OBJECTIME ®

ObjecTime Developer 5.2.1
Getting Started Guide
& Release Notice

Product Release: ObjecTime Developer 5.2.1
Document Version: 1.0
Release Date: February 1999
Part Number: OT-R521-DOC808

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

Printed in Canada

Important Notice

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

Chapter 1

Welcome to ObjecTime Developer 5.2.1

Introduction

The **ObjecTime Developer 5.2.1** release builds on the capabilities introduced in the previous releases to meet your large project scalability needs. You can now maximize your team's productivity by taking advantage of features such as enhanced project files, ClearCase view synchronization, and improved editing modes.

Developer WebPublisher and Developer TestScope, two optional, separately-purchased components are also available for use with the ObjecTime Developer 5.2.1 release.

Developer WebPublisher enables output of an ObjecTime Developer model in HTML format, so you can view and interactively navigate through the model using a web browser.

Developer TestScope extends ObjecTime Developer's design-automation capabilities to model, debug and test.

This chapter provides an introduction to ObjecTime Developer 5.2.1. The chapter's five main areas describe:

- What's new in Developer 5.2.1/5.2
- Year 2000 Compliance
- Packaging changes introduced for 5.2
- Model examples
- Documentation Errata

What's new in Developer 5.2.1/5.2

The following highlights some of the new features available in the **ObjecTime Developer 5.2.1** release:

- **Project file enhancement**: Project files now contain a list of packages, and only those classes that are in the update but not any package. This allows you to make changes to a package (for example, add or remove classes) without requiring a change in the project file, thereby removing the contention for the project file in large team environments.
- **ClearCase Support Enhancements**:
  - **View synchronization**: ClearCase users can now synchronize the toolset with ClearCase views allowing designers to access the work of others in a parallel fashion.
  - **Version Based or View Based project file activation**: You now have the option of activating a project file using either specific version information for classes, or based on the class versions in the current view.
  - **Enhanced editing mode**: When the "Allow edits on non-checked out objects" user preference is in an unchecked state and a class is changed as a result of some operation on a class that has been checked out, for example, signal change in a protocol resulting in a change to a non-checked out class, the toolset checks out the non-checked out class and marks it as changed.
  - **Save to library command**: A new "save to library" command is added to the update menu. This allows ClearCase users to save any changed classes to the library. This command is also executed automatically at compile time.

  See "ClearCase Support Enhancements" on page 79 of the ObjecTime Developer 5.2.1 (Base) *Getting Started Guide and Release Notice* for further information.
- **Microsoft Visual C++ 6.0**: ObjecTime Developer 5.2.1 supports Microsoft Visual C++ 6.0.
- **General problem fixes**: This release fixes problems reported in the 5.2 release. For full details of problems fixed in this release, please see the "Problems fixed in this release" section.
- **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 new features available in the **ObjecTime Developer 5.2** release:

- **Code Generation and Compilation enhancements**: Release 5.2 incorporates significant changes in the way in which code generation and compilation of models is performed. These changes have been made in order to improve performance as well as to better integrate with the customer's software development environment and processes.

  The significant changes are:

  - The toolset has been partitioned into modeling and code generation components.

- **Timestamp driven make**: Industry standard timestamp driven code generation and compilation (using make and makefiles)
- **Loadbuild reuse**: The reuse of loadbuild results is now supported for ObjecTime models using either ClearCase and derived objects or with the VPATH facility supported by GNU make. See Appendix F in the *User Guide* which describes 'Loadbuild Re-use'.
- **Inter-class dependencies**: Dependencies between classes are now explicitly captured. This allows generation of code for classes which only has the includes it requires. This improves compilation performance.
- **Multiple environment configurations**: Maintain separate configurations for different environments. This is useful in situations where you need to frequently switch between targets of different configurations.
- **Multi-targeting and simulation support**: Switch between simulation and multiple target configurations without the need to do a full recompile upon switching.
- **Cross platform support for ClearCase**: You can now use ObjecTime Developer with ClearCase transparently across NT and UNIX platforms. New configuration management enhancements, listed below, make ObjecTime and ClearCase a powerful combination.
- **Configuration Management related enhancements**: Several CM related enhancements are introduced in this release:
    - A new user preference is provided for checkout policy enforcement. When enabled, classes cannot be modified until after they are checked out from the CM system.
    - A new user preference is provided for version sequencing enforcement. When enabled, the toolset reports version skipping as an error, and provides the option to perform a merge of the two different versions.
    - Hierarchical Libraries. Users can navigate through nested libraries in the same way as navigating through a directory structure.
    - External scripts can be used to achieve custom version handling.
    - Branch compatible Library Browser displays branch tags & version extended pathnames.
    - User configurable global path to library scripts through the Library Configuration dialog. If an objectime_scripts_dir is specified, then it overrides the scripts identified through the global path specification.
    - Multi-library 'sync with library' is supported. As well, a new content synchronization option is also provided to enable synchronization to be based on the actual source content in the library instead of just the version number. If you are a ClearCase user, the content diff scripts are provided for you. If you are not using ClearCase, you must implement the scripts to do the content diff.
- **Features for multi-stream development**:
    - The Differences Tool has been enhanced to detect the graphical information of objects, for example, size or position.
    - A new Class Version Merge Tool, based on the Differences Tool, is now available. You can use it to identify differences between two class versions and propagate modifications from one version to another.

- **Project files**: Project files provide an alternate specification for an update. These files provide a versionable, textual specification of the complete model, and contain information such as names and location of classes/packages in the model, environment configuration file names, etc.
- **Data class inclusions**: You can now specify external inclusions on data classes. This is done via the Data Class Editor View menu, inclusions menu option. This improves compilation performance over the pre-5.2 releases by eliminating superfluous inclusions.
- **Windows User Interface**: Users now have the option of switching between the standard ObjecTime user interface, or the Windows look and feel. Please see the ObjecTime Developer *User Guide* to find out full details on how to do this.
- **Color preferences**: Users can now customize their environment colors. Specifically, the workspace color and graphics editor colors can be specified via a user preference.
- **Storage of environment configurations**: Environment configurations can now be stored separately from a model. This allows better control of environment configurations (for example, through versioning), and provides an easy mechanism for sharing configurations throughout a development group.
- **Large model toolset tuning**: New user preferences are provided to enable better user control when dealing with large models. These preferences allow for performance and memory utilization optimizations when working with large models which are more than 3 MB when passivated.
- **Online documentation search engine**: Online documentation now provides a search engine to make it easier to find information.
- **Customer Support website access**: The Customer Support restricted access website is now a single click away. A new menu item launches the user configured web browser, and prompts the user for the username and password to the support website. If you do not have a password for the Customer Support website, please check with your project's ObjecTime prime, or request it through customer support.

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 the 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.

Packaging Changes

The ObjecTime Developer product is available in 3 different product packages for the 5.2.1 product release. The ObjecTime Developer (Base), ObjecTime Developer for C++ and ObjecTime Developer for C make up the three product offerings.

ObjecTime Developer: Includes the toolset and Simulation Services Library components and replaces the product known as Modeler which was available with the ObjecTime Developer 5.1.1 release. It can be used for modeling and Simulation only.

ObjecTime Developer for C++: Includes the base ObjecTime Developer product package and also includes the C++ Target Services Library. It can be used for modeling, Simulation, and total code generation for C++ targets.

ObjecTime Developer for C: Includes the base ObjecTime Developer product package and also includes the C Target Services Library. It can be used for modeling, Simulation, and total code generation for C targets.

Optionally, you may choose to upgrade any one of the packages with support for one, or more, of the Target Services Libraries. This is supported through upgrades which can be applied to your base ObjecTime Developer or either of the language specific packages. For instance, owners of ObjecTime Developer 5.2.1 - C Target Module can optionally add the C++ Target Module to enable C++ code generation.

Developer WebPublisher

Developer WebPublisher 5.2.1 allows you to publish models in HTML format for viewing with either of the two standard internet browsers (Netscape 4.04 and Microsoft Internet Explorer 4).

> **Note:** This product is available for purchase and can be used with all three of the ObjecTime Developer 5.2.1 product packages. Please see the product information sheet enclosed with ObjecTime Developer 5.2.1 for details about Developer WebPublisher's capabilities.

Developer TestScope

Developer TestScope extends ObjecTime Developer's design-automation capabilities to model debug and test.

> **Note:** This product is available for purchase and can be used with all three of the ObjecTime Developer 5.2.1 product packages. Please see the product information sheet enclosed with ObjecTime Developer 5.2.1 for details about Developer TestScope's capabilities.

Installation Keys

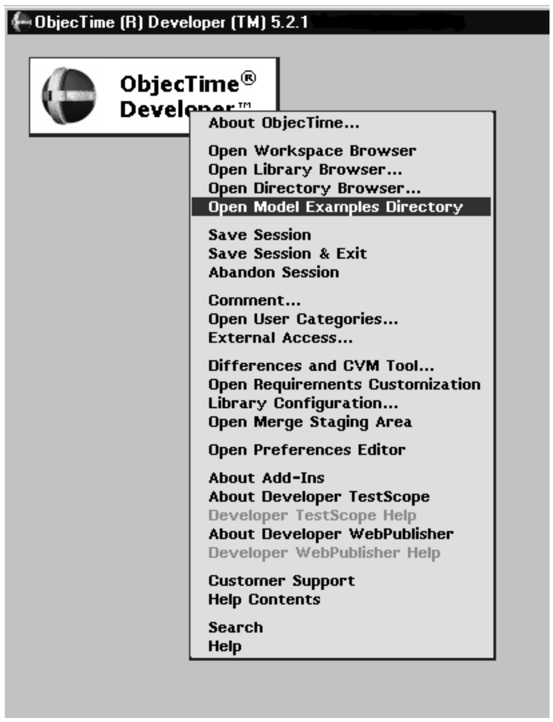
An envelope is included with your ObjecTime shipment which contains the installation keys necessary to install the software from the CD media. These keys are unique to your order and the envelope and contents should be kept in a safe place to facilitate future installations should a re-install become necessary. If these keys are misplaced please send a request for replacement keys to the ObjecTime Support e-mail address (support@objectime.com) providing your company name, project, and the ObjecTime product purchased.

Note: The installation keys are distinct from the license keys. You are required to obtain license keys from ObjecTime Support to run the software.

ObjecTime Model Examples

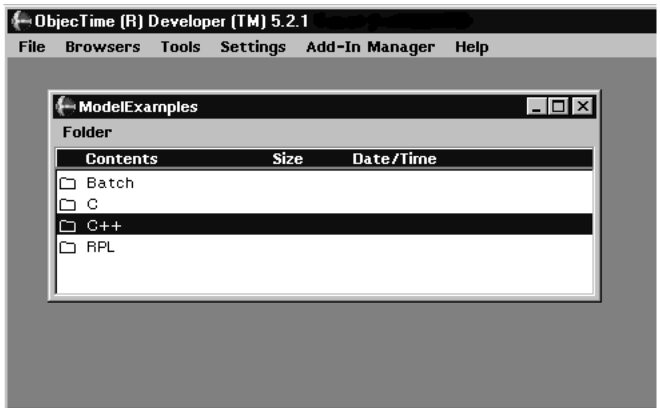
Included with the ObjecTime Developer base product are a number of example models which the users are encouraged to reference and build upon. Four different types of model examples are provided: RPL, Batch Mode, C++ and C. To access the model examples from an ObjecTime session using the ObjecTime Classic User Interface Mode, select the Open Model Examples Directory menu item which can be found in the main ObjecTime menu (see Figure 1).

Figure 1 Main ObjecTime Menu



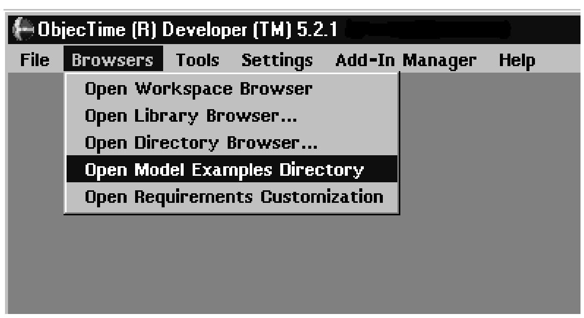
When the Model Examples Directory is opened, the following directory browser will be displayed, and the user will have the option of browsing one or more of the model example directories to activate the model example of choice (see Figure 2).

Figure 2 Model Examples Directory



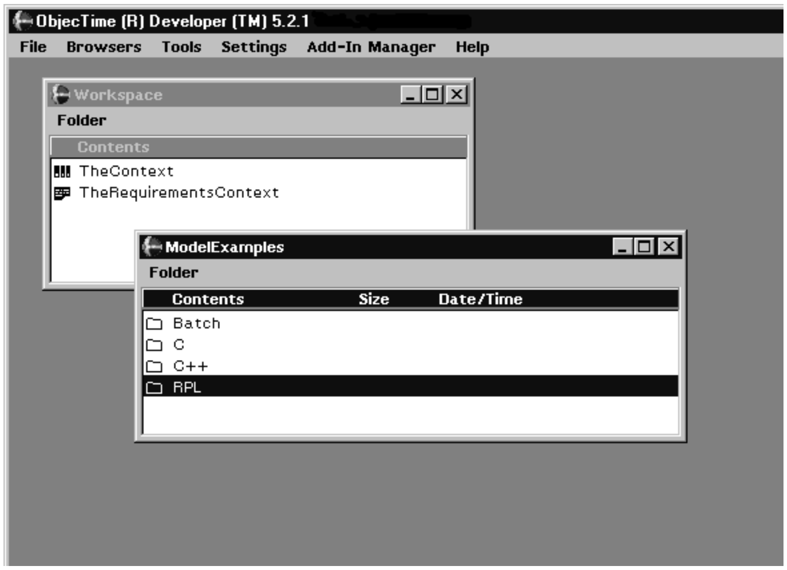
From an ObjecTime session configured as the ObjecTime Windows NT User Interface Mode, the model examples can be found through the Browsers pull down menu (see Figure 3).

Figure 3 Browser—Pull Down Menu.



As with the classic mode, when the Model Examples Directory is opened, a directory browser will be displayed, and the user will have the option of browsing one or more of the model example directories to activate the model example of choice (see Figure 4).

Figure 4 Directory Browser



Documentation Errata:

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

At the end of the Introduction, Chapter 1, page 14:

In the Event of Being “Locked-Out” From ObjecTime...

In certain rare conditions, you may find that ObjecTime does not respond to any user input. In this case you can try one of the following strategies in order to force a save of all your updates within the Workspace Browser as well as the ObjecTime session. Note that each of these files will be saved under a special name with the word “crash” in the filename.

Signaling the ObjecTimeVM process to save

This method will send a special signal to the running ObjecTime program instructing it to save all it’s updates.

In **Unix**:

1 Find the process ID of the running ObjecTime program. The program name is ObjecTimeVM. The following command can be used to determine the process ID for Solaris-based workstations:

```
ps -aux | grep ObjecTimeVM
```

2 Enter in the following command, where `pid` is the process ID determined in step 1:

```
kill -USR1 pid
```

In **Windows NT**:

Use the otsave.exe utility for this purpose. Start the ObjecTime Developer 5.2.1 Command Prompt and type `otsave`. You will be prompted with the process IDs (PID) of the ObjecTime Developer (s) running. Enter the appropriate PID.

Normally, ObjecTime will begin the save operation within a minute or so.

Using the ObjecTime Debug Mode to save

The following steps will bring up a special command window which allows you to save the work in progress:

1 Select the following keys simultaneously: `Control - Shift - C`

2 This will bring up the ObjecTime Debug Mode window. Now type `save`, and terminate by hitting the `Enter` key.

If this works, ObjecTime will immediately begin the save operation.

The emergency save operation described here can be time consuming depending on the number and size of the updates contained within your Workspace Browser. This operation also saves the image file and all of the updates contained in the image. These files can take up a large amount of disk space. If you do not have enough free disk space, the ObjecTime emergency save operation will not function properly and may result in some files being truncated.

Using the ObjecTime Debug Mode to refresh the screen

In rare cases, an ObjecTime session can get out of sync with the X-Window System in Unix. If your ObjecTime window stops refreshing properly, try the following steps:

1. Select the following keys simultaneously: `Control - Shift - C`
2. This will bring up the ObjecTime Debug Mode window. Now type `help`, and terminate by hitting the `Enter` key.

If this works, the ObjecTime window will start refreshing properly again. (PR 8398)

In Model Environment Setup, Chapter 23, page 387 under Parallel Flags add the following section:

Using an environment variable for Parallel Make flags.

Since compilation performance depends on the developer's compilation environment (and not on the update), it may be appropriate to make the parallel make flags field access an environment variable. For example, setting Parallel Make Flags to `$(OBJECTIME_PMAKE_FLAGS)` in the configuration. This Make macro can look-up an Environment Variable set by the user to different values depending on their compilation environment. Assuming the chosen flavor of Make supports the -j option for parallel building, a load-builder might do (in C-shell):

```
setenv OBJECTIME_PMAKE_FLAGS "-j 8"
```

while a developer might be satisfied with:

```
setenv OBJECTIME_PMAKE_FLAGS "-j 4"
```

and a user who does not set this environment variable will build serially. (PR 8993)

In Model Compilation and Execution, Chapter 22, page 371 after Guidelines/Hints add the following sections:

Parallelizing Clearmake

Parallel building is enabled for both code-generation and compilation on Unix platforms. (As of ClearCase 3.2.1, Clearmake does not support parallel building for NT.) However, use of the Clearmake concurrency environment variables CCASE_CONC and CLEARCASE_BLD_CONC is not recommended (see the section below).

It is adviseable to put $(OBJECTIME_PMAKE_FLAGS) in the Parallel Flags field of your configuration browser. This way the user (not the update) decides how much parallelization his compilation host(s) use.

The user must then assign some environment variables. For example:

```
setenv OBJECTIME_PMAKE_FLAGS "-J 4"
setenv CCASE_HOST_TYPE sun5
```

Then, create a build host file (`~/.bldhost.sun5`) as described in the ClearCase Reference manual (cf bldhost). For example:

```
-idle 10%
beef
helium
beef
beef
```

where the machine "beef" has significantly more processing power than the machine "helium".

Clearmake can now be invoked from the command-line, or by the toolset, without arguments, and the parallel flags will be correctly invoked for the Generation and Compilation Makefiles.

Using environment variables for Parallel Clearmake flags.

Clearmake (for Unix) supports the -J option for parallel distributed building. In addition to the scenario described in the above section (Using an environment variable for Parallel Make flags), the environment variables

CCASE_CONC

or, its longer version

CLEARCASE_BLD_CONC

can be set by the user to define the concurrency level if the -J option is not specified within the update itself.

However, because of the way ClearCase buffers and merges parallel build stream output, the progress messages of a ClearCase session are typically spooled until the entire build is finished. ClearCase users are encouraged to use a scenario as described in Using an environment variable for Parallel Make flags above. (PR 8993)

In Batch Mode, Appendix D, page 516, under ‘Command file syntax’ - ‘Commands’ replace the list of Commands with the following :

Commands

```
<commands>           ::=   ! <command> ! *
<command>1           ::=
```

log <text_string> |

1. The activate keyword is optional in order to remain compatible with the initial version of the batch mode grammar.

storageFormat `<format_options>` |

activate [`<update_file_name>`|`<context_file_name>` `<update_name>`]
`<actions>`* |

activateProject[`<full_path>` | `<file_name>` `<optonial_versions>` | `<actions>` |

logSeconds `<optional_comments>` |

logDateAndTime `<optional_comments>` |

saveSessionWhenDone|

abandonSessionWhenDone|

clearSignalMapping|

select `<update_name>` `<actions>`*

(PR8260)

In RTS Control Panel, Chapter 24, page 414, in the 'Trace Configuration Dialog' section replace the definition of States with the following :

States — when turned OFF no FSM states are in the trace. When turned ON included in the trace are changes in the FSM states of the actors as the states change.

In The Generic Library Interface, Appendix C, page 510, under 'objectime_diff' replace the sections titled Description and Input parameters (from toolset) with the following:

Description

Used to determine if the linear form files differ or not. The ignore option (-i) ignores reference to versions, for example, of contained actors, the class from which it is derived, and so on.

Input parameters (from toolset)

```
-I <imageLF_File>     #Required
-O <outputLF_File>    #Required
-i                    #Optional
```

In Differences and Class Version Merge Tool, Chapter 9, page 143 the following text should be deleted:

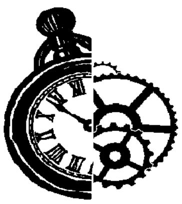
- the word 'semantic' in the first sentence

- the footnote (2) for Message Sequence Charts in the first sentence.

The sentence should now read:

The Differences and Class Version Merge tool allows you to determine the differences between two ObjecTime classes[1], Message Sequence Charts (MSCs), contexts, updates, or packages[2].

Chapter 2



Model Upgrade/Conversion

Model Conversion

If you are moving from ObjecTime Developer 5.2 to 5.2.1, no model changes are required.

Moving models forward, from pre-5.2 to 5.2.1, requires these models to be converted. Most conversions for an update will be automatic and will not require any user action beyond accepting that the conversion will take place. Once converted, the update will not activate with pre-5.2.1 versions of the toolset. Model conversion from pre-5.2 models is required to address the following:

- Dependencies between model components need to be explicitly captured in the stored class (Linear Form) files.
- Environment specifications on compilation unit packages (CUPs) must be preserved and converted to the new configuration specification supported by packages in 5.2.1.
- Detail level code must be changed to accommodate changes to the runtime system interface.

