.1 UNIXWARE212T.x86-SDK-2.1 (SCO UnixWare 2.1.2 using SCO UnixWare SDK Version 2.1)	Generate
Solaris 2.5.1 Solaris 2.6	VRTX4T.m68040-Microtec-4.5T (SPECTRA: Microtec tools for VRTX 4.AAA on 68K)	Generate
Solaris 2.5.1 Solaris 2.6 HPUX 10.20	VRTX4T.ppc603-Microtec-1.3C (SPECTRA: Microtec tools for VRTX 4.AB on PowerPC)	Generate
WinNT 4.0	VRTX4T.ppc603-Microtec-1.4 (SPECTRA: Microtec tools for VRTX 4.Baa on PowerPC)	Generate
Solaris 2.5.1 Solaris 2.6	XEC68V3T.m68360-Microtec-4.5G (Microtec tools for XEC 3.0 on 68K)	Generate
Solaris 2.6	CLASSIX312T.ppc603-gnu-2.7.2 (Chorus tools on PowerPC)	Generate
Solaris 2.6 Solaris 2.5.1 WinNT 4.0	OSE31T.ppc603-Diab-4.1a (Diab 4.1a, SDS 7.1.1 tools for OSE 3.1 for PowerPC)	Generate

a. Through ObjecTime's use of a subset of the Windows 32 API calls which are common to both the Windows NT and Windows 95 operating systems, binary compatibility between the two Windows platforms can be expected.

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 installation. See the C++ Target Guide for further details.
- **Generate** = not supplied as part of the ObjecTime Developer installation, but can be generated from source code that is supplied.
- **NC** = Native Compilation is used. The code is generated on any host and then compiled on the Target. For additional information, please see "Native Compilation (NC)" on page 42 of this Guide.

Targets No Longer Supported in Objectime Developer for C++ 5.2.1

The following are host platforms or compilers that were supported in ObjecTime Developer for C++ 5.2, but are no longer supported with ObjecTime Developer 5.2.1:

Toolset Host	Target
Solaris 2.5.1 Solaris 2.6	LYNX25S.ppc-cygnus-2.7-96ql LYNX25T.ppc-cygnus-2.7-96q1 (cygnus tools for LynxOS 2.5 on PowerPC)
WinNT 4.0	NT40T.x86-VisualC++-4.2[a]
WinNT 4.0	OSE30T.ppc603-Diab-4.1a (Diab 4.1a, SDS 7.1.1 tools for OSE 3.0 for PowerPC)

Chapter 3

Changes in Developer for C++ 5.2.1

This chapter explores some of the changes affecting users of ObjecTime Developer for C++ and the C++ Target Services Library.

General Enhancements:

- Limited support for 8.3 compilers (5.2.1)
- New C++ Target Service Library Ports
- Overrides
- Target Observability
- External Layer Service
- AIX Support

Pre-5.2.1 models being converted may be impacted by the following changes and enhancements:

- Compilation Changes
- Differences in Generated C++ Code
- C++ Target Services Library Changes

Limited support for 8.3 compilers

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.

New C++ Target Services Library Ports

Support for two new ports was added in **ObjecTime Developer for C++ 5.2** , Chorus Classix version 3.1.2 with GNU 2.7.2 and support for OSE release 3 with Diab and SDS tools. For complete reference information, please refer to "5.2.1 Supported Platforms" on page 9.

Overrides

The C++ TargetRTS and SimulationRTS now support equivalent overrides in an actor's subclass.

Table 1 Equivalent Overrides

Model Element	Override	New in
Port	class	4.4.1
	replication factor	5.2
SPP	class	4.4.1
	replication factor	5.2
Binding	replication factor	5.2
Actor	class (even if fixed reference)	5.1.1
	replication factor	5.2

Target Observability

MSCs are now supported when using the C++ Target Services Library. More information can be found in the *C++ Language Guide* in the section 'Debugging C++ Models'.

ObjecTime Developer 5.2 enhanced the integration with Microsoft Developer Studio (VC50) and Tornado on Windows NT. As well, extensive documentation is provided in the appendices of the *C++ Language Guide*.

External Layer Service (ELS)

With the advent of the Custom Peer Controller (CPC) in **ObjecTime Developer 5.2**, it was not necessary to provide support libraries for a standalone ELS program that provides equivalent functionality to a SAP. Either a simple model can be used to integrate legacy code (using the CPC) with the ELS[1] or a program can be written that follows the ELS protocol specifications.

A standalone program from 5.1.1 will continue to be compatible with the 5.2 ELS thereby ensuring that existing software can be used. Please refer to the *C++ Language Guide* for more information on the ELS and to the *C++ Target Guide* for more information on the CPC.

Enhanced AIX Support

Included with the **ObjecTime Developer 5.2** release were enhancements to the integration with AIX by fully supporting "Basic" debug mode. This provides the ability to automatically load an executable under AIX as well as some basic debug commands. For complete details on the debug commands available, please see the ObjecTime Developer *User Guide*.