Automatic model conversion will be supported to move from 5.0, 5.1, 5.1.1 and 5.2 to 5.2.1. Moving from pre-5.0 model versions require conversion to 5.0 before the 5.2.1 conversion is attempted. All available universal patches should be installed on the pre-5.2.1 ObjecTime image and the model to be converted should be passivated with this image before the conversion to 5.2.1 is performed.

> **Note:** For 5.0 this includes universal patches 001 through to 088, for 5.1: universal patches 001 through to 033 and for 5.1.1: universal patches 001 through to 033, and for 5.2 : universal patches 001, 003-015, 017-030.

Adding Dependencies

Before a pre-5.2 model can be compiled in 5.2.1, the inter-class dependencies must be added. Without the dependencies, the inclusions in the generated code will be incomplete and model compilation will fail.

> **Note:** Model dependencies calculation can take several minutes on a large model.

When a pre-5.2 model is brought into the 5.2.1 toolset, the user will be presented with a list of classes for which dependency calculation should be performed. Accepting this will initiate dependency calculation and updating.

Model dependency calculation may not capture all needed dependencies and the user may have to manually complete the task. This can happen, for example, when a class reference is “hidden” behind a macro and only becomes evident when the macro is expanded. Since model conversion does not run a pre-processor on the detail level code, such cases will not be detected and a required dependency will not be added. When a dependency is not detected, the generated code will be missing include directives which will cause the compilation to fail. The error message and error mapping should make it readily apparent which class needs to be added to the dependency list of which other class. Dependencies are added manually with the properties editor of the class for which the dependency is to be added. The dependency can either be typed into the dependency pane or a reference can be dragged from the model browser into the dependency pane.

> **Note:** The new code generation feature has introduced a more disciplined approach to including header files in the generated code. As a result, “hidden” dependencies, for example, a data class only referenced in a transition code block, must be explicitly placed as a dependency for the class. If this is not done, then compilation errors will occur.

When bringing pre-5.2 model components, classes and packages, forward into 5.2.1, the dependency calculation part of the conversion process is not automatically invoked but must be initiated manually. This is because the toolset version which produced the component is not stored in the components source. After all the classes and packages have been merged into the 5.2.1 toolset, “Generate Class Dependencies” can be invoked from the update menu of the toolset. This will present the user with a list of classes which will be analyzed as part of the dependency calculation and updating.

Environment and CUPs conversions

Conversion of CUPs will be automatically initiated when a CUP or an update containing a CUP is brought into the toolset. The result of bringing a CUP into the toolset is that a set of package-level configurations (one for each language option) will be created and associated with the package.

In order for a package level confide to take effect, within an update, it must be associated with an update level confide. This is done from the properties editor of the package confide. The configuration field of the package confide property editor allows the entering of the update level confide with which this confide is associated. When the particular update-level confide is active, the corresponding package-level confide is also active.

If a package has only one or a few relevant configurations language options in pre-5.2, which are activated, then the user can create update level configurations for each language option and manually associate the corresponding package-level configurations with the update-level configurations. If there are:

- a large number of active configurations at various times,
- or a large number of packages,
- then performing the association between package and update level configurations would be very time consuming.

In this case, a special patch can be used to help automate the process.

To automatically associate a large number of packages or package configurations with update level configurations:

1 Merge the CUPs into the toolset.

2 Create a set of update-level configurations with the same names as the package-level configurations, which are to be associated with the update-level configurations.

3 Apply the patch **CUPConfigurationAssociation.patch** which will go over all the packages in the update, associating each package-level confide with the update-level confide of the same name.

This patch, **CUPConfigurationAssociation.patch,** is located in $OBJECTIME_HOME/specials, can be applied to the toolset by dragging it onto the workshops browser of the toolset.

After the model has been converted by activating the patch, the update should be passivate and the session abandoned. This is to ensure that the development image does not contain the conversion patch.

Detail level code changes

Some changes in pre-5.2 detail level code may be required in order to get the model being converted to compile with the 5.2.1 runtime systems (Simulation, C++ or C). Changes will be required if the model has made use of interfaces which had changed in 5.2. For a list of run-time system changes, see the Getting Started and Release Notice for the appropriate language.

- In pre-5.2 versions of the toolset, actor detailed-level code had access to all the signals defined by all the protocols used in a model. In 5.2/5.2.1, the signals names automatically available in the detailed-level code are those which are defined by the protocols of the ports of an actor. This means that detail-level code which made use of signals not defined in protocols on the actor, will no longer compile. If the code references a signal which is not defined in any of the protocol classes referenced by the actor's ports and SAPs, then the code can be made to compile by adding the protocol to the dependency list of the actor.
- Data classes do not have automatic access to any signals and must have explicit dependencies added in order to reference signal names.
- In previous releases, if a user code segment did not terminate in a semi-colon, one was automatically added. This feature has been removed in ObjecTime Developer 5.2/5.2.1.

After model conversion has been completed, it is advisable that all update level inclusions are reviewed to determine if they can now be moved either to the package or class level. In the past, inclusions for data classes had to be specified at the update level that resulted in all classes in the update having this inclusion, regardless of whether it was required or not. With 5.2/5.2.1, data classes now support inclusions and some update level inclusions can be moved to the data class level. Moving the inclusions is not necessary for correct compilation, but can make a significant improvement to the compilation performance.

Removing PWD from Inclusion Paths

"$(PWD)" can probably be removed from your update's Inclusion Paths, if it exists. This inclusion path was typically added automatically in previous releases of ObjecTime Developer. It is unlikely that your project depends on this path for compilation, and potentially dangerous if it does. The interpretation of

"$(PWD)" depends on the implementation of your Make executable, since Make is initially invoked in a directory different from where the compiler is invoked.

Furthermore, Clearmake users will want to remove references to "$(PWD)" from their Inclusion Paths because it makes all compiled files dependent on the expansion of PWD. Failure to remove such references will cause an unnecessary recompile of all files whenever the definition of PWD changes, such as when building from the command-line versus building from the toolset.

Chapter 3

Getting Started with Windows NT

This chapter describes how to install, configure and begin to use ObjecTime Developer 5.2.1 with Microsoft Windows NT. This document assumes that the user has a basic understanding of how to use and administer Windows NT.

The main steps involved in getting started with Windows NT are described in the following sections:

- **Installing a Browser** (“Installing Netscape Navigator” on page 23 or “Configuring for use with Internet Explorer 4.0” on page 24) covers how to set up a browser for viewing the on-line help and documentation. If Navigator or Explorer 4.0 is already installed on your system, this step can be skipped.
- **InstallingObjecTime Developer 5.2.1** (“Installing ObjecTime Developer 5.2.1” on page 26) covers how to install ObjecTime Developer 5.2.1 software on a Windows NT workstation or server.
- **Setting up a User Workstation** (“Setting Up a User Workstation” on page 34) describes how to set up a Windows NT workstation from an existing ObjecTime Developer installation on a central network file server. This step is not required if you are installing ObjecTime Developer 5.2.1 on a local workstation.
- **Starting ObjecTime Developer 5.2.1** (“Starting ObjecTime Developer 5.2.1 on Windows NT” on page 36) describes how to start the Developer 5.2.1 toolset using the ObjecTime Developer Launcher.

Network vs. Local Installation

Two scenarios are available when installing ObjecTime Developer 5.2.1 on Windows NT. You can install Developer on a local workstation disk, or you can install Developer on a central network file server.

Each scenario has advantages: Network installations can be shared between multiple users at a single site, reducing the amount of local disk space required on each workstation, centralizing administration and maintenance, and reducing the effort required to upgrade multiple users. On the other hand, local installation can provide a significant performance advantage, especially with slower network configurations.

During the setup process you will be asked to select a destination directory for the Developer files. To create a local stand-alone workstation installation, select a destination directory on a local disk.

To create a shared network installation, select a destination directory on a shared network disk. After installing the Developer files on the network disk, run Setup from each network workstation and perform a “User Setup,” as described in “Setting Up a User Workstation” on page 34.

Supported Network Configurations

ObjecTime Developer 5.2.1 can either run locally on a Windows NT workstation, or through a Windows NT file server under the following configuration conditions:

- The network must use Microsoft networking, with TCP/IP enabled.

Mixed Unix and Windows NT Installation

ObjecTime Developer 5.2.1 can run on a Windows NT workstation connected to a Unix file server under the following configuration conditions:

- Network file system must be NFS.
- Supported NFS clients are **Chameleon** and **Hummingbird**. Make sure to install the clients properly:
  - Support for mixed case file names must be enabled.
  - Consult the NFS client documentation regarding soft links. Some implementations can’t handle these very well.
- The path to the Setup program must conform to the 8.3 DOS file naming convention, and the path cannot be longer than 63 characters including drive letter and the name of the Setup program.

Installation Requirements

- **Windows NT** – Windows NT 4.0 (Workstation or Server) is required to install ObjecTime Developer 5.2.1. Windows NT (Build 1381 - Service Pack 3), versions prior to version 4.0 are supported as network file servers only.
- **CD-ROM drive** – A CD-ROM drive is required to install Developer 5.2.1 from CD. If a CD-ROM drive is unavailable, copy the contents of the disk to a network file server, map the network disk to a drive letter, and perform the installation from the mapped network location.
- **Administrators Group Membership** – Membership in the Administrators group is required to set up Developer 5.2.1. Refer to the Windows NT documentation on how to assign those privileges.
- **50 MB to 64 MB free disk space –** A minimum ObjecTime Install requires 50 MB of free disk space, and a full installation requires 64 MB. 100 KB of disk space is required for fonts on the drive where Windows NT system files are located, and an additional 300 KB is required if the License Manager is installed.
- **Printer** – The default printer requirement is, at minimum, a Windows NT compatible printer. The recommended printer is a PostScript™ printer.

File System Requirements

- **File names** – The code generation process in ObjecTime Developer makes use of long file names with mixed-case characters. The file system where the Developer software is installed and where Developer working directories are saved must support this type of file name convention for Developer to function properly. File names containing spaces are not supported.

Note: Directory names are subject to the same limitations as file names.

- **Native file systems on Windows NT 4.0**: Developer 5.2.1 supports both FAT and NTFS file systems.
- **NFS –** Developer 5.2.1 supports the use of NFS file systems for network installations and Unix compatibility. If you choose to use NFS, be sure to configure the NFS software such that the case of file names is preserved when saving files to NFS disks.

  Note that while a Windows NT workstation can be set up to use a Developer installation on a NFS disk, the Windows NT setup program cannot perform an installation to a NFS location. To create a shared network installation of Developer on a NFS disk, first install Developer from a Unix workstation using the Unix setup program, and then continue with “Setting Up a User Workstation” on page 34.
- **UNC path names** – Developer 5.2.1 does not support UNC path names (that is, Network Neighborhood path names) for network resources. To use a network resource, map the desired resource to a drive letter.

Local Workstation Requirements

- **Windows NT 4.0 –** Windows NT 4.0 (Workstation or Server) or later is required to run ObjecTime Developer 5.2.1. ObjecTime recommends that you use Windows 4.0 (Build 1381 - Service Pack 3).
- **Pentium processor –** A Pentium Pro or Pentium II processor is recommended for improved performance.
- **64 MB main memory minimum –** For large models or build operations, 128 MB or greater is recommended for improved performance.
- **Toolset Memory Requirements –** “Toolset Memory Requirements” on page 115 describes the memory requirements for models.
- **12 MB disk space per working directory minimum –** Each user may have one or more Developer 5.2.1 working directories which contain the user’s session file. Developer session files are initially 12 MB and will increase in size with use. (Code generation, compilation, passivation, and so on will add to the space required.)
- **256 color graphics adapter –** A high resolution graphics adapter with support for more than 256 colors is recommended.

Installing Netscape Navigator

Online help and documentation for Developer 5.2.1 is provided in HTML format. In order for the help system to function correctly, Netscape Navigator, or alternatively Microsoft’s Internet Explorer, must be installed on the user’s system. If Navigator 4.04, or Explorer 4.0, is already installed on your system, this step may be skipped.

Note: Netscape Navigator is provided with the release as part of the on-line help system. You are licensed to install one copy of Navigator 4.04 per licensed copy of ObjecTime Developer 5.2.1. Please refer to the Netscape license agreement for terms and conditions.

The following procedure describes how to install Netscape Navigator 4.04 for Windows NT.

1 If necessary, **load the ObjecTime Developer 5.2.1 CD** into the CD-ROM drive.

   Note: If the system is configured with the autorun feature enabled, the ObjecTime Developer 5.2.1 Setup program will run automatically. Click "Cancel" to exit Setup.

2 **Locate the Netscape Navigator Setup program**

   Netscape Navigator software is provided on the ObjecTime Developer 5.2.1 CD in the `Netscape\Windows\Setup\Win32` directory. Use the Windows Explorer to open a window displaying the contents of the Netscape folder. For example, double-click "My Computer", double-click the icon for the ObjecTime CD, and continue opening the "Netscape", "Win" and "32bit" folders by double-clicking the appropriate icons.

   Note: If the system is configured with the autorun feature enabled, double-clicking the icon for the ObjecTime CD will automatically run the ObjecTime Developer Setup. To display the contents of the CD, right-click on the CD icon and select "Open."

3 **Read the Netscape Readme**

   Please refer to the Netscape `Readme.txt` file for installation notes and for any platform specific installation instructions before proceeding with Setup. Double-click on the `Readme.txt` icon to view the file. (Note that the `.txt` file name extension may not be visible.)

4 **Run Netscape Navigator Setup**

Double-click the `Setup.exe` icon to run Setup. (Note that the `.exe` file name extension may not be visible.)

Configuring for use with Internet Explorer 4.0

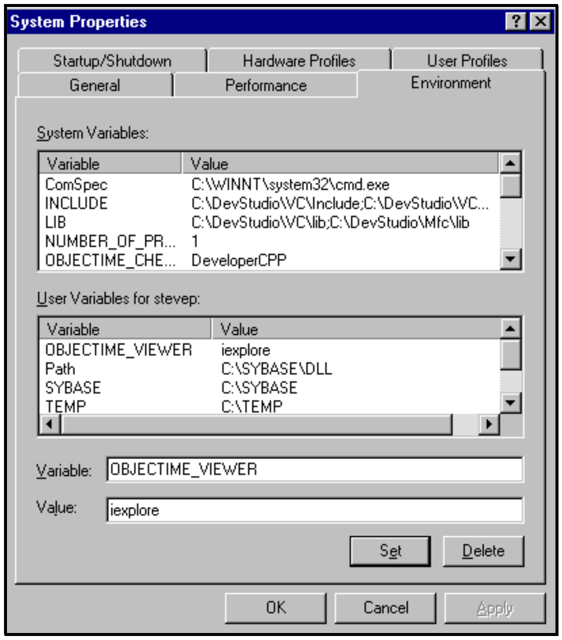
The ObjecTime Help system is configured by default to use Netscape when opening the HTML help pages. It is possible to configure ObjecTime to use any HTML browser which supports an appropriate Application Programmer Integration (API). In particular, it is possible to configure ObjecTime to use Internet Explorer for the help system. It should be noted, however, that ObjecTime's use of anything except Netscape is not fully supported by ObjecTime Limited. Although ObjecTime has tried to offer a system flexible enough to support various browsers, problems or errors stemming from the use of other browsers are outside the scope of ObjecTime Support.

ObjecTime's method to support other browsers makes use of a batch file which is passed parameters indicating the path of the file to open and the tag to search for in the file. In order to use this script, Netscape must be uninstalled from the machine which is running ObjecTime (this is necessary, because ObjecTime first looks for Netscape on the host machine, and if there, will start it up. If it does not find Netscape, ObjecTime will then run the ObjecTimeStartHelp batch file).

There are two methods for changing the help browser behavior in ObjecTime. One is to set the OBJECTIME_VIEWER environment variable to the executable name of the browser you wish to use, and the other is to modify the batch file %OBJECTIME_HOME%/bin/winnt4/ObjecTimeStartHelp.bat to use internet explorer.

Example 1:

Set the OBJECTIME_VIEWER environment variable to ‘iexplore’ in the ‘System Properties’ window under the 'Environment' tab.



Example 2:

Modify the ObjecTimeStartHelp.bat batch file by changing ‘netscape’ to read ‘iexplore’.

Installing ObjecTime Developer 5.2.1

Note: If you have a previous release of ObjecTime Developer installed, you can:

- either uninstall it prior to installing ObjecTime Developer 5.2.1
- or, install it after installing ObjecTime Developer 5.2.1

following the procedure outlined in “Uninstall of “old” ObjecTime Release causes run failure of 5.2.1 in the following ways:.” on page 130 of this document.

A wizard-style setup program is provided to facilitate installing ObjecTime Developer 5.2.1 on Windows NT. The setup program can perform two types of procedures:

- “User Setup” will configure a user’s workstation to run Developer from an existing central network installation of ObjecTime Developer. See “Setting Up a User Workstation” on page 34 for details.
- “ObjecTime Install” will install the Developer files to either a local workstation disk or a shared network disk, and will configure the local workstation so that it is ready to run ObjecTime Developer. This procedure should be used to create either a stand-alone workstation installation, or a central network installation on a shared NTFS disk. This procedure is described below.

1 Load the ObjecTime Developer 5.2.1 CD into the CD-ROM drive

Note: If the system is configured with the autorun feature enabled, the setup program will run automatically and you may continue with step 3.

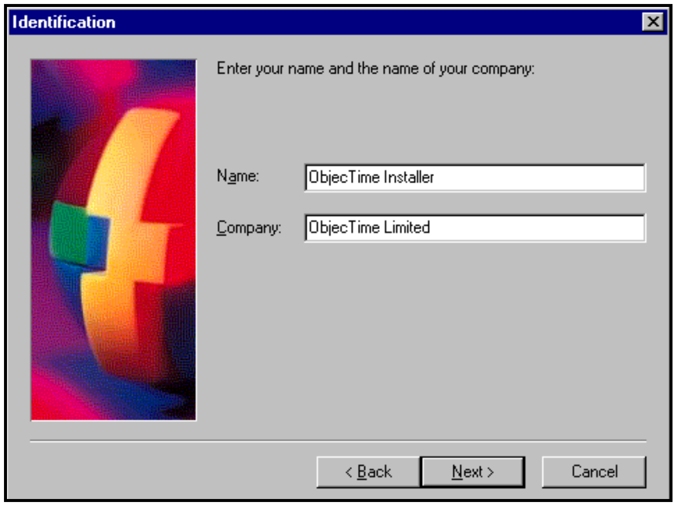
2 Run Setup

Use the Windows Explorer to open a window displaying the contents of the CD-ROM drive. For example, double-click “My Computer” and then double-click the icon for the ObjecTime CD. Double-click the `Setup.exe` icon to run Setup. Note that the `.exe` file name extension may not be visible.

If Nestcape Navigator is not installed on your system prior to installation, the following warning message will be displayed during the installation, at which point you can either abort the installation and install Netscape, or continue with the ObjecTime installation and install Navigator 4.04 or Internet Explorer 4.0 after the ObjecTime installation has been completed.:

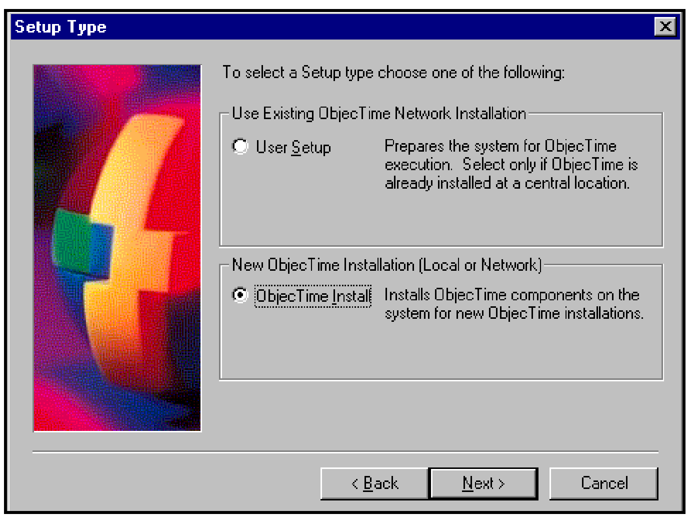
3 Identification Information

You must accept the license agreement to proceed. If you do not agree with the terms of the license agreement, the installation should be aborted and all software and documentation should be returned to ObjecTime Limited. if you accept the terms and conditions of the license agreement, you must identify yourself and the company you represent.



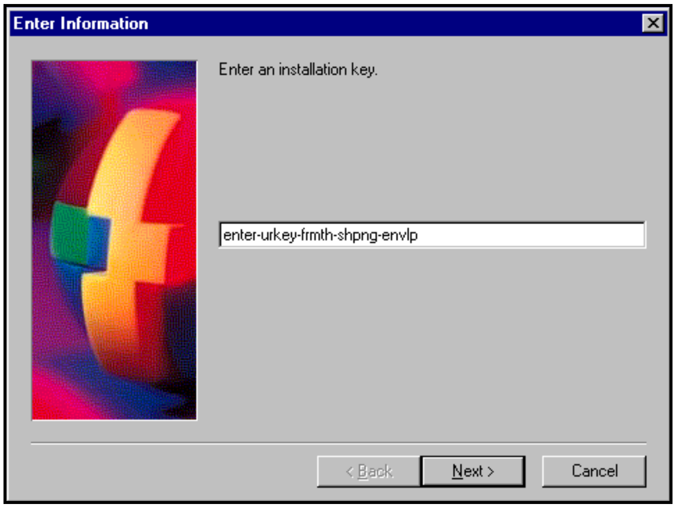
4 Select "ObjecTime Install"

After reviewing the license agreement and entering identification information, you will be prompted with the "Setup Type" dialog. Select "ObjecTime Install."



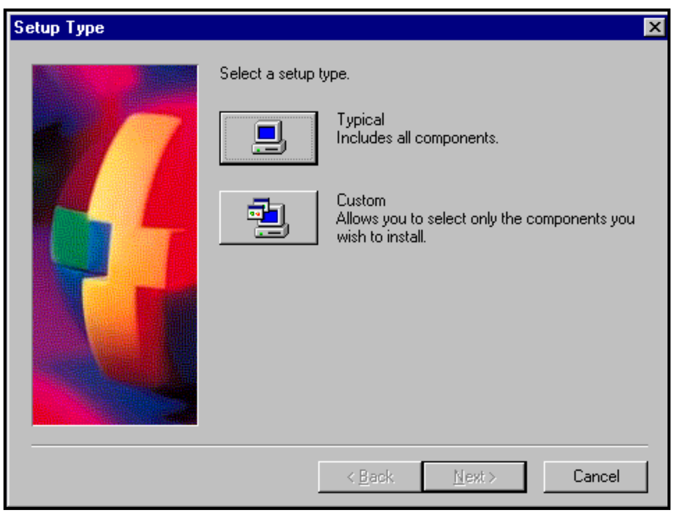
5 Enter Installation Keys

Locate the Installation Key letter which you had received with your ObjecTime software media shipment. You will be required to enter these keys to install the appropriate ObjecTime software packages.



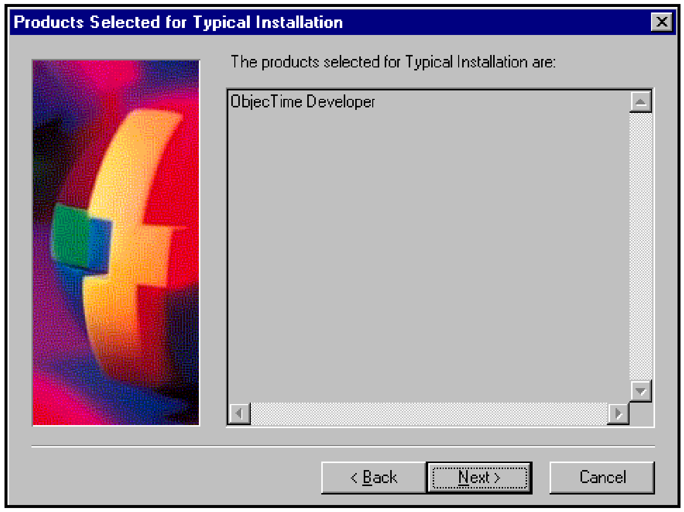
6 Select the Set-Up Type

The default set-up type of Typical should always be used unless you are working with ObjecTime Support on installation or packaging issues.



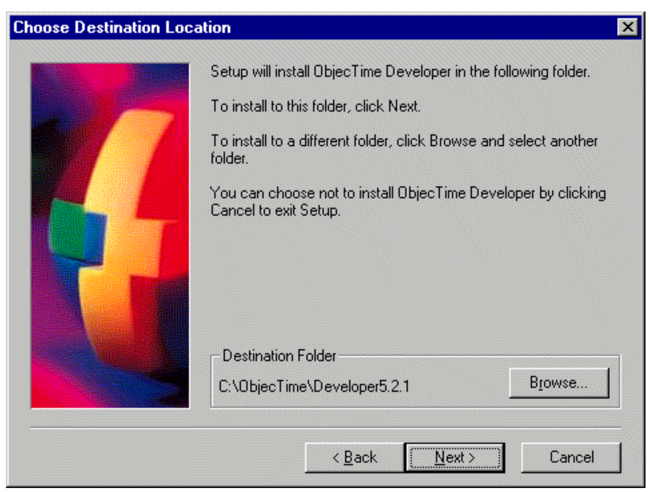
7 Product Package Confirmation

The product packages which will be installed are displayed and you must acknowledge these by clicking on the Next button. These packages will match the products which were ordered through ObjecTime as identified by your installation keys.



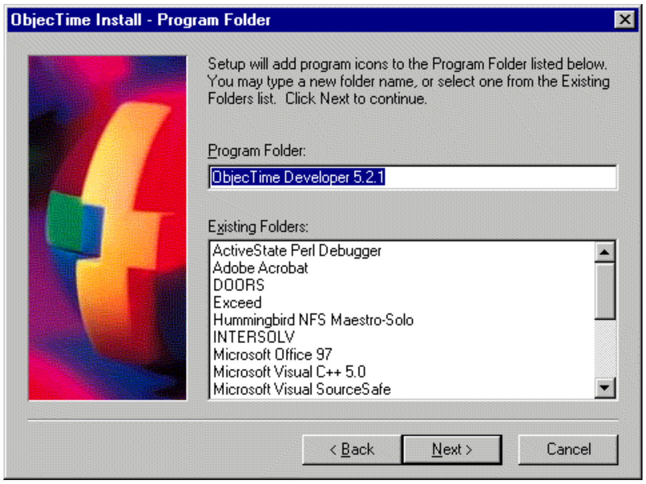
8 Select a Destination Directory

The destination directory is the file system location where the main Developer files are copied. The default location is `C:\ObjecTime\Developer5.2.1`, where `C:` is the drive where Windows NT is installed. ObjecTime recommends that you select the default installation location. If you choose to change the installation location, refer to the limitations described in "File System Requirements" on page 22.



9 Select a Program Folder

A program folder must be identified to which the ObjecTime icons are added. The default is ObjecTime Developer 5.2.1.



10 Setup Options

After entering the Program Folder name you will be asked to select the initial product setup options. These options are user-specific, and may be modified later by selecting the “Preferences” button from the ObjecTime Developer Launcher. The following options are available:

License Manager: The location of the license manager must be specified in order for ObjecTime Developer to run. The license manager may be installed on the local workstation, or the system can be configured to use a license manager that is available on a remote network workstation. Only one license manager needs to be installed on the network.

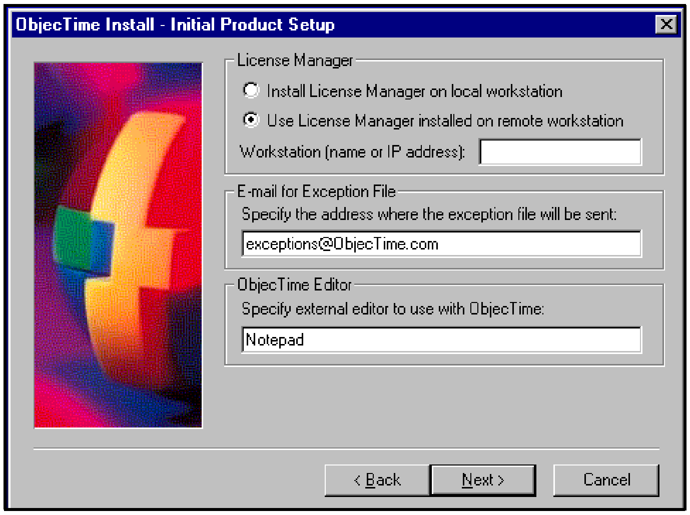
If you select to “Install the License Manager on the local workstation”, you must follow the procedure described in the “License Manager Operations” chapter after completing the installation in order to configure and start the license manager.

Note: If you choose to run the license manager locally, set the TZ (**Time Zone**) variable. Install presents a warning if TZ is not set. See “Setting the Time Zone Variable on Windows NT” on page 64.

If you select to “Use a License Manager installed on a remote workstation”, you must specify the network IP address or host name of the workstation on which the license manager is installed.

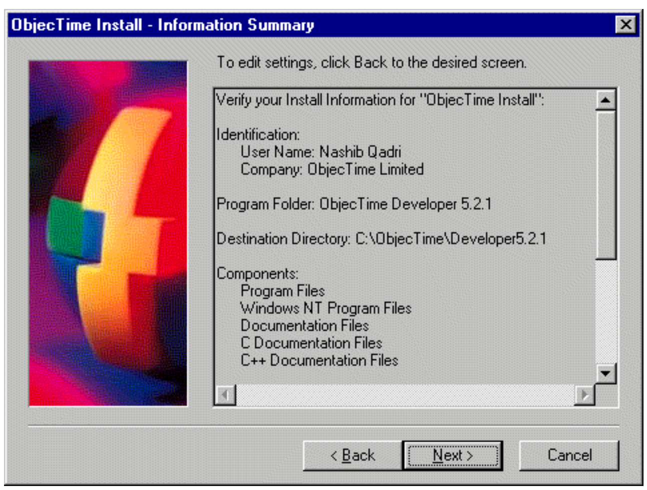
E-mail for Exceptions File: If the toolset encounters a problem and an exception is generated, an exception file is created and mailed to the e-mail address specified. The default address is “exceptions@ObjecTime.com.” If the address is blank, the exception file will not be sent out. This e-mail address is also used when mailing comments to ObjecTime support. It is recommended that exceptions be sent to the default address unless you have an internal support group that assists with such problems.

External Editor: ObjecTime Developer supports the use of a user-specified text editor for editing detailed source code in the toolset. Use this option to specify the full path of the editor executable.



11 Confirm Set-Up

You will be asked to confirm the installation parameters at which point the installation process will begin.

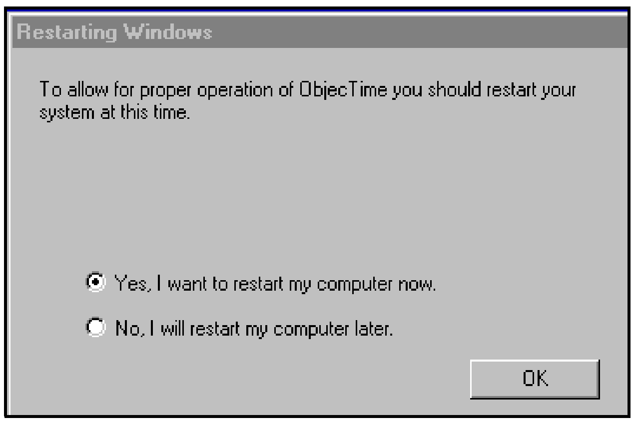


Review the setup options carefully and click “Back” to make any modifications. Click “Next” to begin the installation.

After installation is completed, you will be prompted to read the `Readme` file, which contains last minute additions to the release notes. Review the `Readme` file by clicking on the “Yes” button.

12 Restart System

It is strongly recommended that you restart the system to complete the installation.



Obtain License Keys.

When **upgrading from 5.2 to 5.2.1:**

- after installing the software, activate your **5.2** License Keys.

For a **new 5.2.1** license or when **upgrading from pre-5.2:**

- after installing the software, obtain License Keys from ObjecTime Support to run the software. See “License Manager Operations” on page 55 for complete instructions.

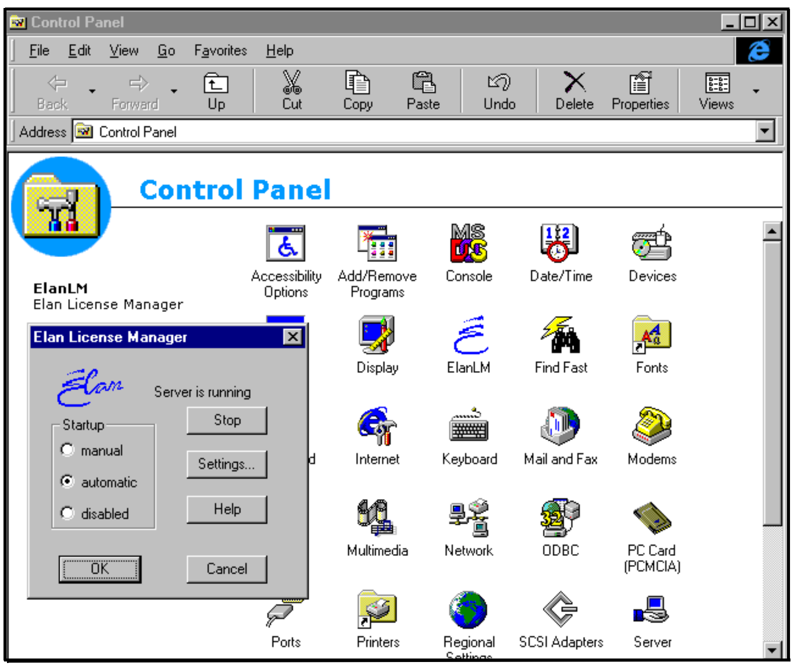
Uninstalling Developer 5.2.1

ObjecTime Developer 5.2.1 may be uninstalled by selecting the "Uninstall ObjecTime Developer 5.2.1" icon from the "ObjecTime Developer 5.2.1" folder created by the setup program.

If the Uninstall icon is not present, open the "Control Panel," double-click on "Add/Remove Programs," select "ObjecTime Developer 5.2.1" from the list of applications, and click "Add/Remove" to uninstall ObjecTime Developer.

The uninstall utility does not remove ObjecTime fonts that are installed by Setup. To manually remove ObjecTime fonts, open the "Control Panel," open the "Fonts" folder, and delete the ObjecTime fonts (otl10b, otl10i, otl10r, otl10s, otl10t.)

If the license manager is running locally on the system, the uninstall procedure does not remove the License Manager service from the system. The license manager can be disabled by the user through the Elan LM applet of the system control panel.



To remove the license manager manually, remove the files objectime_elmd.exe and objectime_elmd.cpl which are the license manager executables and control panel respectfully. These files are located in C:\winnt\system32.

> **Note:** On systems where ObjecTime Developer has been installed incrementally, the uninstall utility may fail to completely remove all the files that were installed. To remove the remaining files, delete the ObjecTime Developer 5.2.1 installation directory manually.

Setting Up a User Workstation

The following procedure describes how to set up a user workstation so that it can run ObjecTime Developer 5.2.1 from an existing network installation. If you are installing a stand-alone workstation, this procedure is not required. For more information on network installations, see “Network vs. Local Installation” on page 21.

To complete this procedure, ObjecTime Developer must already be installed on another workstation or server at a location that is accessible over the network. The ObjecTime Developer installation may either be on a NTFS disk on a remote Windows NT workstation, or it may be on a NFS disk on a remote Unix workstation. If you intend on using ObjecTime Developer over an NFS disk, you must have the appropriate NFS software installed and properly configured on the local workstation. See “File System Requirements” on page 22 for more information.

> **Note:** The network location where the central ObjecTime Developer installation resides **must** be mapped to a drive letter before proceeding with a “User Setup.” For NFS installation, it is recommended that the `Developer5.2.1` directory containing the ObjecTime Developer files be mapped directly to a drive letter. For example, map `\\server\appl\ObjecTime\Developer5.2.1` to the drive letter `D:`.

To map a network drive, right-click on “My Computer” and select “Map Network Drive.” Select an unused drive letter to map, browse to the desired network path, and click “OK.” Refer to the Windows NT documentation for more information on mapping network resources.

The following steps for setting up a user workstation as a client of a network server is similar to the steps involved in an “ObjecTime Install.” However, the only files that are installed on the local workstation with this type of setup are the ObjecTime font files.

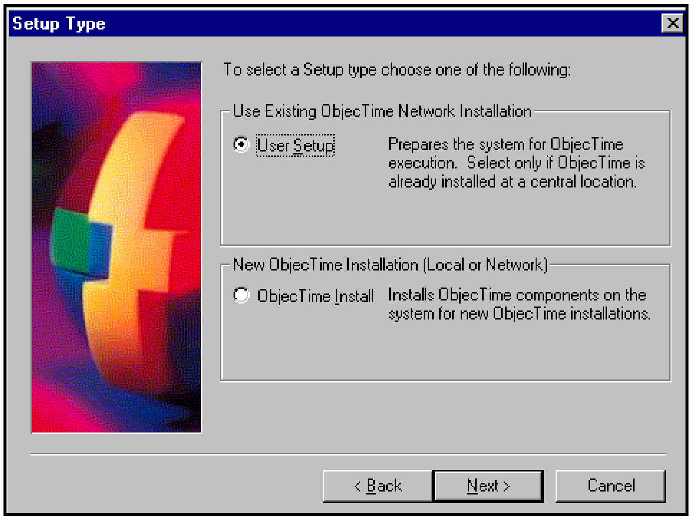
1 Run Setup

Use the Windows Explorer to open a window displaying the contents of the `NTSetup` subdirectory of the central ObjecTime Developer installation. For example, if the central ObjecTime Developer installation is located at `D:\`, double-click “My Computer”, double-click the icon for disk `D:`, and then double-click the `NTSetup` folder. Double-click the `Setup.exe` icon to run Setup. Note that the `.exe` file name extension may not be visible.

Note: You can also perform a “User Setup” by running ObjecTime Developer Setup from the Developer 5.2.1 CD, but you will need to manually specify the network location of the ObjecTime Home directory in step 3 below.

2 Select “User Setup”

After reviewing the license agreement and entering identification information (see items 1 through 3 in the Standard ObjecTime Install), you will be prompted with the “Setup Type” dialog. Select “User Setup.”



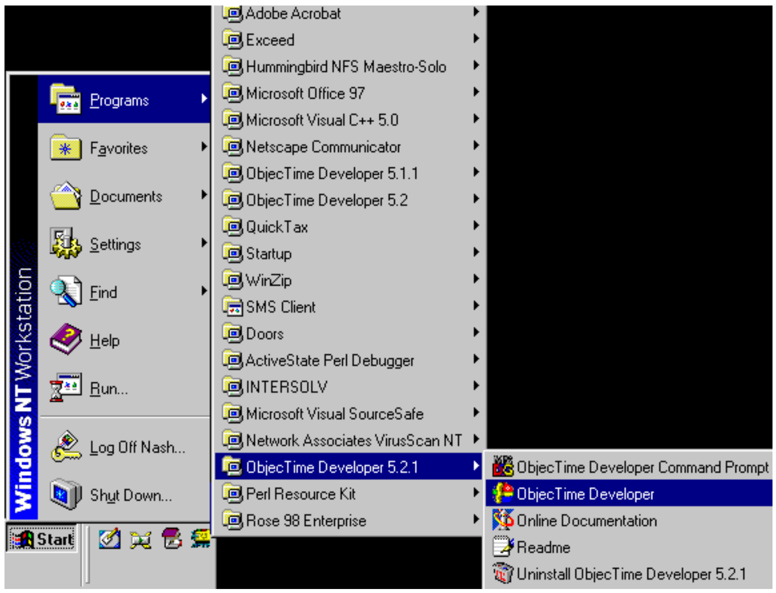
3 Specify the ObjecTime Home Directory

The ObjecTime home directory is the location where the existing network installation of ObjecTime Developer resides. For the above example, the ObjecTime home location is `D:\`.

4 Follow steps 9 through 12 as in the standard ObjecTime Install scenario.

Starting ObjecTime Developer 5.2.1 on Windows NT

Installing Developer 5.2.1 on a Windows NT workstation will create an “ObjecTime Developer 5.2.1” folder containing several shortcut icons.



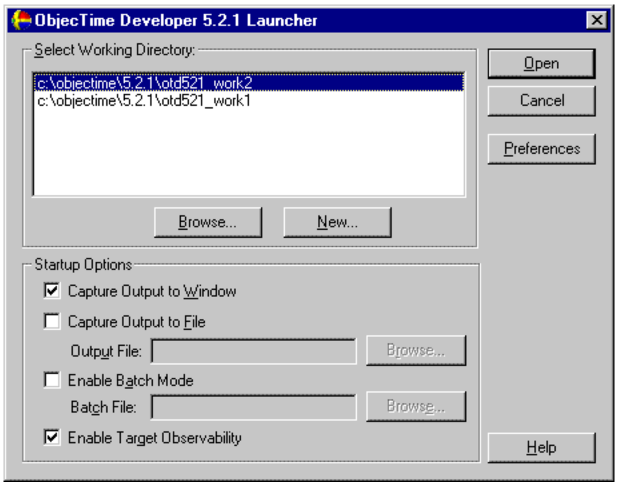
This folder is added to the “Programs” entry of the system’s “Start” menu and the following icons are created:

- **ObjecTime Developer** starts the ObjecTime Launcher program. The Launcher is used to create working directories, configure start-up options, and to start the ObjecTime Developer toolset with a specific working directory. For details, see “Using the ObjecTime Developer Launcher” below.
- **ObjecTime Developer Command Prompt** starts a Windows NT command prompt window configured with the required environment variables for running the ObjecTime Developer command line utilities. All Developer utilities including license manager scripts and Target Services Library build operations should be executed from this window. Refer to the chapter on “License Manager Operations” for further details on the License Manager utilities.
- **Readme** displays release notes containing important up-to-date information not included in the printed documentation.
- **Online Documentation** displays the on-line version of the printed documentation.
- **Uninstall ObjecTime Developer 5.2.1** will uninstall Developer from the workstation. See “Uninstalling Developer 5.2.1” on page 33.

Note: You can start an ObjecTime Developer toolset session for a specific working directory by double-clicking on the `ObjecTime5.2.otd` session file located in the working directory.

Using the ObjecTime Developer Launcher

ObjecTime Developer sessions are saved in "working directories." Working directories contain the Developer 5.2.1 session file (`ObjecTime5.2.otd`) plus any other files that are generated while you use the toolset. For example, when you use the toolset to generate source code and build executables for your models, the session's working directory is the default location where the generated files will be saved.



Although it is possible for a user to have just one working directory, it is usually more convenient to use more than one. For example, you may choose to have a separate working directory for each project that you are working on.

The ObjecTime Developer Launcher provides an interface for creating new working directories and for configuring and starting toolset sessions.

- **Working Directories:** This list box provides access to the most recently used working directories. To start the Developer toolset, select a working directory from the list and click "Open." If you haven't used Developer before, this list will be empty.
- **New:** This button allows you to create new working directories. A dialog appears where you can specify the location for the new directory. Enter the full path name for the new directory and click "Create" to create it. When finished, the new directory name will be added to the Working Directories list.