Compilation Changes

Please refer to the **ObjecTime Developer 5.2.1** *Getting Started and Release Notice* for additional information about changes affecting the compilation of models. In the context of C++, the following changes are of particular interest:

- One source file is created for each actor, protocol class and package.
- The .cpp extension is used for all generated source files.
- Data classes are compiled at the package level. (All data classes in a given package are compiled together.)
- The meaning of compile and load buttons in the compile dialog has changed slightly since executables are not automatically deleted. If, for example, external libraries change, then the executable must be manually deleted. See the ObjecTime Developer *User Guide* for more details.
- With the addition of support for multiple configurations, object files and executables in 5.2 are not removed automatically. Please refer to the *C++ Language Guide* for more details on the structure of the generated directories. One **key change** to note is that the **executable** is placed in a **new location**:

  ```
  <update>/build/<configuration names/<executable>
  ```
- Inclusions are now possible on data classes and can be more specific:
  - interface: inclusion appears in the .h file
  - implementation: inclusion appears in the .cpp file

  Please refer to the *C++ Language Guide* (Using C++ Data Classes) for more details.
- CUPs were removed in **ObjecTime Developer 5.2** and replaced by inclusions and compilation options at the package level.
- As of **ObjecTime Developer 5.2**, it became possible to reuse the products (object files) from a common loadbuild. Please refer to the *User Guide* for more detail.
- If the detailed code in one part of the model refers to another ROOM class, this relationship can now be captured using dependencies for compilation purposes.

1. The Communications Application Note which provides more detail can be obtained from ObjecTime Support.

Differences in Generated C++ Code

A number of changes were introduced in the **ObjecTime Developer 5.2** release to facilitate improved code generation and compilation. ObjecTime Developer 5.2 minimized the impact of upgrading models from previous releases. However, to meet performance objectives, some changes are required. The information that follows captures these changes and describes conversion activities required.

File restructuring

In **ObjecTime Developer 5.2**, only two files are generated per class, a header and an implementation. This includes actor and protocol classes. In previous releases, a file was generated for each user code segment. Also in previous releases, protocol classes were not explicitly generated.

Compilation Dependency Reduction

With the change to two files per class, data classes are no longer globally accessible. If user code segments of an actor class or other data class requires the definition of a data class that is not otherwise accessed by ROOM constructs, for example, ESV type, the class dependencies[2] must be added manually.

Global signals

In previous releases, all user code had access to the definitions of all signals defined for all protocol classes. In **ObjecTime Developer 5.2**, an actor class has automatic access only to signals defined for ports on that actor class. Data classes do not have automatic access to any signals. Access can be granted by adding the appropriate protocol classes to the dependency list of the appropriate actor and/or data classes.

Actor Class Ids

Actor classes are identified in user code segments by objects of class RTActorClass. In previous releases, these objects were integers that indexed into a global table. In **ObjecTime Developer 5.2**, `RTActorClass` is a structure that contains a variety of information concerning the class. Since `RTActorClass` is now a structure, there is no longer an assignment operator for these objects. Any algorithms that collect actor classes must now use pointers to these objects.

Terminating Semi-Colons for User Code

In previous releases, if a user code segment did not terminate in a semi-colon, one was automatically added. This feature has been removed.

2. When a model is brought forward into 5.2.1, dependencies are automatically generated. Refer to the Model Upgrade/Conversion chapter in the *ObjecTime Developer 5.2.1 Getting Started Guide and Release Notice*.

Port/SAP Numbering

The ordering of port and SAP numbers for actor classes have changed to match the order these elements appear in the Linear Form. In particular, interface ports are first, followed by end ports, then finally SAPs. This change does not affect user code segments.

Names of Chain Methods

The name of the state has been dropped from the method names for the chain methods. The full name of the transition starting the chain now appears in the comments for the chain method.

ESV Ordering

The ESVs of an actor class may appear in the definition of the actor class in a different order than in previous releases. This will affect the order in which they physically appear in memory as well as the order in which their constructors may be called.

Class Meta Data Changes

There have been significant changes to the class meta data for actor and data classes, especially with actor classes. These structures are considered private and do not affect user code segments.

CALLSUPER for functions with complex arguments

CALLSUPER will no longer compile for functions that have function pointers as arguments. It is recommended that the SUPER::func() pattern be used to call actor functions.

Cosmetic changes

There have been a number of cosmetic changes to the generated code in order to make it more readable. These changes do not affect the structure or behavior of the model but should help when debugging models.

C++ Target Services Library Changes

This section covers library changes in more detail as follows:

- semantic changes
- API changes
- user code changes necessary for new Target Run-Time Services Library
- actor references - fast access
- actor importation - enhancement
- frame service - mutual exclusion
- concurrency issues
- RTActorClass -type changed
- overrides - memory impact
- new mutexes in the Target RTS

Semantic Changes

RTSignal and RTActorClass cannot be sent as data.