  Note: The file name limitations described on page 22 also apply to Developer 5.2.1 working directories.
- **Browse:** This button allows you to add an existing Developer 5.2.1 working directory to the Working Directories list.

  Note: You cannot specify an arbitrary existing directory as an ObjecTime Developer working directory. The directory specified must contain a valid Developer session file (`ObjecTime5.2.otd`).

Startup Options

The launcher allows you to specify a number of startup options when starting a toolset session.

- **Capture Output to Window:** If selected, console output from the Developer toolset is displayed in a window. This option is selected by default. Either "Capture Output to Window" or "Capture Output to File" must be selected.
- **Capture Output to File:** If selected with a valid file name specified, console output from the Developer toolset will be saved in the specified file.
- **Enable Batch Mode:** If selected, the launcher will start the toolset in batch mode. Specify the file containing the batch mode commands. Refer to the *User Guide* for further information on Batch Mode.
- **Enable Target Observability:** If selected, the launcher will start a **RTS controller** before loading the toolset session. The toolset will automatically be connected to the running RTS controller to enable the target observability feature. When the toolset session ends, the RTS controller is automatically stopped. Refer to the ObjecTime Developer *User Guide* for further information on target observability.

Command Line Parameters

The ObjecTime Developer Launcher executable is called `ObjecTime5.2.exe`, and is located in the `bin\winnt4` subdirectory of the ObjecTime home directory. The default location is:

```
C:\ObjecTime\Developer5.2.1\bin\winnt4\ObjecTime5.2.exe
```

where `C:` is the drive where Windows NT is installed.

The launcher can be started from the ObjecTime Developer Command Prompt, or from a shortcut icon, with the following command-line parameters. These parameters can be useful for automating batch mode sessions or for setting up shortcut icons with frequently used startup options.

- **`<workingDir>\ObjecTime5.2.otd`** allows you to specify a working directory to automatically load. If a valid working directory is provided, the launcher interface will not appear but will immediately start the toolset with the session file in the specified working directory.
- **`-verbose`** is equivalent to selecting the "Capture Output to Window" option.
- **`-verbose=<filename>`** is equivalent to selecting the "Capture Output to File" option.
- **`-file=<filename>`** is equivalent to selecting the "Enable Batch Mode" option.
- **`-control`** is equivalent to selecting the "Enable Target Observability" option.
- **`-console`** is equivalent to starting an "ObjecTime Developer Command Prompt."

Specifying Additional Environment Variables

The Windows NT Developer toolset recognizes the same environment variables as the Unix toolset. You can use `.bat` files to specify the desired environment variables, and then automatically start the toolset session by invoking the Launcher with the appropriate parameters. Example:

```
set USER_MAKE_FLAGS= -j4
ObjecTime5.2.exe C:\OT52\ObjecTime5.2.otd -control
```

Running a batch file containing the above commands from an ObjecTime Developer Command Prompt will automatically start a toolset session with the `C:\OT52\ObjecTime5.2.otd` session file, Target Observability enabled, and with the compile environment variable `USER_MAKE_FLAGS` set to `-j4`.

Chapter 4

Getting Started with Unix

The procedure for installing ObjecTime Developer 5.2.1 in Unix is described in the following section. Note that unless specified otherwise, your system administrator will generally carry out the following steps.

For environments where there is more than one user of ObjecTime Developer 5.2.1, we strongly recommend that the main ObjecTime Developer 5.2.1 files be installed on a centralized file server.

The main steps involved in getting started with UNIX are described in the following sections:

- **Installing Netscape Navigator** (“Installing Netscape Navigator” on page 43) covers how to set up Netscape Navigator for viewing the on-line help and documentation. If Navigator is already installed on your system, this step can be skipped.
- **Installing ObjecTime Developer 5.2.1** (“Installing ObjecTime Developer 5.2.1” on page 45) covers how to install ObjecTime Developer 5.2.1 software on a Unix workstation or server.
- **Setting up a User Workstation** (“Setting Up a User Workstation” on page 47) describes how to set up a Unix workstation from an existing ObjecTime Developer installation on a central network file server. This step is not required if you are installing ObjecTime Developer 5.2.1 on the local workstation.
- **Starting ObjecTime Developer 5.2.1** (“Starting ObjecTime Developer 5.2.1” on page 50) describes how to start the Developer 5.2.1 toolset.

Network vs. Local Installation

Two scenarios are available when installing ObjecTime Developer 5.2.1 on Unix. You can install ObjecTime Developer on a local workstation disk, or you can install ObjecTime Developer on a central network file server.

Each scenario has advantages: Network installations can be shared between multiple users at a single site, reducing the amount of local disk space required on each workstation, centralizing administration and maintenance, and reducing the effort required to upgrade multiple users. On the other hand, local installation can provide a significant performance advantage, especially with slower network configurations.

During the setup process you will be asked to select a destination directory for the Developer files. To create a local stand-alone workstation installation, select a destination directory on a local disk.

To create a shared network installation, select a destination directory on a shared network disk. After installing the Developer files on the network disk, run Setup from each network workstation and perform a "User Setup," as described in "Setting Up a User Workstation" on page 47.

Supported Network Configurations

Pure Unix Installation

ObjecTime Developer 5.2.1 either running locally on a Unix workstation, or using a Unix file server under the following configuration conditions:

- File system must be NFS.
- Network must use TCP/IP.

Mixed Unix and WindowsNT Installation

ObjecTime Developer 5.2.1 can run on a Windows NT workstation connected to a Unix file server under the following configuration conditions:

- Network File system must be NFS.
- Local file system must be NTFS. FAT is not supported.
- Supported NFS clients are **Chameleon** and **Hummingbird**. Make sure to install the clients properly:
    - Support for mixed case file names must be enabled.
    - Consult the NFS client documentation regarding soft links. Some implementations can't handle these very well.
- The path to the Setup program must conform to the 8.3 DOS file naming convention, and the path can not be longer than 63 characters including drive letter and the name of the Setup program.

Installation Requirements

- **CD-ROM drive** – A CD-ROM drive is required to install Developer 5.2.1 from CD. If a CD-ROM drive is unavailable, copy the contents of the disk to a network file server, map the network disk to a Unix file system, and perform the installation from the network location.
- **Administrators Group Membership** – system administrator (root or super-user) privileges are required.
- **50 MB to 170 MB free disk space –** A minimum ObjecTime Install requires 50 MB of free disk space, and a full installation requires 170 MB (which would include support for the complete set of ObjecTime toolset platforms).
- **Printer –** The default printer requirement is, at minimum, a Unix compatible printer. The recommended printer is a PostScript™ printer.

Local Workstation Requirements

- **64 MB main memory minimum –** For large models or build operations, 128 MB or greater is recommended for improved performance.
- **Toolset Memory Requirements** – "Toolset Memory Requirements" on page 115 describes the memory requirements for models.
- **12 MB disk space per working directory minimum –** Each user may have one or more Developer 5.2.1 working directories which contain the user's session file. Developer session files are initially 12 MB and will increase in size with use.
- **256 color graphics adapter** – A high resolution graphics adapter with support for at least 256 colors is recommended.

Installing Netscape Navigator

Online help and documentation for Developer 5.2.1 is provided in HTML format. In order for the help system to function correctly, Netscape Navigator 4.04 must be installed on the user's system. The required software is included on the Developer 5.2.1 CD in the Netscape subdirectory.

> **Note:** Netscape Navigator is provided with the release as part of the online help system. You are licensed to install one copy of Navigator 4.04 per licensed copy of ObjecTime Developer 5.2.1. Please refer to the Netscape license agreement for the terms and conditions.

Unix versions of Netscape Navigator 4.04 are located in the `netscape/unix` directory of the Developer 5.2.1 CD. Separate subdirectories contain versions specific to each toolset platform.

- `hpux_10` contains binaries for HPUX 10.20
- `irix_62` contains binaries for IRIX 6.2
- `sunos413` contains binaries for SunOS 4.1.3
- `sunos_551` contains binaries for Solaris 2.5 and higher
- `aix_4` contains binaries for AIX 4.2.1

Consult the file `netscape/unix/_readme.txt` included on the Developer 5.2.1 CD prior to installing Netscape Navigator.

1 Create a directory where Netscape Navigator will be installed.

```
mkdir /appl/netscape
```

2 Change directory to the created directory.

```
chdir /appl/netscape
```

3 Unpack the Netscape tar file for your toolset platform from the Developer 5.2.1 CD. **Note:** The filesystem location where CD-ROM devices are mounted and the case of filenames on the CD-ROM are dependent upon the version of Unix being used. The following commands may require modification to work on your system.

Solaris: In this example, the Developer 5.2.1 CD is mounted at `/cdrom/objectime`. Solaris usually makes filenames lowercase on CD-ROM devices.

```
tar -xvf /cdrom/objectime/netscape/unix/sunos_551/sparc/netscape.tar
```

HP-UX: In this example, the Developer 5.2.1 CD is mounted at `/cdrom`. HP-UX usually makes filenames uppercase, and appends each filename with '`;1`'. Note that the quotes around the filename are required.

```
tar xvf '/cdrom/NETSCAPE/UNIX/HPUX_10/NETSCAPE.TAR;1'
```

4 Review the Netscape README.install file for any platform specific installation instructions.

5 Add the installed Netscape executable to your path. This should be added to your shell initialization file so that Netscape is available every time you log on.

```
C shell:    setenv PATH /appl/netscape:$PATH
```

Installing ObjecTime Developer 5.2.1

1 Place the Developer 5.2.1 CD in the CD-ROM drive.

2 Mount the CD-ROM device.

You are usually required to be a system administrator (root or super-user) to be able to do this. See the instructions for your particular CD-ROM drive and operating system for details.

AIX: `mount /CDROM`

(or put entry for `/CDROM` in `/etc/filesystems`)

HP-UX: `mount -rt cdfs /dev/dsk/c201d5l1 /cdrom`

IRIX: `mount /CDROM`

(or put entry for `/CDROM` in `/etc/fstab`)

Solaris: `mount -rF hsfs /dev/sr1 /cdrom`

SunOS: `mount -rt hsfs /dev/sr1 /cdrom`

where `/dev/sr1` is the CD-ROM device.

3 From a shell window, change directory to the mounted CD-ROM device.

For example:

```
cd /cdrom
```

4 Run the setup script.

```
./setup.sh
```

On HP-UX, it may be necessary to use the following command (including the quotes):

```
sh './SETUP.SH;1'
```

5 Enter an Installation Key.

Locate the Installation Key letter which you had received with your ObjecTime software media shipment. You will be required to enter these keys to install the appropriate ObjecTime software packages.

```
"Enter an installation key:" enter-urkey-frmth-shpng-envlp<ENTER>
```

6 Review and accept the term of the license agreement.

The license agreement will be displayed and you will be prompted to accept or reject the license agreement. You must accept it to continue:

```
"Enter Y<ENTER> to Accept, R<ENTER> to Read again, or Q<ENTER> to Quit:" Y<ENTER>
```

7 Specify the installation type.

The default set-up type of Typical should always be used unless you are working with ObjecTime Support on installation or packaging issues.

`"Press T<ENTER> for Typicall Installation,`

or `C<ENTER> for Custom Installation:" T<ENTER>`

8 Specify the platforms to be supported by the ObjecTime installation.

Select all platforms to be supported by this installation. The default is no and in the example, only SUN5 was selected by typing “y<ENTER>” at the SUN5 prompt.

```
“Which platforms would you like to be supported?
HP10              Y/N [n]?
IRIX6             Y/N [n]?
SUN4              Y/N [n]?
SUN5              Y/N [n]?   y<ENTER>
NT4               Y/N [n]?
AIX4              Y/N [n]?
Platforms to be supported:
SUN5”
```

9 Specify the installation directory.

The script will prompt you for a directory into which it will copy the Developer 5.2.1 files. The directory name must be specified as an absolute path name. A `Developer5.2.1` sub-directory will be created in the directory that you specify. You must have write permissions for the installation directory. If the directory does not exist, you will be asked if you would like to create it.

```
“Enter absolute installation directory path:”
/testing<ENTER>
```

10 Confirm the ObjecTime Developer 5.2.1 Packages to Install.

You will be asked to confirm the packages and installation directory.

```
The following 6 packages are selected for installation in the
directory ‘/testing’:
(I: Package already installed if ‘Y’)
Package description                  Size in KB    I
==================================================
ObjecTime Platform Independent Code     14804        N
SimulationRTS Common Code                1179        N
Solaris SimulationRTS libraries          2927        N
Generic On-line Documentation and HELP  18968        N
C++ On-line Documentation and HELP       2514        N
Solaris Toolset Libraries                5612        N
==================================================
Selected size:                          46004 kB
Free disk space:                      1218000 kB

Type M<ENTER> to Modify installation directory path, or
Y<ENTER> to Begin installing the selected packages:”  Y<ENTER>
```

11 Obtain License Keys.

When **upgrading from 5.2 to 5.2.1:**

- after installing the software, activate your **5.2** License Keys.

For a **new 5.2.1** license or when **upgrading from pre-5.2:**

- after installing the software, obtain License Keys from ObjecTime Support to run the software. See “License Manager Operations” on page 55 for complete instructions.

Uninstalling ObjecTime Developer 5.2.1

To uninstall ObjecTime Developer use the following procedure:

1. Remove the installation directory and all of its contents.
2. Save any user data files in another location before removing the installation directory.
3. If you are upgrading to ObjecTime Developer 5.2.1, be sure to follow the procedure described in “Starting ObjecTime Developer 5.2.1” on page 50 **before** removing the previous version of ObjecTime Developer.

Setting Up a User Workstation

Environment Variables

ObjecTime Developer requires a number of environment variables to be set. Set the environment variable `$OBJECTIME_HOME` to the new ObjecTime5.2.1 installation directory. Also, set the `$OBJECTIME_LICENSE_SERVER` variable to the name of the workstation running the ObjecTime license manager. Add `$OBJECTIME_HOME/bin` to your path.

These lines can be added to your shell initialization file, so that they are available every time you log on.

Bourne shell (sh or **ksh):**

```
OBJECTIME_HOME=/disk/apps/ObjecTime/Developer5.2.1
export OBJECTIME_HOME
OBJECTIME_LICENSE_SERVER=machine1
export OBJECTIME_LICENSE_SERVER
PATH=$PATH:$OBJECTIME_HOME/bin
export PATH
```

C shell (csh):

```
setenv OBJECTIME_HOME /disk/apps/ObjecTime/Developer5.2.1
setenv OBJECTIME_LICENSE_SERVER machine1
set path=($path $OBJECTIME_HOME/bin)
```

Either logout and then login again, or perform the rest of the upgrade from a new command shell.

Fonts

Note: You should contact your system administrator to determine how to configure the font set-up on your system.

Set the X-windows Font path to point to the new `$OBJECTIME_HOME`. The following command should be added to the X11 start-up script (usually `.xinitrc`, `.x11start` or `.openwin-init`):

```
xset +fp $OBJECTIME_HOME/fonts/<machine-type>
```

where `<machine-type>` is the type of workstation you are executing on (or if executing ObjecTime remotely through another workstation, the type of that workstation). Examples: sun4, sun5, hp, ncd (for NCD X-terminals), ibm.

> **Note:** The ObjecTime fonts will not be set properly, if the user is on an X-Terminal which obtains its boot files from a file server which does not have access to the `$OBJECTIME_HOME/fonts` directory. In this case the fonts should be copied to the file server from which the X-Terminal obtains its boot files.

You may need to add the OT fonts path with:

```
xset +fp /disk6/Release5.2.1/Developer5.2.1/fonts/sun

xset fp rehash
```

Fonts are universal resources and these commands can be typed in any shell on your machine.

Additional Settings

ObjecTime can also be run in batch mode. Please consult the chapter on Batch Mode ObjecTime in the ObjecTime User Guide for further details on this.

Optional settings

The following optional configuration settings may also be made:

For users that wish to use an external editor to edit their RPL, C or C++ code segments, the environment variable `OBJECTIME_EDITOR` must be set to an appropriate window system command to start up the editor.

For example under OpenWindows you could set it as follows to start up an emacs editor:

```
setenv OBJECTIME_EDITOR "shelltool emacs"
```

Or to start up *vi* under the X Window System, use the following for SunOS:

```
setenv OBJECTIME_EDITOR "xterm -e /usr/ucb/vi"
```

Or for Solaris, HP or IBM use:

```
setenv OBJECTIME_EDITOR "xterm -e /usr/bin/vi"
```

The user may also wish to select a default printer at this time. The environment variables PRINTER (for a Sun), and LPDEST (for an HP) will be used by ObjecTime when printing. For example, if the desired printer is ps3 then the following line could be added to the `~/.cshrc` file (assuming csh):

`setenv PRINTER ps3` **(for SUN)**

`setenv LPDEST ps3` **(for HP)**

If you are using a color terminal, you may wish to have the FrameMaker documentation output produce color graphics. To enable this, add the following two lines to your `.xrdb` file:

maker.colorDocs: True

maker.colorImages: True

You will then want to reinitialize your X Window System resources as follows:

xrdb -load .xrdb

If you are running ObjecTime from an NCD X-Terminal, then you may wish to add the following key mapping changes in your X Window System start-up file in order to use the alt key.

xmodmap -e “keysym Alt_L = Meta_L Alt_L”

xmodmap -e “keysym Alt_R = Meta_L Alt_R”

When using HPView on X-Terminals, you must change the keyboard focus policy to provide automatic window focus where the cursor is; otherwise you will not be able to type into textpanes within ObjecTime.

> **Note:** You will have to add the two xmodmap functions relating to Alt_L and Alt_R to your XWindow start-up file in addition to those specifically referring to HPView.

For the HP7XX series workstation, in order for the short-cut keys to work, the following xmodmap changes must be made:

xmodmap -e “remove mod1 = Mode_switch”

In order to use the shift-Tab to allow the user to go to a previous node in the RPL editor, execute the following xmodmap command:

xmodmap -e “keycode 63 = Tab”

The user environment variable OBJECTIME_LICENSE_HOLDTIME can be set to the number of seconds for a toolset license, that has just been relinquished by a user, to be reserved by the License Manager for that user. The default hold time is 300 seconds. For more details see “License Manager Operations” on page 55.

Starting ObjecTime Developer 5.2.1

These operations are normally carried out requiring that you use your userid.

Create a new working directory for ObjecTime5.2.1:

```
create_objectime_dir <new-dir-name>
```

where `<new-dir-name>` is the name of the new local ObjecTime directory to be created. The new release level of ObjecTime is now ready to run.

Change the current directory to the new ObjecTime directory. Start up ObjecTime by typing the following:

1 `cd <new_dir_name>`

2 `objectime&`.

Activate the previous designs, if any, by dragging your updates from the appropriate Directory Browser to the Workspace Browser.

After activating all pre-5.2.1 designs, each user should passivate them again, so that they are saved in 5.2.1 format. This should be to another directory, so that the original updates are not lost.

If no problems occur, you may delete the older ObjecTime user directory after a suitable period of time (and, if applicable, the directory containing the old (5.0, 5.1/5.1.1, 5.2) updates).

Delete the main directory for the previous release once all users are up and running successfully with the new release.

Startup Options

The ObjecTime script may take a number of different options.

Either of the following two options can be used to change the display variable

- `DISPLAY=<displayName>`
- `-display <displayName>`

The following three invokations are equivalent:

1) `objectime -display xterm1:0`

2) `objectime DISPLAY=xterm1:0`

3) `setenv DISPLAY xterm1:0 ; objectime`

The following options control how "verbose" the objectime script is:

- `-q`
- `-quiet`
- `-v`
- `-verbose`

The final command line option is:

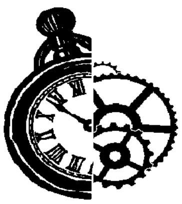
- -control

This option controls whether or not the Target Observability controller is started automatically.

`objectime -control` (start controller automatically)

`objectime` (do not start controller)

Chapter 5



Supported Platforms

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

5.2.1 Host Platforms

Toolset Host	Simulation Services Library Name
AIX 4.2.1 (PowerPC)	AIX4.ppc-CSet-3.1.4
	AIX4.ppc-gnu-2.8.1[a]
HPUX 10.20	HPUX10.hppa-gnu-2.8.1
	HPUX10.hppa-HPC++-10.11
IRIX 6.2	IRIX6.r4400-gnu-2.8.1
	IRIX6.r4400-ProDev-7.2
Solaris 2.5.1 Solaris 2.6	SUN5.sparc-gnu-2.8.1
	SUN5.sparc-SunC++-4.0.1
	SUN5.sparc-SunC++-4.1
	SUN5.sparc-SunC++-4.2
	SUN5.sparc-Green-1.8.8
Sun OS 4.1.3	SUN4.sparc-gnu-2.8.1
	SUN4.sparc-SunC++-4.0.1
	SUN4.sparc-Green-1.8.8
WindowsNT 4.0	NT40.x86-VisualC++-5.0
	NT40.x86-VisualC++-6.0

a. Note: Do not use the 02 (or higher) optimization setting.

Platforms No Longer Supported in Objectime Developer 5.2.1

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

Toolset Host	Simulation Services Library Name
WindowsNT 4.0	NT40.x86-VisualC++-4.2

Chapter 6

License Manager Operations

Licensing Changes

On startup the toolset will acquire licenses for the toolset and for all available code generators, unless they are suppressed by setting the appropriate environment variables. The code generator licenses will be shared with the code generator when it is invoked from the toolset.

The new licenses used by ObjecTime Developer 5.2.1 are:

- 9004 Total number of 5.2.1 toolset sessions. One license is allocated for each active toolset.
- 9030 Total code generation licenses
- 9031 C++ code generation licenses.
- 9032 C code generation licenses
- 9033 Simulation code generation licenses. This enables code generation for the SimRTS for both the C++ and C versions of the product.