`RTSignal` is now transferred on the external layer as a string rather than an `RTInteger`. This means that `RTSignal` may not be sent as the data portion of a message. `RTActorClass` is no longer a subclass of `RTInteger` and can no longer be sent over the layer service. `RTActorClass` constructor which takes an int parameter has also been removed.

Application Programmer Interface (API) Changes

isRefCompatible

The function `isRefCompatible` was removed from the generated code in 5.2 and is now calculated when an actor is incarnated or imported. This results in a substantial space savings. Due to overall performance enhancements in the frame service, the overall performance is the same as 5.1.1 or faster.

Adjusting the System Clock

ObjecTime Developer 5.2 fully supports the ability to adjust the clock that is used by ObjecTime to derive both absolute and relative timers. In order to support applications that must operate in an environment where the real-time clock needs to be adjusted, two new methods have been added to the **RTTimerSAP** class.

If ever the real-time clock needs to be adjusted the following functions must be used to bracket the change so that the run-time system can be aware of that event and make any necessary adjustments to its internal data structures so that relative timeouts (requested via one of the informIn methods or one of the informEvery methods) are not affected.

```
void       adjustTimeBegin( void );
void         adjustTimeEnd( const RTTimespec & delta );
```

The application must coordinate time changes through an actor with a timing SAP. That is, the adjustment API must be called from within the thread of control of the ObjecTime model, not from an external thread of control. The example below encapsulates the behavior in a function. We assume that the timing SAP is called *timer* and that there are O/S functions available for reading and writing the system clock which we term as *sys_getclock* and *sys_setclock*, respectively.

```
void AdjustTimeActor::setclock( const RTTimespec & new_time )
{
   RTTimespec old_time;
   RTTimespec delta;

   timer.adjustTimeBegin(); // stop ObjecTime timer service

   sys_getclock( old_time ); // an OS-specific function
   sys_setclock( new_time ); // an OS-specific function

   delta  = new_time;
   delta -= old_time;

   timer.adjustTimeEnd( delta ); // resume ObjecTime timer service
 }
```

Additions to the Frame Service

The following frame service methods have now been implemented:

1. isKindOf
2. className
3. incarnationAt

Changes to RTLogSAP and External Types

In 5.1.1 the log service could "log" objects which were of an external type. In **ObjecTime Developer 5.2** the log service must be invoked with the `SEND_EXT`.

Code from 5.1.1 that matches the following pattern:

```
   External_type ext;
   Log.show( ext );
```

Would be replaced in 5.2 with

```
   External_type ext;
   Log.show( SEND_EXT(ext) );
```

Incarnations() method is no longer supported

The `incarnations()` method of `RTActorRef` and `RTPortRef` are no longer supported. Typically these methods were either not used, or used in the following fashion:

```
inc = reference.incarnations();
for( int i = 0; i < reference.size(); i++ )
  Do_somethingTo(inc[i]);
```

This can easily be replaced with:

```
for( int i = 0; i < reference.size( ); i++ )
  Do_somethingTo(reference[i]);
```

The constructor for RTController has changed

The first parameter for the constructor for `RTController` in 5.1.1 was a pointer to the `RTResourceMgr`. This pointer is no longer needed.

User Code changes necessary for new Target Services Library

The detail level code of models may have to change for the following reasons:

1. **If it references ActorClassIds:** In the case where users are managing a pool of actor classes by their integer class Ids, they should modify it so that a pool of pointers to `RTActorClass` is maintained instead.
2. **If the detail level code does not end with a semi-colon:** Pre-5.2 toolsets added a semi-colon in this case but 5.2 does not and a compile error will result.
3. **If a signal is sent out a port in which the protocol does not contain the sent signal:** This may cause a compile error in 5.2, depending on what other ports are referenced by the actor. In pre-5.2, any signal could be sent out of any port, regardless of whether it was part of the port protocol.

Actor References - Fast Access

Description of change

The frame service in **ObjecTime Developer 5.2** uses a linked list to manage the free list for both optional and imported references. This change removes the need to search for a free item and therefore provides improved performance without the need for the user to maintain their own free list. This results in an increased memory allocation size of 2n pointers (where n is the replication factor of the Actor reference). An optional Actor reference will therefore increase from 1n (5.1.1) to 3n (5.2) pointers. Imported Actor references are described later.

Rationale for change

This change addressed a feature request for the frame service to remove the need to search for a free item. This is especially critical in an imported Actor reference.

Workaround

If the increased memory usage or the increased size of the memory allocation cannot be accommodated then it is possible to remove this linked list through `RTConfig.h`. The negative side effect of this change is that certain scenarios (in which multiple threads manipulate actors, via the frame service, in the same `RTActorRef`) may cause memory corruption.

RTConfig.h contents

```
// Maintain a free list in RTActorRefs for optional components?
// The free list costs two pointers per replication but avoids
// a linear search in incarnate and import operations.
// Pre-5.2 compatible behavior is available by making
// RTFRAME_USE_FREELIST zero.
#ifndef RTFRAME_USE_FREELIST
#define RTFRAME_USE_FREELIST 1
#endif
```

Actor Importation - Performance Improvement

Description of change

The frame service in Release 5.2 pre-allocates the linked list used to manage the import list (since an Actor can be imported in many different references). This results in an increased, up-front, memory allocation size of 2n pointers (where n is the replication factor of the imported Actor reference). An imported Actor reference will therefore increase, taking into account the free list, from 1n (5.1.1) to 5n (5.2) pointers. Release 5.1.1 allocated this linked list ‘on-the-fly’ which means that the new/delete were called on every import/deport. When taking into account the fixed overhead for each block of memory, the break-even point (the point at which less total memory for the import list is being used) is when approximately 25% of the reference contains imported actors.

Note: In all cases an import or deport will no longer need to access the heap.

Rationale for change

This change was made to increase the performance of import/deport especially in the area of memory management.

Workaround

If the potentially increased memory usage or the increased up-front size of the memory allocation cannot be accommodated, then the design will have to be modified to reduce the replication factor.

Frame Service - Mutual Exclusion

Description of change

The frame service includes a number of additional checks in Release 5.2 to guard against potential race conditions. It should be noted that these changes were based on an analysis of the current algorithms since the guarded conditions have not been detected to date in the field or through product verification. These race conditions were possible when two or more threads are importing, deporting, or destroying actors in the same actor reference.

Rationale for change

This change was made to enforce ROOM semantics under all scenarios.

Workaround

Three options in `RTConfig.h` provide a range of increased checking. By default, they are all enabled but in the case where a pre-5.2 model is being converted, it may be necessary to back-off on some of these options.

RTConfig.h Contents

```
// Possible values for RTFRAME_CHECKING:
// The frame service is intended to provide operations on components
// of the actor which has the frame SAP. The checking can be relaxed
// or removed.
#define RTFRAME_CHECK_NONE   0
// no checking (pre-5.2 compatible)
# define RTFRAME_CHECK_LOOSE  2
// references must be in the same thread
#define RTFRAME_CHECK_STRICT  4
// reference must be in the same actor

#ifndef RTFRAME_CHECKING
#define RTFRAME_CHECKING RTFRAME_CHECK_STRICT
#endif

// Setting this macro to 1 guarantees that the frame service is thread
// safe. This is an option because some applications may use the frame
// service in ways that don’t require this level of safety.
#ifdef  RTFRAME_THREAD_SAFE
#define  RTFRAME_THREAD_SAFE  1
#endif
```

Summary of Concurrency issues

ObjecTime Developer for C++ 5.2 was enhanced to address some very rare circumstances in which potential race conditions may occur in multi-threaded environments. Potential issues are as follows:

- Actor reference management

- import list corruption
- relay port reservation errors (inconsistent bindings)

If an existing design is working correctly, it will continue to work correctly in Release 5.2. However, if the design uses the frame service concurrently (from multiple threads) then the changes in Release 5.2 will guarantee that no race conditions will occur.

It should be realized that there is no additional performance penalty in terms of the execution time it takes to perform a given frame service call. However, priority inversion may occur if a low priority thread is doing an extended frame service call and a high priority thread makes a frame service call.

Table 2 Frame Service Options

Option	Default	Impact	Notes
RTFRAME_USE_FREELIST	1 (enabled)	2n extra pointers for both optional and imported Actor references. Access time to a free slot is constant.	Also provides mutual exclusion when dealing with Actor references. If disabled then RTFRAME_CHECKING should be enabled
RTFRAME_THREAD_SAFE	1 (enabled)	ROOM semantics enforced. Potential for increased latency time before a higher priority frame service call can start.	If disabled then bindings may be inconsistent if there are concurrent frame service calls.
RTFRAME_CHECKING	STRICT	ROOM semantics enforced.	Needs to be at least LOOSE if free list is disabled.

RTActorClass - Type changed

Description of change

The frame has a new set of APIs for dealing with Actor classes - `RTActorClas`s. The APIs which used an int have been removed.

Rationale for change

This change was done to allow more efficient code generation.

Workaround

Code relying on `RTActor` class being an int will NOT compile in 5.2. All operations are still supported:

```
- frame.my_class()              => RTActorClass
- frame.classesof(RTActorId)    => RTActorClass
```

Summary

- `RTActorClass` is not a subclass of `RTInteger.`
- `RTActorClass` constructor which takes an int parameter has been removed.
- An `RTActorClass` can no longer be sent in a message
    - send a pointer to the class
- The `_classId` method has been removed.
- Manage pointers to `ActorClasses` instead of the `ActorClasses` themselves (works in 5.1.1 and 5.2)

```
Static const RTActorClass * tests[] =
        {&ClassRTSignal,
        &ClassFoobar,
        ...}
```

Overrides

Description of change

Additional overrides are supported in 5.2. Code generation patterns and the TargetRTS have been modified to handle the overrides described in “Equivalent Overrides” on page 15.

Rationale for change

There were a number of long standing requests to remove these restrictions from the C++ TargetRTS.