License Acquisition Suppression

Whenever started, all variants of ObjecTime Developer (Basic, C, or C++) acquire a license token for each available variant for which the license manager has licenses. It is important to know this in installations where many variants of the tool is installed, using the same license manager.

For example, if you have three licenses for ObjecTime Developer 5.2.1 (OTD Base) and seven for ObjecTime Developer 5.2.1 for C++ (OTD C++), for a total of 10 simultaneous users. The first seven users to log on will get tokens for the C++ code generation, whether they are using OTD C++ or OTD Base. If the three OTD Base users get their tokens first, only four of the OTD C++ users will be allowed to generate code. Also note that even though the OTD Base users have a C++ code generation token, they will also not be able to generate code as the necessary libraries were not installed on their system.

There are two solutions to this situation: Using the license manager functionality to restrict the users able to get tokens, or setting environment variable for the users/workstation to limit the tokens acquired.

Using the License Manager

The first method is to use the capabilities of the License Manager to restrict the users/workstation allowed to acquire certain license tokens. This is done by creating or modifying the License Manager's resource file to include lines such as:

Reserve C++ licenses (9031) for the group

9031:cppusers:user1,user2,user3,user4,user5,user6,user7:7:30

or:

Exclude group from using C (9032) licenses

9032:basicusers:usera,userb,userc:EXCLUDE:0

Refer to the "License Manager Operation" appendix of the *User Guide* for more information regarding this capability.

Environment Variables

Code generation license acquisition can be suppressed by setting user environment variables to indicate that licenses of a certain type not be acquired.

To suppress the license acquisition, set the following environment variables to 0 before starting ObjecTime:

- 9031 C++ code generation: OBJECTIME_CPP_GENERATION
- 9032 C code generation: OBJECTIME_C_GENERATION

ObjecTime Developer Licensing

The license manager is used to control access to ObjecTime Developer. In **ObjecTime Developer 5.2.1**, the ObjecTime License Manager controls access to the toolset for Unix and Windows NT hosts. The License Manager can be run on either Unix or Windows NT machines.

> **Note:** If you install ObjecTime in a stand-alone configuration, you can install the License Manager to execute on the same workstation as ObjecTime.

Licensing is managed by a License Manager program which is generally run on some centrally accessible file server. If you install ObjecTime in a stand-alone configuration, you can install the License Manager to execute on the same workstation as ObjecTime. In addition to having the license manager service clients on a network, the license manager can also be used to provide licenses to clients remotely connected using **dialup networking**. All that is required, in terms of obtaining a license for ObjecTime Developer, is that the machine hosting the license manager is accessible, and identified as the license server, to the machine running the toolset. TCP/IP networking is required to communicate with the license server.

ObjecTime Licenses

The License Manager is responsible for issuing tokens for the various products and their associated features that a customer may have purchased. In order to use a particular product and its feature(s), the executing product must obtain the appropriate feature token (see the table below for a list of valid feature tokens). Hence the License Manager must be running for you to initiate any product feature. The ObjecTime 5.2.1 License Manager can support ObjecTime 4.4, 5.0, 5.1, 5.1.1, 5.2 and 5.2.1 licenses simultaneously. The 4.4, 5.0, 5.1, 5.1.1, 5.2, 5.2.1 licenses are managed as a common pool with the maximum number of tokens available at any one time equal to the total number of licenses purchased.

Table 1 Toolset Feature Requirements

Feature (license)	Toolset version			
	4.4	5.0	5.1 and 5.1.1	5.2 and 5.2.1
9000 (4.4 and total)	X	X	X	X
9001 or 9002 (Unix or NT)			X (one of 9001 or 9002 but not both)	X (one of 9001 or 9002 but not both)
9003 (5.1)			X	
9004 (5.2.1 Toolset)				X
9010 (C/C++ Modeling)		X		
9020 (5.0)		X		

Feature (license)	Toolset version			
9030 (Total Code Generation)				X
9031 (5.2.1 C++ Code Generation)				X
9032 (5.2.1 C Code Generation)				X
9033 (5.2.1 SimRTS Code Generation)				X

Notes

1 Compatibility with earlier versions of ObjecTime will be maintained. It will be possible to start to use the 5.2.1 license manager and supporting scripts with toolset versions back to 4.4.1.

2 Demo key support will carry over to the new license keys.

The above table shows the licensing requirements for various versions of the toolset. Different versions of the toolset require different sets of licenses to run. On startup, the toolset attempts to obtain the licenses necessary. The table indicates which licenses are required for each toolset version, with an X in the cell selected by version and feature.

In the remainder of this section, the term *product* refers to the product/feature combination.

Only Floating Licenses are supported. A Floating License enables the product to be executed on <u>any</u> workstation, within a networked workgroup, up to a maximum of *N* simultaneous usages, where *N* is the number of Floating Licenses purchased. A single License Manager running on a centralized server can manage Floating Licenses for the workgroup. If desired, the set of Floating licenses can be split among multiple License Managers.

License Registration

When **upgrading from 5.2 to 5.2.1:**

- after installing the software, activate your **5.2** License Keys.

For a **new 5.2.1** license or when **upgrading from pre-5.2:**

- after installing the software, obtain License Keys from ObjecTime Support to run the software. See “License Manager Operations” on page 55 for complete instructions.

To produce these keys, we require certain workstation information such as its ***machineid*** and ***IP address*.**

License manager registration

To enable the License Manager program, Objectime Support requires the machineId and the IP address of the file server or workstation upon which the License Manager program will be executing. Please provide ObjecTime Support with this information by filling out the *License Manager Registration Form* and sending it to us.

Obtaining the workstation machineId and IP address

A script utility has been provided which, in most cases, can provide the required machine ID and IP address information necessary to obtain License Keys from ObjecTime Support.

> **Note: 5.2 License Upgrade to 5.2.1 License** - New License Keys are **not** required when upgrading from ObjecTime Developer 5.2 to ObjecTime Developer 5.2.1. Simply follow the installation instructions using your 5.2 License Keys.

To obtain this information, execute the following script in a command or shell-tool window:

```
$OBJECTIME_HOME/bin/ObjecTimeKeyInfo
```

This command returns information about the server machine on which you will run your license server. The command is run from the **server machine**, taking no parameters, and returns the machine's **IP address** and **machineId**.

Command:

Unix: `ObjecTimeKeyInfo`

Windows NT: `ObjecTimeKeyInfo`

> **Note:** On Windows NT, commands must be invoked from the **ObjecTime Developer Command Prompt**. This is a console window started from the ObjecTime command group which has the environment variables set appropriately. Attempting to run the commands on a PC from a normal console window will cause the commands to fail.
>
> Example:

```
ObjecTimeKeyInfo
host: machine
IP addr: 192.139.251.207
MACHINEID: c08bfbcf762a
Server target : NT
ObecTimeKey information written to file otinfo
```

New 5.2.1 License - The `ObjecTimeKeyInfo` command provides a file, `otinfo`, that you should email as an attachment to ObjecTime support (support@ObjecTime.com) when requesting keys.

- Please include the following user information: your company name, project, and ObjecTime prime. This will assist ObjecTime support in identifying and handling your key request quickly.
- On Windows NT, the **MACHINEID** is the **Volume Serial Number** of the first logical hard disk on the PC. This will normally be **C:** and the ObjecTimeKeyInfo command will look for this drive. If for some reason the first drive is not **C:**, then the **Volume Serial Number** for the first logical disk must be obtained by using the **DIR** command and noting the **Volume Serial Number** of this drive.

- When running the license server on Windows NT, the **IP address** that the license manager locks to will be that of the installed network card. If dialup networking is subsequently invoked on the server machine and a dynamic **IP address** generated, then the license manager may use this address for host locking. If this **IP address** is different than the network card address used for key generation, then the license manager will report that the license keys are not valid for the current machine and licensing will fail. Note that the License Manager requires a valid IP address for the host on which it is running. A WindowsNT workstation (or laptop) with its network card removed will not have a valid IP address and the license manager will fail to start on such a configuration.
- Please note that the **Unix** script makes use of the */etc/hosts* file which must be readable by the user. If not, you may have to run the script as *root*. If the utility is unable to determine this information, please consult your Unix administrator.

Invoking License Manager Executables

All the licensing commands are available on both Unix and Windows NT. The commands are identical on both platforms with the exception that on Windows NT the starting and stopping of the license manager is done from the **ElanLM** control panel. Also, on Windows NT the commands are case insensitive. That is, capitalization is not significant when invoking them on Windows NT, but on Unix, incorrect capitalization will result in the command not being found.

Note: Before you can invoke any ObjecTime License Manager executables you must set the following environment variables:

- `OBJECTIME_LICENSE_SERVER` - Set to the host name of the machine where the License Manager is to run.
- `OBJECTIME_HOME` - Set to the main installation directory of the ObjecTime release.

You must then ensure that `$OBJECTIME_HOME/bin` is also set in your PATH.

Note: On Windows NT, commands must be invoked from the **ObjecTime Developer Command Prompt**. This is a console window started from the ObjecTime Developer command group which has the environment variables set appropriately. Attempting to run the commands from a normal console window will cause the commands to fail.

Installation of Encrypted Keys

Unix: `activateKey [-f keyfile] [-k keydir] [-demo]`

Windows NT: `activateKey [-f keyfile] [-k keydir] [-demo]`

Arguments:

- `keyfile` - the filename containing license keys. If the `keyfile` argument is absent then you will be prompted for the key. The `activateKey` command will need to be run for each key being installed, where a key is a digit string separated by line breaks in the provided file containing the keys. If the `keyfile` argument is used, but the `keyfile` is not in the current directory, then the full absolute path to the `keyfile` must be specified.
- `keydir` - the full path to the directory into which to write the license files. If the `keydir` argument is absent then these will be written to `$OBJECTIME_HOME/license`.
- `demo` - this option is used when installing demo keys which are license keys that are not locked to any particular machine. Demo keys are used for evaluation purposes and have an expiry date associated with them. Specifying **`-demo`** tells the key activation routines that these are demo keys and not to prompt for the license server IP address and machineid. If this option is specified with non-demo keys, the keys will install but the license manager will not recognize them as valid keys. Only specify **`-demo`** if you know that you are installing time-limited demo keys.

Example:
```
prompt> activateKey -f keyfile -k /directory/user/test
ObjecTime Key activation program.
Key input file:keyfile
```

```
Enter IP address of the server (default 192.139.251.207):
Enter server target type UNIX or NT (UNIX default)
Licenses installed in /directory/user/test.
```

If the IP address of the workstation where the License Manager is running does not match the IP address that was encoded into the Primary Key for any product, then that key will be discarded and an error report will be output to the License Manager log.

> **Note:** Prior to installing or adding keys, please terminate or stop the License Manager if it is already currently running. Users who already have a token will be unaffected by the shutdown though no new tokens can be issued until the License Manager is back up again.

License Manager

Starting up the License Manager

Note: It is recommended that the same userId be used to both install the license keys and start the license manager. This is because the license manager will periodically re-write the license files, and if the ownership of the files prevents this, the license manager will report an error in the log file and no licenses will be available.

Unix:

```
startLicenseManager [-v msglevel][-l logfile][-k keydir][-r resfile]
```

Arguments:

`msglevel` - A number from 1 to 9 indicating the amount of information to write to the log file. If absent, then msglevel defaults to 3.

1. Error messages only
2. License failures
3. License activity
4. Client connects/disconnects
5. Message per packet received
6. Message per packet sent
7. Further client and Zombie process info
8. Key information
9. All available information

`logfile` - The path and filename for the logfile. If absent then the default is `$OBJECTIME_HOME/license/ObjecTimeLicenseManager.log`

`keydir` - The path where the license key files can be found. If absent it defaults to `$OBJECTIME_HOME/license.`

`resfile` - This is the absolute path and name of a customer maintained resource file used to control the reservation of licenses located in the license key directory.

WindowsNT:

On Windows NT, the license manager is started and stopped from the ElanLM control panel. Prior to starting the license manager, the **TimeZone** variable needs to be set. Before activating the license manager, several settings must first be made:

1 Startup should be set to **automatic**. This will cause the License Manager to be started whenever the server machine is re-booted.

2 Click the settings button and set the license directory and logfile name then close the window. The logfile must specify the full path or no log file will be written.

3 Click the launch button.

Note: On Windows NT, this command requires an administrator class user to execute.

By default the log file `$OBJECTIME_HOME/license/ObjecTimeLicenseManager.log` is created with message output level 3. It is recommended that you always keep a log, since if something goes wrong the error will usually be detailed therein. Things like invalid keys, mismatching key counts, and other interesting information will be recorded, but, depending upon your message output level, so too will token grants and releases. Further, if you want to be able to keep track of these things for reporting purposes, you must have a log file.

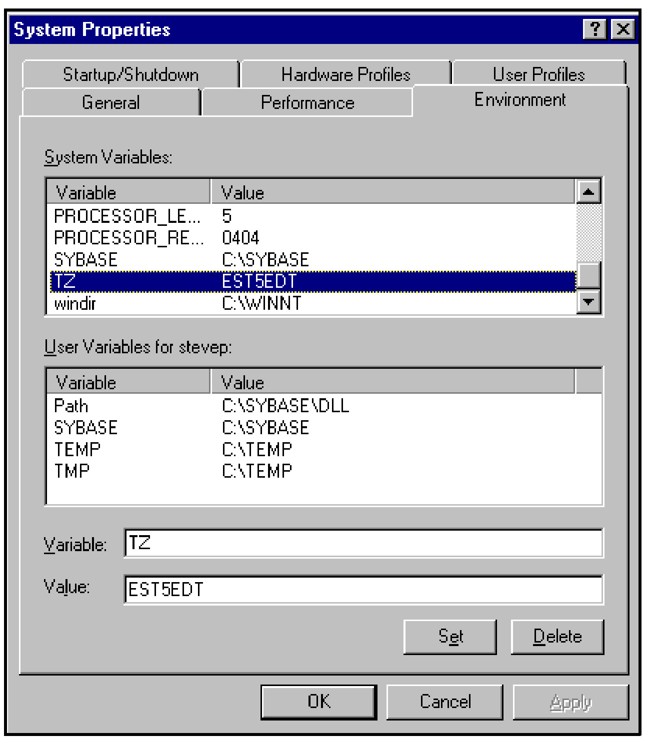
Please note that the process name of the License Manager is `elmd`.

Setting the Time Zone Variable on Windows NT

For proper operation of the license manager over time changes the Time Zone variable must be set. This should be handled by the installation process. The Time Zone variable does not need to be set on Unix systems.

To set the Time Zone variable, proceed as follows:

1. Open the **Control Panel**.
2. Select **System**.
3. Click on the **Environment tab**.
4. Click on any system variable. (not user variable)
5. Replace it with **TZ** /correct value.Eastern Standard Time is used in the figure below.



Use the following syntax to set the Time Zone environment variable:

- set TZ=tzn[+ | -]hh[:mm[:ss]][dzn]
- **tzn** - Three-letter time-zone name, such as PST.
- **hh** - Difference in hours between UTC and local time. Optionally signed. You must specify the correct offset from the Coordinated Universal Time (UTC).
- **mm** - Minutes. Separated from hh by a colon (:).
- **ss** - Seconds. Separated from mm by a colon (:).
- **dzn** - Three-letter daylight-saving-time zone such as PDT. If daylight saving time is never in effect in the locality, set TZ without a value for dzn. ObjecTime Developer assumes the United States's rules for implementing the calculation of Daylight Saving Time (DST).

For example, to set the TZ environment variable to correspond to the current time zone in Germany, you can use one of the following statements:

set TZ=GST1GDT

set TZ=GST+1GDT

These strings use GST to indicate German standard time, assume that Germany is one hour ahead of UTC, and assume that daylight savings time is in effect.

6 Click **Set**.

Make sure that the variable is added to the **System** section and not the user section. You need Administrator privileges to do this.

Automatically starting up the License Manager

It may be convenient to automatically start up the License Manager at file server boot time. This can be done by including the following lines in the appropriate file (for sh) replacing `YourReleaseDirectory` and `server` as appropriate:

```
OBJECTIME_HOME=/YourReleaseDirectory; export OBJECTIME_HOME
PATH="$PATH:$OBJECTIME_HOME/bin"; export PATH
if [ -f $OBJECTIME_HOME/bin/startLicenseManager ]; then
   OBJECTIME_LICENSE_SERVER=server
   export OBJECTIME_LICENSE_SERVER
   $OBJECTIME_HOME/bin/startLicenseManager
fi
```

Note: In the above lines, `OBJECTIME_HOME` must be set to the full path name of the ObjecTime installation directory (that is, `$INSTALL/Developer5.2.1`). As well, `OBJECTIME_LICENSE_SERVER` must be set to the node name of the file server running the License Manager.

The following shows the particular file to insert the above lines into based on the file server platform type:

- SunOS: `/etc/rc.local`
- Solaris: `/etc/rc2.d/S94ObjecTime`
- HP-UX: `/etc/rc` within function `localrc()`

- IBM AIX: `/etc/rc`

  Note: The License Manager in this case is owned by *root*, and hence can only be terminated by *root*.
- **Windows NT:** On WindowsNT, the License Manager is controlled through the control panel and cannot be started from the command line. This command must be run by an administrator-class user.

Before activating the license manager, several settings must be made first:

1. Startup should be set to **automatic**. This will cause the License Manager to be started whenever the server machine is re-booted.
2. Click the settings button and set the license directory and logfile name then close the window. The logfile must specify the full path or no log file will be written. The ObjecTime install will fill in default locations for the logfile and license files directory. These defaults are the same as assumed by the other licensing scripts.
3. Click the launch button.

Bringing Down the License Manager

On **Unix**, the command

```
killLicenseManager
```

will terminate the License Manager specified in the environment variable `OBJECTIME_LICENSE_SERVER`.

On **Windows NT,** the license manager is stopped from the **ElanLM** control panel.

Note: On Windows NT this command requires an administrator class user to execute.

License Manager Operation

When an ObjecTime product is initiated, the user is informed how many tokens were granted and their type. For example, when starting up the ObjecTime Toolset, the following messages may be displayed:

```
objectime: [3] Connected with server "machine1"
objectime: [3] Granted 1 license for "Total"
objectime: [3] Granted 1 license for "TotalUnix"
objectime: [3] Granted 1 license for "Toolset 5.2.1"
```

`[3]` indicates the client ID used by the License Manager. `machine1` is the node name of the file server which is running the License Manager. Upon exiting the ObjecTime Toolset, the following message would then be issued, indicating that the token has been returned to the pool:

```
objectime: [3] disconnecting
```

License Queuing

If a license token is not available upon the start-up of ObjecTime, ObjecTime will be automatically queued and a list of users who currently have active tokens will be output. The queuing can then be cancelled using

^c (that is, control-c). On Windows NT, queuing results in a dialog box being displayed. You may press the Cancel button to stop the queuing operation.

If communication with the License Manager had been lost, and then re-established, a new license token will have to be allocated. If none are available, the user will be given the option of queuing for a license or invoking the *emergency* passivation feature (that is invoked for example, when the ObjecTime application is signalled via `kill -USR1`).

License hold time

When a user relinquishes a toolset license, the License Manager will reserve that license for 300 seconds for re-accessing by that user. To override this default time, the user environment variable `OBJECTIME_LICENSE_HOLDTIME` can be set to the number of seconds for a toolset license to be reserved for re-access. Setting this environment variable to zero will result in the license being immediately returned to the pool upon exiting the ObjecTime session. On Windows NT, this setting is accessible via the ElanLM control panel.

License auditing

During normal execution of a product, each corresponding Unix process with a token periodically notifies the License Manager (every 200 seconds) that it is alive and is still using its token. If a process fails to report (for example, dies), without releasing the token, the token will be returned to the pool after ten minutes.

The utility `killUserLicense <userid>` also allows the administrator to force the de-allocation of an ObjecTime session's license token. This can be used in those situations where a session still has a token (but the user is unable, for whatever reason, to terminate the session properly and thereby release the token), or when a token is still being held by the License Manager (see License hold time above). If more than one license token has been issued to a particular user (distinguished by their userId), then a list of these will be given from which the one to de-allocate can be chosen. Note that only the owner of the License Manager process can use this utility.

If the License Manager goes down or is otherwise unable to communicate to the process, those who already have a token will be able to continue work and will be reissued tokens when the License Manager resumes operation.