Workaround

There is no visible impact due to these changes except for an increased usage of statically allocated data. On most RTOSs this memory is statically configured (as part of the h/w configuration) and may need to be re-engineered. Overall dynamic memory is decreased by approximately the same amount.

Mutexes in the Target Services Library

Frame mutex (new in 5.2)

- shared by all frame service calls
- protects frame service data structures
- for calculating bindings (followIn(), followOut())

Binding Mutex (same as in 5.1.1)

- to protect binding in frame service
- checked by inter-thread messaging

Problems Addressed in this Release

For a complete list of problems that 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.

General Information

Limits

SimulationRTS Limits

- 16,384 actors can be incarnated at run-time in a model.
- When using the SimulationRTS you must have selected the “Basic” debugging tool in order allow the “Load” option to be used.

Target Services Library Limits

- Actor References: There can be up to 2^{31}-1 replications of each actor reference.
- SAPs are identified by a 16-bit number. There can be up to 65535 SAPs in a single actor class, and up to 2^{31}-1 replications of each.
- States are identified by a 16-bit number. There can be up to 65535 states per actor. This can be configured in RTConfig.h.

Target Observability Limits

- Dye traces are not possible with Target Observability. (PR 8994)
- The maximum number of controller probes for a single executing TargetRTS is 30. One controller probe is generated for each navigator, each behavior or structure monitor, and each port, state or transition daemon created in the RTS.
- The maximum threshold for a trace window configuration is 10,000.
- If the target doesn’t reply to periodic polls within a 10 second ‘target alive’ interval, the toolset displays a warning dialog indicating that the connection to the target has been lost. If the target returns, the toolset displays another information dialog indicating that the target has been successfully reconnected.

External Layer Limits

- When binding SAPs to SPPs using the External Layer Service, the replication of the proxy SPP which is used to bind SAPs to a remote SPP is defined by a static int defined in RTProxySPP. The default value is 100 in ObjecTime 5.2.1.

  By linking in a new definition:

```
int RTProxySPP::replicationFactor = 200;
```

this can be increased without modifying the Target Services Library.

Target Services Library Invoke - Usage Hints

- The invoke communication service primitives of the Target Services Library (TargetRTS) do not copy the data supplied in the versions which accept an argument of type ‘const RTDataObject &’. They are:

  ```
   RTEndPortRef::invoke( RTMessage *, int, const RTDataObject & ) const
   RTEndPort::invoke( RTMessage *, int, const RTDataObject & ) const.
  ```

  This should have no impact on properly written applications other than to improve performance.

  Proper applications are those in which the code of each transition is written as if it were the body of a function with the following signature (this is one triggered by a signal accompanied by data of type RTInteger). `RTDATA` is actually a macro, but it should be treated as a read-only pointer which refers to read-only data.

  ```
  void transition( const RTInteger * const RTDATA )
  {
        // transition code goes here
  }
  ```

  Similar comments apply to guard, choice-point, entry and exit code fragments. Failure to respect access restrictions on the data provided in the message can lead to unintended interactions between actors. These are interactions that were previously hidden because each receiver of an invoke had a separate copy of the data to with which to work. Take the example of an actor handling the `doubleIt` signal.

  A correct implementation is:

  ```
  void transition1_doubleIt()
  {
        RTInteger value( *RTDATA );
        value *= 2;
        msg->reply( doubled, value );
  }
  ```

  Consider the implications if it were written as:

  ```
  void transition1_doubleIt()
  {
        RTInteger * value = (RTInteger *)msg->data;
        *value *= 2;
        msg->reply( doubled, *value );
  }
  ```

  If this behavior is invoked:

  ```
  {
        static const RTInteger one( 1 );
        RTMessage              response;
        doubler.invoke( &response, doubleIt, one );
        const RTInteger & result = *(const RTInteger *)response.data;
  }
  ```

The result is that the value of ‘one’ changes each time this code is executed until it overflows and becomes zero. This happens because, each call to `transition1_doubleIt` sees:

```
msg->data == (void *)&one
```

This happens sooner if the port ‘doubler’ is replicated because each receiver doubles the value of one. Of course, this is an oversimplified example. In a more complex application the interactions may be more subtle. In some cases, application code may need to create a copy of the incoming data which can then be modified, but this is better, from a performance perspective, than having the run-time system copy the data always.

Special Notes and Reminders

- In ObjecTime Developer Release 5.2.1, the `OTRTSDEBUG` debug symbol is used to control the level of target debugging that is possible:

  ```
  OTRTSDEBUG==DEBUG_VERBOSE
  ```

  and is required to run target observability. However, only in the case of OTRTSDEBUG==DEBUG_NONE is the instrumentation of the TargetRTS totally removed. Therefore performance measurements should be done with this configuration.

- **Windows NT supports 4 priority classes**:

```
Name in Task Mgr      Name in MFC .h and on MicroRTS cmd line
realtime              realtime
high                  high
normal                normal
low                   idle
```

  A model started with:

  ```
  Test_Actor -priorityClass=idle
  ```

  would execute in the idle (aka low priority) class.

- **Optimizimg C++ Memory Utilization**

  In ObjecTime 4.4 a special memory management feature, “memcache” was provided.

  In 5.2.1 “memcache” is NOT provided as part of the release. Its function was to provide more efficient memory management for small objects by using a chunk of memory partitioned into pools of fixed size blocks. It was implemented by overriding NEW on a global basis.

  Experience has shown that while optimizing C++ memory utilization (for example, faster allocation, reduced fragmentation) is often important in real-time systems optimal solutions end up being application specific.

  Application designers need to consider when to override NEW either for a particular class or globally but also need to ensure that the issues of thread safety, engineering and utilization metrics are also covered in a manner that best fits the application and operating system.

- **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.

- **RTAsciiDecoding doesn't use all the data it receives.** `RTIBuffer` reserves the last byte of the buffer supplied for a null, even though it is being used in a read-only manner. The application must specify that the buffer is one byte larger than it really is.

Hints on porting existing UNIX models to Windows NT using MS Developer Studio and Visual C++ 5.0 and 6.0

Porting existing models developed on ObjecTime Developer for UNIX, to ObjecTime Developer 5.2.1 for Windows NT requires some special attention. The first thing you should do is port your model to ObjecTime Developer 5.2.1 for Unix. Once this conversion has been completed, the move to ObjecTime Developer 5.2.1 for Windows NT should be performed.

- First ensure that the Microsoft Visual C++ environment is functioning outside of ObjecTime. To facilitate proper inclusion of header files, MSDEV (or Dev. Studio) should be installed in a path that contains no spaces. Normally the **`INCLUDE`** variable is used to find Microsoft Visual C++ header files which has no problem with spaces in the pathname. By surrounding include directory paths in double quotes (") in the toolset, the problem with finding other header files is eliminated.
- Microsoft Visual C++ 5.0 and 6.0 are quite rigorous, so, depending on the compiler you were previously using, expect to make changes in your C++ code.
- If two class names differ only in case, you will be warned that one of the classes will be renamed. The reason for this is because C++ is case sensitive but WindowsNT is case insensitive. References to the original name must be adjusted likewise.
- The naming conventions for libraries is different from Unix - 'lib' is not prefixed. This is only an issue for third-party libraries which do not follow the Windows NT naming conventions
- You must reparse external data packages.

Overrides files for source level debugging

It is now possible to configure ObjecTime Developer to use an alternative debugger for source level debugging. Example configuration files can be found in $OBJECTIME_HOME/tools/overrides. See "Observing TargetRTS Models using the Simulation Tools" in the chapter **Running and Debugging C++ Models** of the *C++ Language Guide*.

Exiting the Target Services Library - Context->Abort()

The Target Services Library will not cleanly shut down on an `exit()` (or signal). If the user shuts down with a `context()->abort()` from an actor on the main thread, then all of the actors will be destroyed (and associated destructors run). Any other forced shutdown will simply exit with no cleanup.

Return value from Send

SimulationRTS returns the replication factor of the port on which the send is called regardless of how many of the instances are actually connected at the other end.

TargetRTS can return 0 if the send failed (Failure reasons are well-described in the *C++ Language Guide*) If the send was successful, it will return the value of the actual number of successful sends regardless of the replication factor of the port (except, of course, that the returned value would never exceed the port replication factor).

Target Services Library Binding Algorithm

The Target Services Library will not create actors that cannot satisfy the bindings of the container where they are being created. The binding resolution algorithm will take the minimum of the two ends of the binding and the explicit replication factor on the binding and use that value for the binding replication.

> **Note:** Be careful when using split bindings and check all warnings since the SimulationRTS and the TargetRTS are different. The SimulationRTS creates actors that cannot be bound. The TargetRTS does not create actors that cannot be bound. Therefore, some of the binding conflict warnings are in fact errors.

Flattening Source

The Target Services Library source is organized into a large number of small files (many with the same name, but which exist in different directories). This type of source layout may be problematic for some debuggers and code analysis tools. If you run into problems due to this, then you can rebuild the Target Services Library in a “flat” format. This “flat” format converts the source into a “one-file-per-class” format, named <class>.cpp.

This can be done in two different ways:

1 Add to `config/<config>/setup.pl`

```
$flat = 1;
```

2 Set the environment variable

```
SET BUILDOPTS=-flat
```

After either of the above, make sure `build-<config>/depend.mk` does not exist and then specify `[n]make CONFIG=<config>.`

Target Conformance Testing

A test model can be made available to you for the purposes of conformance and performance testing of your target port. The model can be used to measure the performance and functionality of your C++ Target Services Library for a given target environment.

This model will be provided to you by ObjecTime Limited for informal testing purposes only and there is no implied guarantee of performance through the use of these test models in your development or testing phases. Please contact ObjecTime Customer Support for more information on this model, or if you wish to obtain this model for test purposes.

ClearCase

Editing Modes

In a ClearCase environment, the 'Allow edits on non-checkedout objects' user preference should be OFF since, it is necessary to save the changes to a library before the user can compile with the new changes. As well, in the library, checking out a file before modifying it in any way is a more natural operation. If the changes are temporary, the object can be unchecked out. This will cause a merge operation to be triggered allowing the user to revert to the previous version of the objects being unchecked out.