Querying the License Manager

Currently allocated licenses

The command

```
licenseInfo
```

will give you information regarding who is currently using the system, and the number of used and available licenses. This information will be given for all active license managers. For example:

```
ObjecTime License Manager Information System
Please wait...
```

```
Server user1:
  CID LID User                            Feature    Group    Started
   --- --- ------------------------------ ---------- -------- -------
    1   1 user1@user1                    Total      -       Aug 03 11:36
    2   2 user1@user1                    Total      -       Aug 03 11:40
Total [9000]: 25 licenses, 2 in use; installed Aug-03-98
              Expires Oct-30-98.

TotalUnix [9001]: 25 licenses, 0 in use; installed Aug-03-98
                  Expires Oct-30-98.

  CID LID User                       Feature    Group    Started
  --- --- ------------------------- ---------- --------  -------
    1   1 user1@user1              TotalNT      -       Aug 03 11:36
    2   2 user1@user1             TotalNT      -       Aug 03 11:40
TotalNT [9002]: 25 licenses, 2 in use; installed Aug-03-98
                Expires Oct-30-98.

Total5.1 [9003]: 25 licenses, 0 in use; installed Aug-03-98
                 Expires Oct-30-98.

  CID LID User                     Feature    Group    Started
  --- --- ----------------------  ---------- -------- -------
    1   1 user1@user1             Total5.2.1    -     Aug 03 11:36
    2   2 user1@user1             Total5.2.1    -     Aug 03 11:40
Total5.2 [9004]: 25 licenses, 2 in use; installed Aug-03-98
                 Expires Oct-30-98.

C++ [9010]: 25 licenses, 0 in use; installed Aug-03-98
                 Expires Oct-30-98.

Total5.0 [9020]: 25 licenses, 0 in use; installed Aug-03-98
                 Expires Oct-30-98.



  CID LID User                     Feature    Group    Started
  --- --- ---------------------- ---------- -------- -----------
    1   1 user1@user1           TotalCodeG -       Aug 03 11:36
    2   2 user1@user1           TotalCodeG -       Aug 03 11:40
TotalCodeGen [9030]: 25 licenses, 2 in use; installed Aug-03-98
                     Expires Oct-30-98.

  CID LID User                          Feature    Group    Started
  --- --- ---------------------------- ---------- -------- -------
 S  1   1 user1@user1                   CodegenCPP -       Aug 03 11:36
```

```
    2   2 user1@user1                   CodegenCPP -       Aug 03 11:40
CodegenCPP5.2 [9031]: 25 licenses, 2 in use; installed Aug-03-98
                      Expires Oct-30-98.

  CID LID User                          Feature    Group    Started
  --- --- -------------------------- ---------- -------- ------------
 S  1   1 user1@user1                   CodegenC5. -       Aug 03 11:36
    2   2 user1@user1                   CodegenC5. -       Aug 03 11:40
CodegenC5.2 [9032]: 25 licenses, 2 in use; installed Aug-03-98
                    Expires Oct-30-98.

  CID LID User                          Feature    Group    Started

  --- --- -------------------------- ---------- -------- -----------
 S  1   1 user1@user1                   CodegenSim -       Aug 03 11:36
    2   2 user1@user1                   CodegenSim -       Aug 03 11:40
CodegenSimRTS5.2 [9033]: 25 licenses, 2 in use; installed Aug-03-98
                         Expires Oct-30-98.


ObjecTime license server information.

C:\>pause
Press any key to continue . . .
```

Usage statistics

To receive a breakdown of license manager usage use the command

```
serverUsageReport [log_path] [daily]
```

where

- `log_path` - is the absolute path-name for the file where status information has been logged. Default: `/tmp/ObjecTimeLicenseManager.log`
- `daily` - allows you to optionally have the activity report broken down by daily usage, rather than amalgamated. This parameter is case insensitive.

The following shows an example using the reporting facility:

```
serverUsageReport

ObjecTime License Manager Information System

                   Total    Total   Over   Number Number Percent  Total
Feature          Requests  InUse  SoftLim Issued Denied Denied Time Used
------------------------------------------------------------------------
CodegenC5.2           54     4             54      0     0%     6:45:48
CodegenCPP5.2         54     4             54      0     0%     6:45:52
CodegenSimRTS5.2      54     4             54      0     0%     6:45:46
Total                   58     3             55      3     5%     12:10:53
Total5.1                 1     1              1      0     0%      0:07:37
Total5.2              54     3             54      0     0%    12:03:06
TotalCodeGen            54     3             54      0     0%     12:02:44
TotalNT                 51     2             51      0     0%      9:04:40
TotalUnix                4     3              4      0     0%      3:06:09
```

Chapter 7

Documentation Roadmap

ObjecTime Developer 5.2.1 Documentation Set

The ObjecTime Developer 5.2.1 Documentation Suite for Release 5.2.1 contains the following documents: 5.2 *User Guide, 5.2 C++Language Guide, 5.2 C Language Guide, 5.2 RPL Language Guide, 5.2 Tutorial Guide, 5.2 C++ Target Guide, ObjecTime Developer 5.2.1 Getting Started Guide & Release Notice, C++ Target Module 5.2.1 Getting Started Guide & Release Notice* and *C Target Module 5.2.1 Getting Started Guide & Release Notice*. The Documentation Sets are structured as follows:

Table 2 ObjecTime Developer (OTD) 5.2.1 Documentation Sets

ObjecTime Developer	ObjecTime Developer for C++	ObjecTime Developer for C
User Guide 5.2	User Guide 5.2	User Guide 5.2
Tutorial Guide 5.2	Tutorial Guide 5.2	Tutorial Guide 5.2
C++ Language Guide 5.2	C++ Language Guide 5.2	C Language Guide 5.2
RPL Language Guide 5.2	RPL Language Guide 5.2	
OTD 5.2.1 Getting Started & Release Notice	OTD 5.2.1 Getting Started & Release Notice	OTD 5.2.1 Getting Started & Release Notice
	C++ Target Module 5.2.1 Getting Started & Release Notice	C Target Module 5.2.1 Getting Started & Release Notice
	C++ Target Guide 5.2	

Note: Where differences exist, the documentation highlights **Windows NT** and **Unix** specific information.

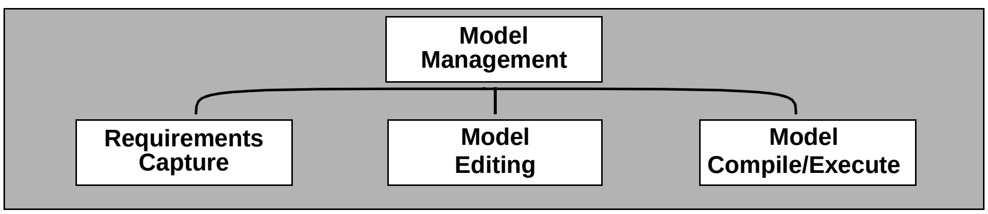
User Guide

This document contains detailed reference material for all the tools and windows in the ObjecTime Developer 5.2.1 Toolset. It also contains instructional task-related information for users who are less familiar with the Toolset interface and need instruction on how to perform certain tasks. The *User Guide* contains descriptions of all the basic concepts underlying the ObjecTime toolset including topics common to the C++ and C Language usage.

Note: It is strongly recommended that you review the *User Guide* in detail.

The ObjecTime Developer 5.2.1 Toolset contains tools for all aspects of the real-time development lifecycle. The document is organized into five major parts describing these tools: Model Management, Model Editing, Requirements Capture, Model Compilation & Execution, and an Introduction which provides a general overview of ObjecTime for new users.

Figure 5 ObjecTime User Guide Organization



C++ Language Guide

The *C++ Language Guide* contains information on how to use the Toolset to develop and compile models in a C++ environment for a variety of targets. It also contains information on the generated code structure of a C++ model, and information on customizing certain aspects of the generated code. The C++ programming interface to the Run-Time System Services and all built-in data types are also provided. This is required reading and reference material for building any C++ model in ObjecTime. In addition, the document describes how to interface ObjecTime with other applications through the External Layer interface. Essentially, any information about the use of C++ in an ObjecTime application can be found in this document.

C++ Target Guide

The *C++ Target Guide* describes the architecture of the Target Run-Time System (Target Services Library) for ObjecTime Developer 5.2.1. This document describes the structure of each Target Services Library part and its collaborations in enough detail to allow users to understand and debug their Target-based models. It will also allow users to understand how and where they can customize Target-based programs.

C Language Guide

The *C Language Guide* describes all aspects of C usage in ObjecTime, including the toolset interface for using C within ObjecTime (and the use of C and C++ actors in one design), the semantics of the ROOM run-time system service calls and rules for integrating external libraries, data structures, and applications with your ObjecTime C models.

RPL Language Guide

The *RPL Language Guide* contains information on building RPL-based models in ObjecTime. This document describes the RPL language, and all built-in data types, along with the RPL programming interface to the Run-Time System Services. Information about the RPL syntax-directed editor is also contained in this document.

Tutorial Guide

There is one *Tutorial Guide* with three different sections : (1) the RPL version of the Tutorial, (2) the C++ version of the Tutorial and (3) the C version of the Tutorial.

Getting Started Guide & Release Notice

It is recommended that the user reads this guide to get the most up-to-date information on this release. For ObjecTime Developer 5.2.1, the *ObjecTime Developer Getting Started and Release Notice* describes the base product. The *ObjecTime Developer for C++ Getting Started and Release Notice* and the *ObjecTime Developer for C Getting Started and Release Notice* describe the areas specific to the C++ and C Language Modules.

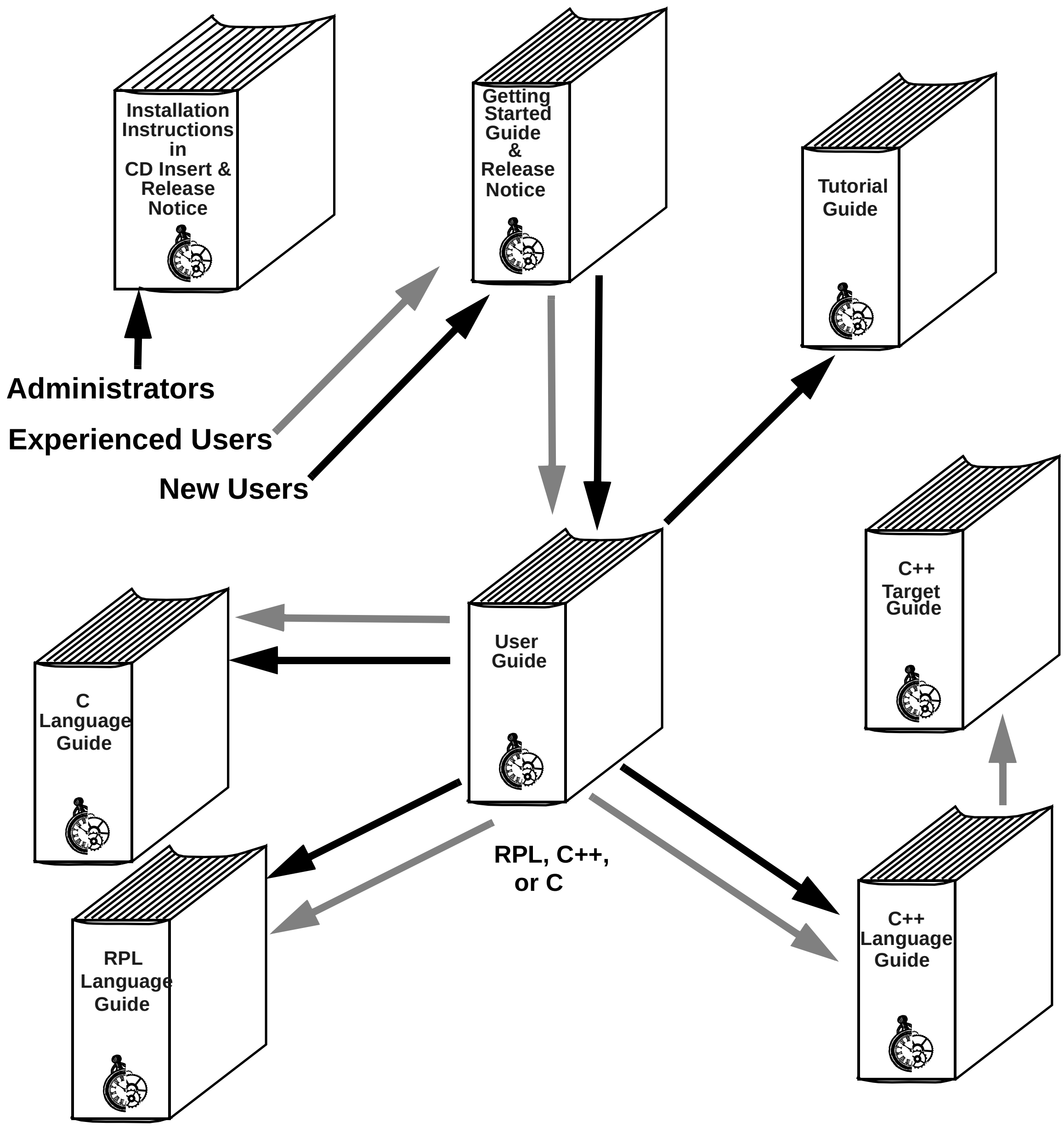
Suggested Reading Path

We strongly recommend that users who are new to the ObjecTime Developer concepts and toolset read the *User Guide* and follow the examples in the *Tutorial Guide* (RPL, C or C++ sections as appropriate). All users will need to reference the *User Guide*, the *C++ Language Guide*, the *C Language Guide* or the *RPL Language Guide* as they build models in ObjecTime Developer. Figure 6 below shows the recommended reading paths for different user needs.

Note: For more information on Real-Time Object-Oriented Modeling (ROOM), please refer to the following sources:

- the ObjecTime Developer ***User Guide***,
- the ***Real-Time Object-Oriented Modeling***. Selic, Gullekson, and Ward. John Wiley & Sons, Inc., 1994.

Figure 6 Recommended Reading Paths in the ObjecTime Developer Document Set



Note: The ***C Language Guide*** is only included in the ObjecTime Developer C Target Module documentation set. The ***C++ Target Guide*** is only included in the ObjecTime Developer C++ Target Module documentation set. The information in the ***C++ Language Guide*** is applicable to both the C++ Simulation Services Library and the C++ Target Services Library.

Online Reading

The complete set of HTML-based ObjecTime Developer 5.2.1 documentation is linked to the ObjecTime Developer 5.2.1 Help. When a user clicks on the menu item **Help Contents** of the toolset menu, the configured browser comes up with the top level documentation page. Also, the user can obtain online context sensitive help by clicking on the menu item **Help** of any ObjecTime Developer menu. All online documentation may be accessed through the online Help System with every cross-reference in the documentation set as a hypertext link to the referenced material. Clicking on any cross-reference will take you to the referenced location.

Online Search Engine

The ability to search the online documentation has been added into the 5.2 product release. From the ObjecTime Online documentation, the search engine can be brought up from any one of a number of places.

Figure 7 Help Contents

Figure 8 A Table of Contents

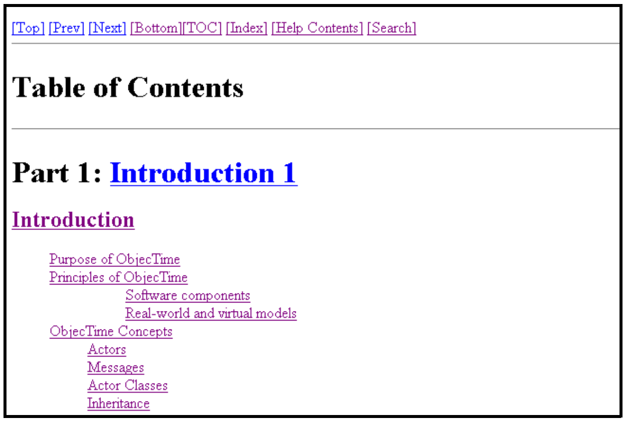
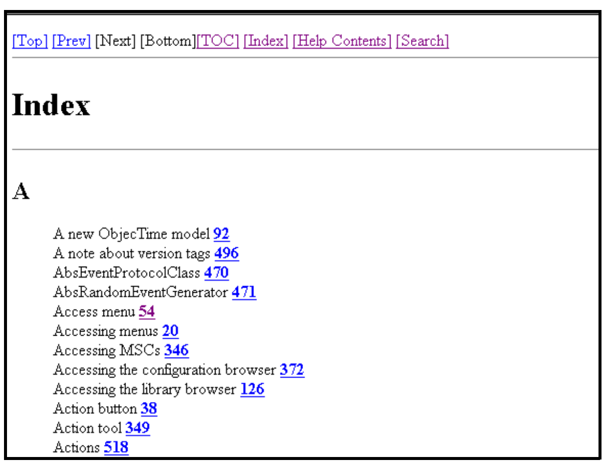

[Top] [Prev] [Next] [Bottom][TOC] [Index] [Help Contents] [Search]

Table of Contents

Part 1: Introduction 1

Introduction

Figure 9 An Index

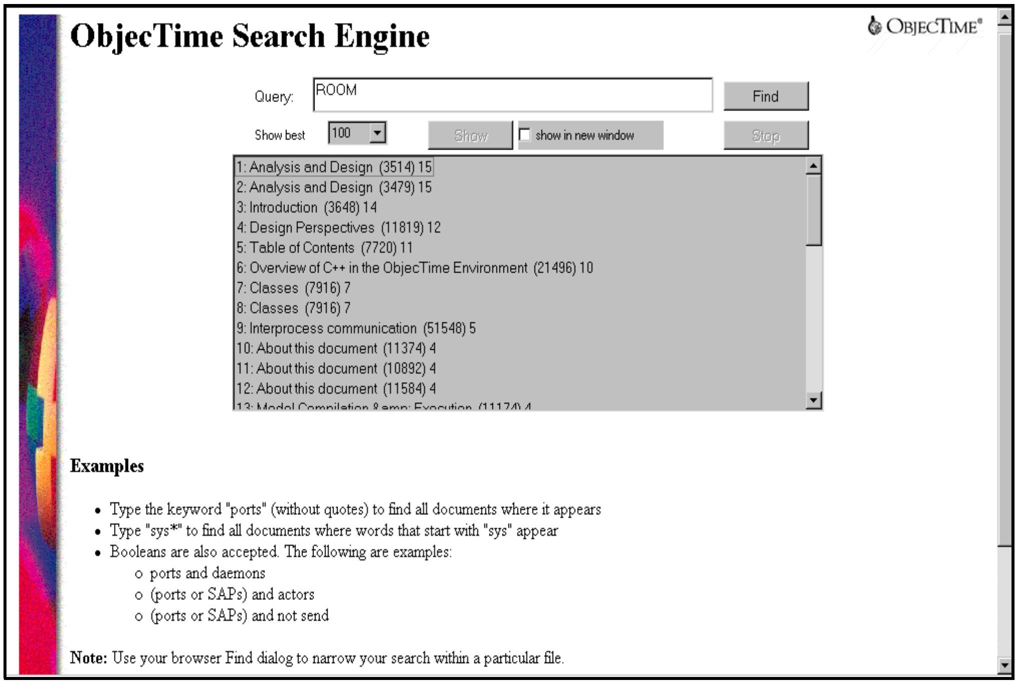

[Top] [Prev] [Next] [Bottom][TOC] [Index] [Help Contents] [Search]

Index

A

The online search engine can be used to identify documents within the online documentation which contains the identified keywords.

Figure 10 Search Engine



Within the identified documents, the find function of the browser should be used to jump to the specific references.

Chapter 8

ClearCase Support Enhancements

Introduction

This chapter discusses ObjecTime Developer's ClearCase Support Enhancement introduced in the **5.2.1** release. **ObjecTime Developer 5.2.1** is designed to better support the ClearCase environment, and in particular Clearmake. It extends compilation capabilities to allow compilation outside of the Toolset, and use of Clearmake's facilities to manage dependencies and store loadbuild artifacts.

Definitions

This section contains brief definitions of some of the terms used in this chapter.

Toolset - this refers to the main executable program of ObjecTime Developer. Model editing is done within the Toolset. All scripts and the external code generation subsystem are not part of the Toolset for the purposes of this document.

Code Generation Subsystem - this refers to the ObjecTime subsystem that generates C++ source code from an ObjecTime model.

Root Package - this refers to the logical package that represents an update/context. All classes in the update/context appear in this package. This term is used only for the purpose of discussion within this chapter.

Summary

The ClearCase Support Enhancements:

- allow Toolset synchronization with ClearCase views through commands that synchronize the Toolset's view with the ClearCase view of classes.
- allow designers to access parallel work from others via the synchronization noted above.
- allow external builds through a modification to the code generation subsystem to use the project files stored in ClearCase.
- allow the re-use of loadbuild artifacts through the use of ClearCase wink-in. This is implemented through modifications to the external code/makefile generation process to properly support Clearmake rules within this process.

- redesign project files such that they do not contain all classes from an update. They contain only packages and the classes from the root package.
- allow traceability of loadbuild artifacts back to the originating class files in the project, using the ClearCase config records of derived objects.
- enable Clearmake's automatic dependency tracking, using the ClearCase config records of derived objects.

ClearCase Support Enhancements support ClearCase version 3.2 running under Solaris 2.5.1/2.6 or Windows NT 4.0.

The following are not supported by the ClearCase Support Enhancements:

- external builds for the ObjecTime simulation environment are not supported.
- only the C++ language is supported.

Project Files

This section is a brief description of an ObjecTime project file and how it is used with the ClearCase Support Enhancements.

Project files in **ObjecTime Developer 5.2** contained a list of classes, packages, thread mappings and configurations, along with their versions, which were activated to form an update. Activating a project file merged the referenced objects into the Toolset and created an update. When a project file was submitted, you were required to submit all checked out versioned objects referenced by the project file.

ObjecTime Developer 5.2.1 changes the definition of project files to avoid storage of all classes within the class list. Instead, the class list only references the classes which appear in the root package but do not appear in any of the packages referenced by the project file. The entire list of classes is computed by using the package list and traversing the contents of the packages. Using this scheme, classes may be added to or removed from packages without requiring a change in the project file. This scheme removes the contention for the project file between developers when working in a large team environment.

From a ClearCase view, it is now possible to initiate a Clearmake without involving the Toolset. **ObjecTime Developer 5.2.1** extends the use of project files to allow them to be used for an external build process.The following arguments may be supplied at the command line:

- select a specific project file to use for the build
- select an output path for the build
- select the top actor to use for the build
- select the active configuration from those listed within that project file to use for the build.

If no configuration is selected, the default configuration stored within the project file is used. If no top actor is selected, the default top actor specified within the project file is used.

Note: When a project file is used for a Clearmake build process, the versioning information is ignored. Instead, the version that is in the current view is used. This applies to both internal builds, that is, builds initiated from within the Toolset, and external builds.

ObjecTime Developer 5.2.1 extends the activation of project files to include an option which allows you to merge either the specifically named versions of entities (classes, packages, configurations) or the version of these entities which are in the current view.