Chapter 5

Troubleshooting

Installation Issues

Please refer to the base *ObjecTime Developer 5.2.1 Getting Started Guide & Release Notice* for installation details and troubleshooting.

Compilation problems

- **Compile fails on valid C++ model for TargetRTS or SimulationRTS with Microsoft Visual C++**

  The INCLUDE and LIB environment variables may not be properly set. Start “ObjecTime Developer Command Prompt” from the “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.

- **Error linking Actor (“error from nmake”)**

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

- **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
```

Windows NT: On Windows NT, use Windows Explorer to get to the Update Root Directory

and delete the LF, build and C++ directories, plus the Makefile. Then, initiate a Recompile From Changes Only (or just a Recompile, if you have removed all of your load-build paths).

Windows NT Compilation Command Line Limits

If you encounter a compilation error message that complains about the command line being too long, the cause may be that the length of your compile or linker has exceeded a limit.

Windows NT compilation has command line limits in two areas: source compilation and linking. Both limits have been explored for the Visual C++ 4.2, Visual C++ 5.0, VRTX PPC Microtec 1.4 and Tornado 1.0.1 PPC Cygnus 2.7.2 compilers.

Source File Compilation

The variables in source compilation are the update name, the $OBJECTIME_HOME path (%OBJECTIME_HOME% in NT), compilation options, the local working directory and include directories. The only compiler that has a measurable limit is VRTX. The command line limit is 768 characters.

A workaround for the problem is to reduce the number of include directories by combining include files. Other solutions are to shorten paths and names for the variables listed in the previous paragraph.

Linking

The variables in linking are the update name, the $OBJECTIME_HOME path (%OBJECTIME_HOME% in NT), the link options, the number and name length of libraries, the library search paths and the local working directory. The link limits are shown below:

Table 3 Link Limits

Platforms	Link Limit
Visual C++ 4.2:	2049 characters
Visual C++ 5.0:	more than 20875 characters
VRTX PPC Microtec 1.4	4147 characters
Tornado 1.0.1 PPC Cygnus 2.7.2	4150 characters
HPUX 10.20	16384 characters (make: “couldn’t load shell.stop”)

A workaround for the problem is to shorten paths and names for the variables listed in the previous paragraph.

Error when compiling the Target Services Library

If you get an error when compiling the Target Services Library, it is often useful to see the commands that `make` is issuing. However, the Target Services Library suppresses this output. In order to see the output, enter:

```
make -n CONFIG=<config> MAKEOPTS=aaaa.o
```

This allows you to see the commands that will be executed to build the first object file from `main.cc`. The commands have the same structure for all object files. If the compiler uses a different suffix for object files, substitute it for '`.o`'.

Alternatively. you could edit `BUILD.pl` which is found in:

```
$OBJECTIME_HOME/C++/TargetRTS/src
```

to avoid prefixing the commands with '`@`'. Look for the line that includes a reference to '`CC_CMD`'.

Appendix A

Developer 5.2.1 Directory Contents

Files and directories included in 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.

- `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.

Known Limitations/Restrictions

Supported Platforms for C++

- The following platforms are supported for version 5.2.1 of the ObjecTime Toolset: AIX 4.2.1 (PowerPC), HPUX 10.20, IRIX 6.2, Solaris 2.5.1, Solaris 2.6, SunOS 4.1.3 and Windows NT 4.0.
- ObjecTime does not guarantee correct operation if you use the -O2 or higher optimization setting with the gnu compiler on the AIX4 single and multi threaded platforms. (PR1943)
- During compilation on pSOS platforms, a Warning: ‘pointer to function cast to pointer to non function’ appears in the Error Browser. This is a valid warning. The (void*) array _types[] is used in Target Observability to allow us to print non-ASN.1 types that are fields of Sequences. It is void* rather than a function pointer because flags can be part of the array as well.

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)

External Layer

- In situations using External Layer short circuit connections with the C++ Target Services Library, you should use a SPP replication factor greater than the number of connections required. This is because of a race condition in which a new connection may be requested before the old one is removed. (PR1874)

Windows NT

- 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++ or update/C directory is opened.
- When you start ObjecTime Developer with Target Observability enabled, there might be a slight delay before the socket connection between the Toolset and the Controller is established. During this time, the '( ) Load' radio button in the Compile dialog box is disabled. If this happens, click [Cancel], wait a couple of seconds, and try again. (PR4181)

- If you start a compilation of a larger C++ model, and abandon the ObjecTime session, the Task Manager reveals that the compiler continues to execute. (PR3834)

Compiling in Windows95

When compiling an ObjecTime executable for Windows95, you must set the following variable:

```
OBJECTIME_OS=Windows_95
```

C++

- The Simulation Services and Target Services Libraries have different time bases when the RTTime class is used. The Toolset and SimulationRTS are based on local time. The C++ Target Services Library is based on Coordinated Universal Time (UTC). The RTTimespec class does NOT have this problem. (PR1854)
- Layer SPPs in C++ actors must be specified with a maximum replication factor, equal to the maximum number of SAP's bound to it. An *unspecified* replication factor in this case equates to "0".
- There is a difference between the incarnate() function in the SimulationRTS and the Target Services Library. When incarnating at a particular index, the SimulationRTS expects the indexing to start from 1, while the Target Services Library expects the indexing to start from 0. You may find it useful to define a macro which hides this difference so that you don't have to change your code when moving from the SimulationRTS to the Target Services Library. See "incarnate" on page 285 of the ObjecTime Developer 5.2 *C++ Language Guide* for more detail.
- Two or more different updates with the same name in different Workspace Browser folders are not distinguished during compilation.
- 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, users must explicitly include the header files containing the definition of TRUE and FALSE.
- If functions on actors or data classes are declared inline they may cause link errors if the body of the inline function requires other ObjecTime data classes which are not yet available to the compiler.
- Currently only PSOS configures the stack size of the main task. The designer-provided value for `RTMain::mainStackSize` is not considered for any other target ports.

Debugging

- XXGDB will exit with a segment violation under certain specific circumstances if you run a model in the Target Services Library with target observability. This happens if you select both Halt and Source Breakpoint in the RTS control panel, and run your model. If you then try to open a source file from within XXGDB after the ObjecTime Halt takes effect but before the XXGDB breakpoint is reached, then XXGDB will exit. There is no problem if you open a source file after the XXGDB breakpoint is reached. (PR480)
- When downloading a very large model to the target board using the SDS Single Step ver 7.0.3 and 7.1.1 debugger, you might receive this message:

  debug:"ram.ou1": error: corrupt symbol table detected

  The reason for this is a bug in these debuggers. SDS is currently looking into this problem. (PR4872)

Target Services Library

- `setup.pl` files in the Target Services Library may contain references to ObjecTime paths. They must be customized as necessary. (PR3396)
- When using the external layer service with the single-threaded Target Services Library, the listening end must be permitted to initialize before the other processes can connect. The “step” debugger command is sufficient.
- When using `nmake` to recompile the Target Services Library you must use the make variable CONFIG to set the default target:

  ```
  nmake CONFIG=NT4.0T.x86-VisualC++-5.0 or use makent.bat
  ```

Target Services Library Debugger

- The 'info' command displays all the 'Components' in a 0-based list, but they must

  be entered as 1-based values.(for use in the info and trace commands)

  Fix: Add 1 to the Component value if you want to use it in another 'info' command. (PR4910)

Compiling on remote systems is sensitive to timestamps

- Generation of code and Makefiles must be completed on the toolset host, and the generated files must be visible to the compilation host. If the files are generated on a directory local to the toolset host and are copied to a directory local to the compilation host, then the files' time-stamps must be preserved. Modification of the timestamps of dependent files may result in re-generation of code and Makefiles on the compilation host, which typically will not have the generator executables.

RTSController Messages

Occasionally, under certain circumstances, the RTSController produces the following messages:

- RTSControlMaster: send on toolset spp failed
- RTSControlMaster: recall - no message

These messages are generated when the following conditions are true:

- Target Observability is turned ON and active
- the rtsController was started manually, that is, not with ‘objectime -control’.
- the application was loading and unloading on a Tornado board, either in manual mode or with the Tornado Debugging Tool activated.

These messages require no user action. (PR 5319)

Compilers/Compilation/Run-Time System - C++

- On rare occasions the compiled version of a design may get out of sync with the design itself. This is manifested by recent design changes not appearing to take effect at run-time resulting in various run-time errors. Selecting ‘Recompile’ in the compile dialog will clear the problem.
- Simulation communication is not synchronized over multiple run-time systems (RTSs) interworking via the Layer Service. Angiotraces cannot be captured over interworking RTSs. See the description of Layered Networking in the *User Guide*.

- There is a link incompatibility between different GNU versions. If you see a link error involving the symbol _addChainSegment, then it is likely that you are using an incompatible version of GNU, and you should upgrade to the 2.8.1 version.
- Using an INVOKE in the initial transition of an actor (or entry code of the initial state) may cause ambiguous user run-time errors if the destination actor has not yet been initialized. If the destination actor is contained within the invoking actor, the problem does not occur, as actors are initialized in depth-first order.

Native Compilation (NC)

Several targets do not support cross compilation. This implies that the code generated and makefiles are first generated on a supported host then the make/compile are executed on the target.

There are a number of restrictions with this setup:

- a shared filesystem must be created on the files or the files must be ftp'd to the target
- incremental compile will not be possible unless a shared filesystem is used.
- the compile and load operations are manual
- the error stream is captured in an external file (compile.output) and can be read back into the toolset to perform the compile error to toolset mapping. This is accomplished through the Import Compilation Results option from the Update menu item.

Inclusion Paths

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

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.

ObjecTime®
www.objectime.com

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