The development process

This section discusses one possible set of steps to employ when using the Toolset in conjunction with the various roles involved. There are potentially three main roles and are described as follows:

- **Developer:** The developer is responsible for adding/changing/deleting new objects to/from the update, such as classes/configurations/threads setups
  - The developer works in read-only mode, that is, 'Allow edits on non-checked out objects' user preference is turned off.
  - Before editing a class, the developer will check it out
  - The developer makes changes and then saves them to the library
  - If the developer adds or removes classes within packages, then the affected packages must be checked out and resubmitted to the library
- **Integrator:** The integrator is responsible for adding/maintaining/deleting packages, ensuring that packages/project files are updated as needed and reflect the most recent developer-released lineup of classes.
  - At the integration cutoff, the integrator will merge in the project file for the previous loadbuild. This brings in the current version (version in view) of the packages (and other objects) referenced by the project file and the objects referenced by these packages. Then checking out and submitting the new project file creates a version that reflects the most recent lineup of root level objects.
  - This job may be performed either interactively or in batch mode.
- **Load builder:** The load builder is responsible for building the load from the appropriate project files and is able to work outside the toolset, if the most current versions of the project file to build from are stored in the library. Note that if the loadbuilder builds loads outside of the toolset, it is suggested that a verification step be done within the toolset as part of the loadbuild process. This verification step, performed by selecting the 'verify' mode on the compile dialog, would ensure that the ROOM model validation is performed. See "No ROOM compile time checking" on page 93 for more detail.

  Note: The role of the integrator may be played by a developer.

  Checking in a file does not automatically imply submission to the build. Submission to the build is an explicit process whereby the top level integrator (builder of the update) is explicitly notified of deletions/additions to the project file. This could be automated/enforced as an integral part of the build.
- The Configuration Management topic is covered in-depth in ObjecTime's Advanced Development Workshop.

Toolset Enhancements

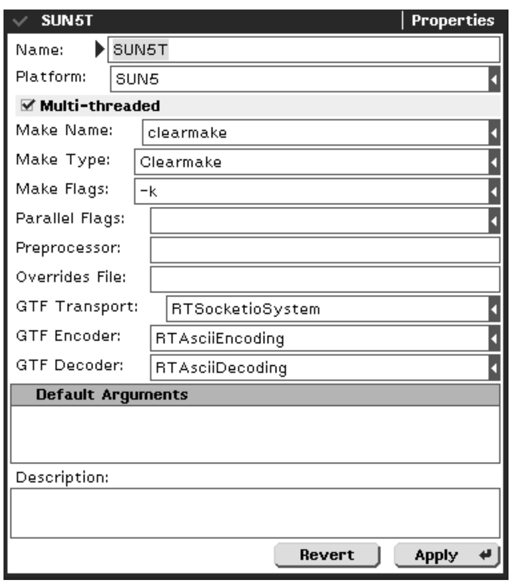
This section discusses the changes to the Toolset to implement the ClearCase Support Enhancements. The changes are described as changes to the ObjecTime Developer version 5.2 release.

Enabling Clearmake mode

Make Types are specified in the Targets properties editor as seen in Figure 11, “ClearCase/Clearmake mode,” on page 83. ObjecTime Developer 5.2.1 changes the behavior of the ‘Clearmake’ make type. This make type will now be used to indicate that the ClearCase/Clearmake mode is in effect. When this mode is turned off, by selecting another make type, the Toolset behaves as it did in 5.2. Using ClearCase as a library system does not imply or require using Clearmake as the Make Type.When this mode is turned on, the following changes occur to the compile process:

- The compile dialog button Generate Changes Only is disabled and turned off.
- If the Recompile button on the compile dialog is selected, a `clearmake -u` is performed.
- No IF (Intermediate Form) files are generated during the compile. The compilation works directly off the files stored in the ClearCase library.
- The Save to Library command (see below) is automatically run when the Compile menu item is executed. You are informed of any issues encountered with saving.

FIGURE 11. ClearCase/Clearmake mode

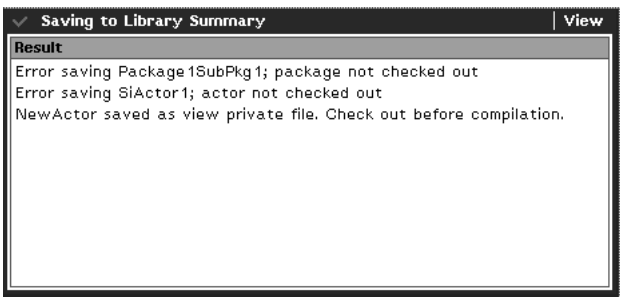


Save to Library

The Save to Library command is added to the Update application menu of the model and update browsers. This menu item performs the following:

- saves the current changes to each checked-out class in the library.
- for added classes, creates a view private file if the class has not been checked out. If the class has been checked out, then it is treated like the changed classes above.
- references to deleted classes are removed from the project file. Deleted classes continue to exist in the library and in older versions of the project files.
- does not cause a new version of the project file to be written to the library (unless the project file is checked out).

FIGURE 12. Saving to Library Summary

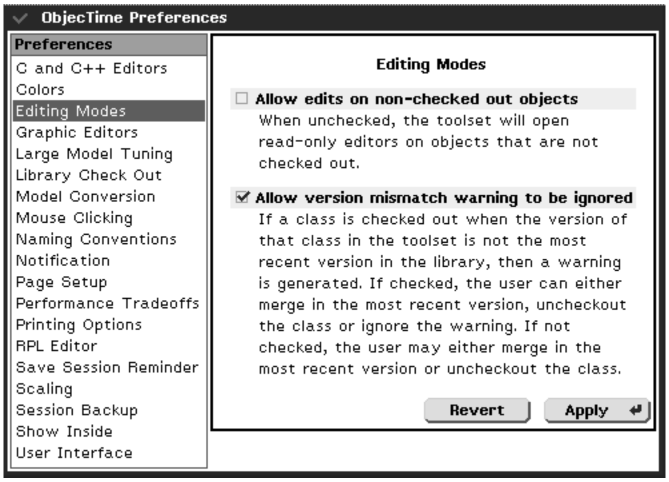


Enhanced editing modes

The user preference ‘Allow edits on non-checked out objects’ allows you to specify the use of read-only editors on objects that are not checked out. When this option is not selected, a class must be checked out before it can be edited. One deficiency in this scheme is that edits on classes can cause changes to other classes that are not checked out. For example, removing signals from a protocol class can cause transition events to change thereby changing the class containing the transition. ObjecTime Developer 5.2.1 addresses this deficiency by forcing a class to be checked out when it is marked as changed (with a solid delta).

> **Note:** When using ClearCase, the ‘Allow Edits on non-checked out objects' checkbox should be OFF.

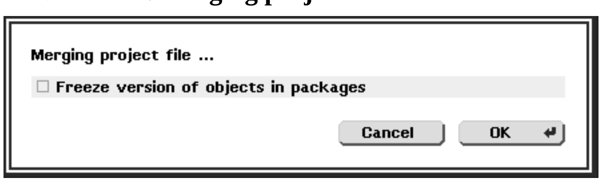
FIGURE 13. ObjecTime Preferences



Project file activation

As discussed earlier, ObjecTime Developer 5.2.1 extends the activation of project files to include an option which allows you to merge either the specifically named versions of entities (classes, packages, configurations) or the version of these entities which are in the current view. This is done through the 'Freeze version of objects' check button on the dialog which appears when a project file is brought into the Toolset. This check button, when set, causes the specifically referenced package, class and configuration versions to be brought into the Toolset. When this button is unchecked, the versions of the packages, classes and configurations that are in the current view are brought into the Toolset. It is recommended to leave the button unchecked and to modify your ClearCase config spec to pull in the appropriate versions of all objects.

FIGURE 14. Merging project file



External diff before marking solid delta

When edits are performed in the Toolset, they can result in changes to other objects (classes/packages/configurations). These subsequent changes result in these other objects being marked with solid deltas. Because the algorithm that marks these objects with solid deltas is overly pessimistic, there are cases where library objects are marked with a solid delta when it is not required. This is automatically enabled when Clearmake mode is ON. If "Allow edits on non-checked out objects' is OFF, the objectime_diff script is run to determine precisely if objects need to be marked with a solid delta and checked out. If they are not different, the objects are not checked out.

Configuring your project to use Clearmake

There are four steps to configuring your project to use Clearmake, and they are given below.

Configuring your view

There are very few restrictions imposed on the config spec that defines your view. ObjecTime Developer does not internally use any branches, labels or attributes on file elements, so it should adapt readily to any pre-existing configuration management. The file elements visible within your view are defined by your config specs. For more information on config specs, refer to your ClearCase documentation.

We recommend that the CHECKEDOUT rule should precede any other rules (unless you do not intend to check out any files), so that the toolset can read and write checked-out files.

When different files in the project are selected by different rules in the config spec, the state of the whole project in the view must be kept self-consistent. This is particularly important during Generation, since any single class can read many other LF classes.

On Windows NT, the view is mapped to a drive letter (typically `Z:`). A known limitation currently requires all developers to map their view to the same drive letter. Furthermore, all developers' drive-mappings must be the same case, that is, Z: or z:. This is required because build scripts stored in the config records must match identically for wink-in to work.

On Windows NT, the MVFS (multiversion file system) must preserve case. This is set in the Control Panel by proceeding as follows:

1 open the ClearCase icon

2 select the MVFS tools

3 check the "Case Preserving" checkbox.

Configuring your environment

You are required to tell the code generator where it can put its temporary files so that `clearmake` does not track them. The environment variable OBJECTIME_CODEGEN_TEMP must be set to any pre-existing directory not in a ClearCase view. For example, on Unix:

```
setenv OBJECTIME_CODEGEN_TEMP /tmp
```

On Windows NT, the OBJECTIME_CODEGEN_TEMP environment variable can be created from the Control Panel's System folder with a value such as C:\temp.

If you do not have this environment variable set, wink-in will likely not occur.

Configuring the session image

You must specify the path to the forClearCase scripts in your session's Library Configuration menu. On Windows NT, typically this path is %OBJECTIME_HOME%\bin\LibraryInterface\forClearCase. On UNIX, this path is $OBJECTIME_HOME/bin/LibraryInterface/forClearCase. Unix users can alternatively put a symbolic link (`.objectime_scripts_dir`) in all ClearCase libraries.

From the Library Configuration menu, de-select "use objectime_library_info for sync".

If you are using the user preference "Allow edit on non-checked out objects" (this option is on by default), you must be very careful to check-out all files that get black deltas. It is suggested to de-select this user preference, when using ClearCase.

Configuring the model

All classes must be versioned in ClearCase, as well as the config files, the package files and the project file. Package configurations, however, do not need to be separately versioned since they are versioned as part of their package. Access to project and config files may be restricted only to those user, integrator, or loadbuilder roles. Access to package files and their classes may be restricted to the developers who regularly work on that package.

The Output Path (part of the project properties) must be within the same ClearCase view as the linear form class files. If the session was launched from that view, then PWD (the default Output Path) will work. If the session was not launched from the view, the Output Path must contain a path within the view. On Windows NT, this would typically mean the Output Path must be explicitly the drive letter `Z:`.

To activate Clearmake mode, the active configuration's Make Type must be Clearmake. The Make Name should be "`clearmake`" (whatever your Clearmake executable is named). See Figure 11, “ClearCase/Clearmake mode,” on page 83.

In non-Clearmake mode, dependency analysis is done by a `makedepend` script which examines inclusion paths. In Clearmake mode, dependencies are tracked by Clearmake which watches file-accesses within the view and ignores file-access outside the view. You may wish to version external inclusion files in ClearCase and change the inclusion paths within your models.

Using Clearmake for developers

Creating a new object

All objects (projects, project configurations, packages, actor classes, data classes and protocol classes) must be versioned in ClearCase before a Generate or Compile can be issued. After creating the new object in the ObjecTime session, it should be checked-out from a ClearCase library.

When classes within a package are added, deleted or renamed, the package file must be checked out as well.

When classes not within a package, or packages or project configurations are added, deleted or renamed, the project file must be checked out as well. To reduce project file contention, these activities should be minimized. Most (if not all) classes should appear within a package file.

Making changes to project file

You will need to checkout the project file whenever performing the following activities:

- adding, deleting or renaming unpackaged classes (classes not in a compilation package)
- adding, deleting or renaming project configurations
- adding, deleting or renaming packages
- changing the Threads configuration
- changing the default Output Path
- changing the default Top Actor
- changing the default Active Configuration

In order to reduce checkout contention for the project file, typical development activities should not normally affect the project file. Configuration management may choose to restrict write-access to the project file, such that the above activities are only performed by load-builders or integrators.

This does not restrict developers. Developers can ‘temporarily’ change the Output Path, Top Actor and Active Configuration used in their session without writing the project file.

Classes can be added, deleted and renamed from packages by checking out the package file. Consequently, write-access of the package file is required for developers performing this activity.

It is a known limitation that the project file does not get a black delta when one of these changes necessitates a checkout.

Invoking Clearmake from the Toolset

Once the model is properly configured, invoking a Compilation using Clearmake is the same as any other compilation: select the Top Actor and compile it.

While a toolset-initiated Compile is running, the progress dialog will flash messages about what files it is generating, compiling or winking-in. On Unix, you can also run:

```
tail -f compile.output.<ACTIVE_CONFIG>
```

to follow Compilation progress, where `<ACTIVE_CONFIG>` is the name of the Active Configuration.

Invoking Clearmake from the command-line

If invoking Clearmake from command-line, use the following command:

```
perl $OBJECTIME_HOME/codegen/generate.pl \
  -p <OUTPUT_PATH> -j <PROJECT_FILE> \
  -m <MAKE> -t Clearmake \
  [-c <ACTIVE_CONFIG>] [-a <TOP_ACTOR>] [-o <OUTPUT_FILE>] \
    compile
```

For convenient copy-and-pasting, the exact command used by the last Compile (that is, as issued by the toolset's Compile) is written out to the Compilation Results file, "`compile.output.<ACTIVE_CONFIG>`". The options are as follows:

- OUTPUT_PATH is an absolute directory-path that tells where the build should be produced. Different projects can have the same OUTPUT_PATH because the build results actually go in a sub-directory named after the Project. The OUTPUT_PATH used by the toolset is stored in the project file. The default Output Path of a Project file is $PWD, the directory from where the toolset was started. The OUTPUT_PATH must be provided to `generate.pl` (no default is available) because it starts writing files before the Project file is read.
- PROJECT_FILE is an absolute file-path to the project file.
- MAKE is the name of the Make executable. Typically, this is "`clearmake`", which must be on your path.
- The option "-t Clearmake" (when produced by the toolset) specifies the Make Type used in the Active Configuration. This activates the Clearmake mode.
- ACTIVE_CONFIG is the active configuration, which by default is specified in the project file. A developer can override the active configuration without having write access to the project file.
- TOP_ACTOR is the top actor, which by default is specified in the project file. A developer can override the top actor without having write access to the project file.
- OUTPUT_FILE is the Compilation Results file, which by default is "`compile.output`" (or “`compile.output.<ACTIVE_CONFIG>`” if the active configuration is provided on the command line). This can be read into the toolset using "Import Compilation Results..."

Enabling parallel builds with Clearmake

Parallel building is enabled for both code-generation and compilation on Unix platforms.

Note: As of ClearCase 3.2.1, Clearmake does not support parallel building for NT.)

However, use of the Clearmake concurrency environment variables CCASE_CONC and CLEARCASE_BLD_CONC is not recommended. Because of the way ClearCase buffers and merges parallel build stream output, the progress messages of a ClearCase session are typically spooled until the entire build is finished. ClearCase users are encouraged to use a scenario as described below.

As discussed in the section, 'Using an environment variable for Parallel Make Flags' on page 12, it is advisable to put `$(OBJECTIME_PMAKE_FLAGS)` in the Parallel Flags field of your configuration browser. This way you (not the update) decide how much parallelization your compilation host(s) uses.

Your must then assign some environment variables. For example:

```
setenv OBJECTIME_PMAKE_FLAGS "-J 4"
setenv CCASE_HOST_TYPE sun5
```

Then, create a build host file (~/.bldhost.sun5) as described in the ClearCase Reference manual (bldhost). For example:

```
-idle 10%
beef
helium
beef
beef
```

where the machine "beef" has significantly more processing power than the machine "helium".

Clearmake can now be invoked from the command-line, or by the toolset, without arguments, and the parallel flags will be correctly invoked for the Generation and Compilation Makefiles.

Recompiling with Clearmake

Recompile invokes a "`clearmake -u`" which will turn OFF Clearmake's build-avoidance. At present, it does not turn OFF the code-generator's build-avoidance and a header file that does not change does not get rewritten. The step "Making Configuration Files" will also exercise build-avoidance in creating makefile fragments, and so on. If you want to guarantee that all new files get created, delete the C++/ and build/ subdirectories to remove all previous derived objects. Then, do a recompile to ensure no derived objects get winked-in.

Winked-in is not performed when recompile is selected.

Which classes get compiled

While code is generated for all classes in the project, only classes within the 'transitive-closure' of the top actor get compiled. Transitive-closure is calculated only after all classes are generated. The transitive-closure calculation is used to reduce compilation time.

Swapping between Clearmake and non-Clearmake mode

Changing the Make Type of the Active Configuration between Clearmake and non-Clearmake necessitates a recompile to force all Makefile and Makefile Fragments to be regenerated. Alternatively to this, the configuration should be duplicated, modified, and renamed.

Changing from Clearmake mode to non-Clearmake mode by activating a new configuration requires a recompile to force all IF files (intermediate-form) to get rewritten to the output directory. Clearmake mode does not produce IF files. In non-Clearmake mode, all build artifacts are written as view-private files.

Changing from non-Clearmake mode to Clearmake mode by activating a new configuration does not require a recompile. Clearmake will replace all view-private files with their respective derived object counter-parts. The content of these may be identical.

Zero-length .dep files

When using Clearmake to compile directly from the ClearCase library, ObjecTime Code Generation will create zero-length .dep files. These files are required for the code generator’s build avoidance when using Clearmake.

Using Clearmake for loadbuilders

For the most part, loadbuilders follow the exact same procedures as developers.

If during a partially-successful loadbuild, a change is made to some of the project files, the changed files should be submitted by the loadbuilders before continuing the build (even when the changes are known to be not final.). This will ensure the final product is completely traceable to versioned files, and not dependent on view-private files.

Restrictions and Limitations

This section discusses the restrictions imposed on the user when using the external build facility.

No unspecified replication factors

The use of unspecified replication factors is currently not supported with ObjecTime Developer 5.2.1 for ClearCase external builds. To use the external build capability you must manually remove unspecified replication factors.

The code generator flags any unspecified replication factors appearing on actors and ports as an error.

No ROOM compile time checking

Model compilation usually includes validation of a ROOM model prior to code generation of the model. **When using the external build facility of the ClearCase Support Enhancements, many of the usual ROOM validation checks are not performed since these are usually done inside the Toolset.** This section discusses the validation which occurs when doing Toolset-based compilation, but which is skipped when doing an external build. Care should be taken to avoid these situations when performing an external build. In general, it is suggested that when making changes to models that the verification step of the compilation process be executed within the Toolset. This will help to eliminate problems due to the external build not detecting model errors.

Violation of ROOM semantics during compilation can result in errors or warnings, or a combination of both. Errors are serious violations, which prevent code generation from taking place. Warnings are minor violations that can be due to incomplete models, or to violation of semantics, which aren't strictly enforced, for example, binding replication mismatches, and hence do not prevent generation of the code. The Toolset can optionally generate warnings for the following:

- the sum of the replication factors of all the bindings connected to a port is not equal to the replication factor of the port, or, if an interface port, the product of the port's replication factor and that of its actor reference
- a choice point does not have both a true and a false continuing transition segment
- a choice point does not have a condition defined
- an FSM state is not reachable from the initial transition
- an FSM state cannot be exited to another state other than a containing state
- a transition does not have a triggering event defined

The Toolset generates a warning if the following is true:

- more than one transition with no guard condition in the local scope of a FSM state is triggered by a particular incoming signal of a particular end port or SAP/SPP, such that there are duplicate triggers in the same scope

The Toolset reports errors whenever any of the following are true:

- an outgoing signal from one endpoint of a binding is not defined as an input signal for the other endpoint, or its data type (if not Null) is not the same or a subclass of that of the input signal
- a cyclic dependency exists between data classes. For example, class Boo references class Foo which references Boo subclass BooHoo which references Boo

The Toolset reports the following error if the model is to be compiled for the Target RTS:

- an actor reference, port or SAP/SPP is excluded in an actor subclass to be compiled for the Target RTS

The Toolset reports the following error if the model is to be compiled for the C Target RTS:

- a C actor class contains an imported actor reference

No N-way merge support

There is no support for automatic N-way merges. You must do multiple 2-way merges within the Toolset.

Time-stamp driven make support

The ClearCase Support Enhancement does not co-exist with a time-stamp driven (non-Clearmake) process (since intermediate form files will not be generated). The ClearCase/Clearmake make type mode can be used to switch between the two options. In Clearmake mode, you must use a Clearmake executable to build (typically ‘clearmake’). In non-clearmake mode, you should not use a Clearmake executable.

No shared views

The use of ClearCase shared views is not a recommended development paradigm and is not supported by OTD 5.2.1.

View-extended path names

The use of view-extended pathnames is not a recommended development paradigm, and is not supported in ObjecTime Developer 5.2.1. Any other mechanism for referring to files in another view is similarly not supported. All classes should be visible within the same view.

Exclusions

The following are not supported in ObjecTime Developer 5.2.1 ClearCase Enhancements:

- builds for the ObjecTime SimulationRTS environment are not supported using Clearmake.
- only the C++ language is supported by the ClearCase external build.
- the capabilities implemented with ObjecTime Developer 5.2.1 apply only to Clearmake and not other make variants.

Known issues

- The project file does not get a black delta when one of these changes necessitates a checkout.
- The recompile button does not turn off the code-generator's build-avoidance, although it will tell Clearmake to force recompilation of older generated files. If the code-generator had previously produced a view-private file, or a file dependent on a view-private LF file, the code-generator's build-avoidance will not regenerate the file (and hence, not update its config record).
- On Windows NT, you must use the same drive letter for viewing across the project. All references to linear-form files (including the check-out dialog, the project file, and the package files) contain the drive letter. When using ClearCase, this forces all developers to map their view to the same drive letter (typically `Z:`).
- In order to use Clearmake, and build from the library, the storage format must be textual. If the binary storage format is used, Clearmake cannot be used to build the library.

Chapter 9

Changes in Developer 5.2.1/5.2

This chapter provides additional information on the changes covered in “What's new in Developer 5.2.1/5.2” on page 2 in Chapter 1 of this Guide. For details of the changes applicable to the C and C++ Language Modules, please refer to the Changes in ObjecTime Developer 5.2.1 chapters in the *ObjecTime Developer for C Getting Started and Release Notice* and the *ObjecTime Developer for C++ Getting Started and Release Notice* respectively.

The main areas covered in this chapter are as follows:

- ClearCase Support Enhancements - 5.2.1
- Packages- 5.2
- Multi-Language Framework (MLF) - 5.2
- Make Utilities Supported - 5.2
- Data Class Inclusion - 5.2
- Deterministic Loadbuild - 5.2
- Library Management - 5.2
- Problems fixed - 5.2

ClearCase Support Enhancements

ClearCase

ObjecTime Developer 5.2.1 is designed to better support the ClearCase environment, and in particular Clearmake. It extends compilation capabilities to allow compilation outside of the Toolset, and use of Clearmake's facilities to manage dependencies and store loadbuild artifacts. For a detailed description, see "ClearCase Support Enhancements" on page 79 in this Guide.

Packages

CUPs Replacement

In **ObjecTime Developer 5.2**, CUPs were replaced with enhanced environment specification for packages. The ability to generate and compile the code for a package has been used to allow designers to reuse the results of a build and not have to perform the code generation and compilation themselves. This requirement is now satisfied by a facility to reuse the results of a loadbuild for all model components and it is no longer possible or required to compile a package independently of an update. Packages must now always be compiled in the context of an update. See “Environment and CUPs conversions” on page 18 in this guide for further details.

It is still possible to associate some environment settings with a package. But because packages must be compiled in the context of an update, not all environment settings make sense at the package-level. The package-level configuration now supports compiler flags, include files and paths, and library files and paths. The flags and inclusions are applied to all the classes in the package and child packages. The library path and files are propagated to the update-level from each package, and so apply to the entire executable.

Code Generation & Compilation Changes

ObjecTime Developer 5.2 incorporated significant changes in the way in which code generation and compilation of models is performed. These changes have been made in order to improve performance, as well as to better integrate with the customer's software development environment and processes.

The significant changes introduced in release 5.2 were as follows:

- Using timestamp (make) driven code generation and compilation
- Allow re-use of build results. The generated C++/C and object files resulting from a build of the models can be reused by designers, thus saving them the time required to compile the model themselves before commencing development work. This is supported with either the VPATH mechanism of GNU make or the sharing of derived objects when using ClearCase.
- Explicit inter-class dependencies are now tracked improving the generated code's compilation performance. The inclusion relationships between classes are now calculated based on the inter-class dependencies. This results in faster compilation performance because the number of include files which are read, when compiling a class, is now restricted to only those files that are required.

The meaning of some of the Compile Dialog Options is changed in ObjecTime Developer 5.2.1. Please refer to Chapter 22 'Model Compilation and Execution' in the ObjecTime Developer *User Guide*.

Make Utilities Supported

Effective in release **5.2 of ObjecTime Developer**, compilation and code generation are controlled by Makefiles. You have the ability to choose which make utility will be used for the code generation and compilation phases of model generation. Different make utilities expect Makefiles in slightly different formats, and some specific features of these utilities are supported by selecting particular Make types. Effective in release 5.2.1, the Make types available are `Clearmake`, `MS_nmake`, `Gnu_make` and `Unix_make`.

The Make Type should be set to `Clearmake` when using the ClearCase Support Enhancement. You can only use this Make Type when you are using ClearCase as your CM system. Clearmake is supported for various platforms. See Chapter 8 for further details. Using ClearCase as your CM system does not require that you specify `Clearmake` as your Make Type and `clearmake` as your Make Name; any other make utility will work correctly when using ClearCase as the CM system.

The Make Type should be set to `MS_nmake` when the make utility is nmake on Windows NT. This is required because of a minor formatting difference between `MS_nmake` and `Unix_make`. `MS_nmake` is the default Make Type on Windows NT.

The Make Type should be set to `Gnu_make` when using Load Build Paths for reusing load-build results. This uses `Gnu_make`'s implementation of VPATH to reuse load-build results. If you have no Load Build Paths, the `Gnu_make` Makefile format is identical to `Unix_make`.

`Unix_make` produces a Makefile which can be interpreted by almost any Make utility. `Unix_make` is the default Make Type on Unix machines.

The Make Type (and corresponding Make Name) are set from the target's Properties Editor of the Language Options in the Update Configuration. The Make Name must be a compatible executable of the chosen Make Type.

Specifying `clearmake` as the Make Name is only recommended if the Make Type is `Clearmake`. The makefiles generated for a non-`Clearmake` Make Type, when run with `clearmake`, result in the code generation being performed twice for the changed classes. Although the compilation results are correct in this circumstance, there is a significant time penalty due to the extra code generation taking place.

Recommended make utilities

The following table lists the recommended make utilities for all toolset platforms. Except where noted below, all listed make utilities are compatible with Make Type `Unix_make`. This table is neither exhaustive nor exclusive; other compatible `make` utilities may be provided with your compilation host operating system or your compiler.

Table 3

Make type	Platform	Notes
AIX make	AIX 4	
ClearCase clearmake	supported Unix platforms	version 3.2 or later
	Windows NT	version 3.2 or later
Gnu make	all Unix platforms	Version 3.71 and above is recommended. Supported by Make Type “`Gnu_make`” or “`Unix_make`”.
	Windows NT	Use version "3.74+wrs-2", available with Wind River Systems' Tornado. Supported by Make Type “`Gnu_make`” or “`Unix_make`”.
HP make	HPUX 10	
Irix make	Irix 6	
Microsoft nmake	Windows NT	Version "1.62.7022"is recommended (available with Microsoft Visual C++ 5.0). Requires Make Type “`MS_nmake`”.
Pmake (Parallel make)	various Unix platforms	Various compatible third-party distributions are available.
Sun make	Sun 4, Sun 5	includes SUNW_SPRO and SVR4 make

Data class inclusions

As of **ObjecTime Developer 5.2**, inclusions can be added to data classes. If the definition of a data class requires external inclusions, then these can be added at the data class level and need not be specified at the update level as in pre-5.2.1 releases of ObjecTime. This will improve compilation performance since it eliminates superfluous inclusions.

Deterministic Loadbuild

As of **ObjecTime Developer 5.2**, due to the way signal numbering is handled for incremental code generation, it is possible that the generated code for the two instances of the same model be different. These differences will be confined to the signal numbering values and are the result of the way the code generator adds new signals to a model. In order to avoid a complete recompile when a new signal is added, the code generator adds new signals to the end of a list. This means that the value a signal receives depends on what other signals have been added before.

To obtain a completely deterministic build, all that is required is that a total recompile be performed from a clean directory. This will ensure that the generated code will be identical and any differences between two models that are completely regenerated, will be the result of actual model differences and not caused by the order of operations as performed in the toolset (as has been the case in the past).

The order of merging affects the order in which the classes are listed in the project file. This in turn affects the order of code generation for:

- RTSystem.h
- RTSignal.h
- Data classes aggregated at the package level

The next two sections provide details on how to merge classes in a consistent manner.

Fully Specified Merge Script

Merge maintains the internal order of the added classes in the same order in which they are listed in the merge script. For example, if you use the following merge script:

```
! select AnUpdate
      merge from /whatever/mylib.otlib
            Actor1.actor *
            Actor2.actor *
            Actor3.actor *
            Protocol1.port *
            Protocol2.port *
            Protocol3.port *
            Data1.data *
            Data2.data *
            Data3.data *
      endMerge !
```

then the following is true:

- the internal order of the actor classes is Actor1, Actor2, Actor3;
- the internal order of the protocol classes is Protocol1, Protocol2, Protocol3;
- and the internal order of the data classes is Data1, Data2, Data3.

Note: The order only matters relative to other entries of the same type.

So, the following merge script results in the same internal order as above:

```
! select AnUpdate
      merge from /whatever/mylib.otlib
            Actor1.actor *
            Protocol1.port *
            Data1.data *
            Actor2.actor *
            Protocol2.port *
            Data2.data *
            Actor3.actor *
            Protocol3.port *
            Data3.data *
      endMerge !
```

Note that the internal order can be observed in a couple of ways:

1. Open a Properties Editor on the update. The class/package list displays the entries in their internal order.
2. Listing the contents of a project file.

Partially Specified Merge Script

If the required classes are not all listed in the merge script, then the merge will determine that some classes are missing, and then look in the library/directory. This process is driven by the combination of the order specified in the merge script, and the order of the references in the classes that are being merged. After retrieving the first set of missing classes, then more classes may be required, so this process is repeated until all the classes have been retrieved.

For example, assume you have a library containing the following actor classes:

- A1 which contains references to A2 and A3 (in that order)
- A2 which contains references to A4 and A5 (in that order)
- A3, A4, and A5 which are 'empty' (i.e. don't reference anyone else)

Now, if you just specify A1 in the merge script, the resulting internal order will be A1, A2, A3, A4, A5.

If you specify A1 and A4 in the merge script, the resulting internal order will be A1, A4, A2, A3, A5.

> **Note:** If you are just merging packages and then relying on the merge operation to extract all the referenced classes, then you are doing a partially specified list.

If an entry in the partially specified list is modified (for example, a reference to a new class is added to it), then the resulting order of all the classes after it could be different than they would have been when merged with the previous version of the class.

If a fully specified list was used, then there is much better control over this (for example, new classes can be added at the end of the full list). For a model that needs exactly repeatable code generation, then you should use fully specified lists.

Library Management

Library capabilities enhancements

In **ObjecTime Developer 5.2**, several enhancements were added to ObjecTime Developer's library capabilities. While some of these enhancements are intended to improve ObjecTime's ability to interwork with ClearCase, all of the enhancements can be used with any Change Management system. Even though some of the enhancements require modifications to the library scripts, the release is fully compatible with existing scripts from previous releases.

The **ClearCase** library scripts have been rewritten in Perl and are common for both Unix and NT. The new scripts take advantage of the improved Version String Handling and Sync With Library enhancements described below.

Default Location for Library Scripts

It is no longer necessary to place an `.objectime_scripts_dir` directory for Unix, or an `objectime_scripts_dir_nt` directory for NT, in every library directory. Instead a default directory location can be entered in the Library Configuration pane under the toolset menu. If a `scripts_dir` does exist in the library, the scripts in that directory will be used; otherwise, the scripts that are in the default directory will be used.

> **Note:** If the library system is down, the objectime_library_info script causes ObjecTime Developer (OTD) to hang (that is, the OTD toolset waits for an .otlib library script to terminate).

Check out and read only modifications

Classes which are not checked out from the CM system would be read only, as in a context. Before you can make editing changes, check out the class. This read only enforcement would apply to all objects which can be checked out of a library including project files and configurations. Enforcing the read only aspects of classes only applies to direct edits of the classes. If editing a class has an impact on other classes which are not currently checked out, then the toolset will in no way prevent these changes from occurring, although a delta will appear beside the affected class. This class of changes primarily applies to editing a class and having its subclasses change as a result.

This feature would be controlled by and could be disabled through a user preference.

Currently, when you check out a class, a check is performed which compares the version in the CM system to that in the update. If there is a mismatch, a warning appears which can be ignored. This will be changed so that you will be presented with a dialog which gives the option of merging in the class from the CM system or cancelling the CheckOut request. If you cancel the checkout, the result must be that the class is not checked out.

This feature will also be controlled with a user preference.

Read-only for unchecked out classes

A user preference has been added that disallows the ability to edit unchecked-out elements; see the preference "Editing Modes". Preferences are set from the Preferences Editor under the toolset menu.

Hierarchical library browsing

A Libraries pane has been added to the Library Browser which allows libraries that are contained within libraries to be easily browsed.

Sync With Library

An alternative library synchronization method has been added that relies on external library scripts objectime_sync and objectime_diff. Use of these scripts allows the update to be synchronized with the Linear Form (LF) files contained in the library directory or directories where the files have been copied. Versions of the objectime_sync and objectime_diff scripts are provided for ClearCase. In addition to the new scripts, the Library Configuration must be set to now use `objectime_library_info` for sync, in order to use external LF comparisons for synchronizing.

In addition, it is also possible to synchronize the delta symbols in the Update Browser based on the contents of the LF files. This capability also relies on the objectime_sync and objectime_diff scripts.

Since LF files contain the version string of elements that they reference, if a reference element is upversioned, the LF file that is stored in the library that represents the element that contains the upversioned element is technically out-of-date even though its design has not changed. The Sync With Library capability allows differences of this type to be ignored.

Improved Version String Handling

To support complex version strings (for example, Release1/BugFixes/1) and version branching, the objectime_library_capabilities script can be extended to turn on the NoVersionNumbers and NoVersionSort capability.

The VersonNumbers capability turns off the default assumptions made on check-out and assumes that the objectime_check_out script returns both the current and the next (upon submit) version string.

The newest version of objectime_check_out for ClearCase (as part of ClearCase integration) returns two version strings as opposed to just one. The first string is the version that is the most recent in the library and the second string is the version number on submitting it back to the library. This is done to remove the most recent comparison data from the toolset.

The NoVersionSort capability disables the default sorting performed by the Version Browser and assumes that the list should be displayed in the order output by the objectime_version_info script.

ClearCase

The ClearCase Unix scripts have been replaced with Perl scripts that are common across Windows NT and Unix. The new scripts return the ClearCase version ID (without the /main/ prefix). This allows ClearCase branching to be used without having to modify the scripts.

The versions of ClearCase supported are:

- 3.1.1 : for HPUX 10.20 and Solaris 2.6 (with a 3.1.1 patch).
- 3.2 : for Solaris 2.6, HPUX 10.20, SunOS 4.1.3 and Windows NT 4.0.

Two new library scripts objectime_sync and objectime_diff have been provided for ClearCase.

RCS

The RCS system must support the “x” option in order to interwork with ObjecTime Developer. The “x” option allows for the specification of suffixes for RCS files. Please refer to your RCS documentation to confirm this.

Linear Form

The following summarizes the changes made to the linear form grammar for release 5.2.1. Customer tools which process linear form will have to be modified to accommodate the changes in the grammar.

Added tokens

```
BLACKBOX
DEPENDENCIES
LOADBUILD
OUTPUT
PROJECT
RECTILINEAR
STEREOTYPE
```

Added productions

```
optStereotype/* NEW in 5.2.1 */
   : /*empty*/
   | STEREOTYPE TEXTSTRING
   ;

optDependencies         /* NEW in 5.2.1 */
        : /*empty*/
        | DEPENDENCIES ‘{‘ dependencyList ‘}’
        ;

 dependencyList
   : /*empty*/
   | dependencyList dependencyItem ‘;’
   ;
```

```
 dependencyItem
   : DEFINE classType className optInPackage optLibraryVersion optSte-
reotype optDescription
   ;

optStereotype           /* NEW in 5.2.1 */
        : /*empty*/
        | STEREOTYPE TEXTSTRING
        ;
graphicLine             /* NEW in 5.2.1: optRectilinear */
       : graphicSpec optWidth optSmooth optRectilinear FROM pointsList
        ;

optRectilinear          /* NEW in 5.2.1 */
        : /*empty*/
        | RECTILINEAR
        ;

optCompilationPath                      /* NEW in 5.2.1 */
        : /*empty*/
        | PATH TEXTSTRING
        ;


projectSpec                     /* NEW in 5.2.1 */
        : PROJECT projectName
                        optLibraryVersion
                        optDescription
                        optDependencies
                        projectPublicComponents
                        threadsSpec
                        libraryPaths
                        outputPath
                        loadBuildPaths
                        ';'
        ;

 projectName
   : IDENT
   ;

 projectPublicComponents
   : /*empty*/
   | PUBLIC '{'
               projectComponentList
               activeEnvironment
```

```
                  optTopActor
                  '}'
      ;

    projectComponentList
      : /*empty*/
      | projectComponentType componentName optLibraryVersion optDerived-
   FromSuperClass optInPackage ';' projectComponentList
      ;

    projectComponentType
      ;

    activeEnvironment
      : ACTIVE ENVIRONMENT IDENT ';'
      ;

    optTopActor
      : /*empty*/
      | TOP ACTOR actorClassName optLibraryVersion ';'
      ;

    libraryPaths
      : LIBRARY PATHS '{' pathList '}'
      ;

    loadBuildPaths
      : LOADBUILD PATHS '{' pathList '}'
      ;

    pathList
      : /*empty*/
      | PATH libraryPath ';' pathList
      ;

    outputPath
      : OUTPUT PATH libraryPath ';'
      ;
   optEnvironmentSpec
           : /*empty*/
           | environmentSpec
           ;

   environmentName
           : TEXTSTRING
           ;
```

Changed productions

```
modelEntitySpec
        : actorClassSpec
        | protocolClassSpec
        | dataClassSpec
        | constantSpec
        | packageSpec
        | requirementSpec
        | mscSpec
        | environmentSpec              /* NEW in 5.2.1 */
        | projectSpec                  /* NEW in 5.2.1 */
        ;

realValue
        : NUMBER                       /* NEW in 5.2.1 */
        | REAL_NUMBER
        | constantName optLibraryVersion
        ;

dataClassSpec
        : DATA CLASS dataClassName
                        optLibraryVersion
                        optDerivedFromSuperClass
                        ISA dataTypeSpec
                        optDependencies        /* NEW in 5.2.1 */
                        ‘;’
        ;


choiceSpec                     /* NEW in 5.2.1: inclusionsSpec */
        : CHOICE properties choiceTypes inclusionsSpec methodsSpec
        ;

enumeratedSpec                 /* NEW in 5.2.1: inclusionsSpec */
        : ENUMERATED properties enumeratedValues inclusionsSpec meth-
odsSpec
        ;
sequenceSpec            /* NEW in 5.2.1: inclusionsSpec */
        : SEQUENCE properties fields inclusionsSpec methodsSpec
        ;


protocolClassSpec
        : PROTOCOL CLASS protocolClassName
                        optLibraryVersion
```

```
                            derivedFromOrService
                            properties
                            inMessagesSpec
                            outMessagesSpec
                            mscsSpec
                           optDependencies             /* NEW in 5.2.1 */
                                            ';'
        ;
actorClassSpec
        : ACTOR CLASS actorClassName
                            optLibraryVersion
                            optDerivedFromSuperClass
                               optExclude
                            properties
                            actorInterfaceSpec
                            actorImplementationSpec
                            actorConfigurationSpec
                           optDependencies             /* NEW in 5.2.1 */
                                            ';'
        ;


localIncludeItem                    /* NEW in 5.2.1: optStereotype */
        : DEFINE inclusionName optActor optStereotype properties
        ;

defaultPackageDefinition
        : optCompilationPath                    /* NEW in 5.2.1 */
          properties
          packagePublicComponents/* changed */
          packagePrivateComponents/* changed */
          packageSignals
          packageActors
          optEnvironmentSpec
          optThreadsSpec
        ;

packagePublicComponents
        : /*empty*/
        | PUBLIC '{' packageComponentsList '}'
        ;

packageComponentsList
        : /*empty*/
        | packageComponentsList packageComponent ';'
        ;
```

```
packageComponent
        : packageComponentType componentName optInPackage optLibrary-
Version optPermissions
        ;

packageComponentType
        : classType | PACKAGE
        ;

componentName
        : IDENT
        ;

environmentSpec
         : ENVIRONMENT environmentName optLibraryVersion bracketedEx-
pression ‘;’
        ;

compilationPackageDefinition
        : ISA COMPILATION defaultPackageDefinition
        ;

anyValue
        : IDENT
        | NUMBER
        | REAL_NUMBER
        | TEXTSTRING
        | boolean
      | enclosedExpression
       | bracketedExpression
       | ‘&’ | ‘*’ | ‘@’ | ‘\\’ | ‘:’ | ‘.’ | ‘,’ | ‘!’ | ‘?’
       | ‘=’ | ‘<‘ | ‘>’ | ‘+’ | ‘-’ | ‘#’ | ‘;’ | ‘/’ | ‘~’
       | ACTIVE | CONFIGURATION | FIELDS | PROPERTY | PROPER-
TIES | SEQUENCE | SYSTEM
       | THREADS | UNDEFINED | VALUE | VALUES | VERSION | VER-
SIONS
       | ‘[‘ ‘]’
       ;
```

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 10

General Information

Toolset Memory Requirements

This section discusses the memory requirements for the ObjecTime Toolset and the disk space requirements for saved models. This information will help you better understand how memory is utilized in the ObjecTime Toolset and to help plan system requirements for the development environment before a large project is started.

This section does not discuss the size of the generated executable for an ObjecTime model (for an estimate of the size of the generated executable for a C++ model see the 'ObjecTime Model Size Estimation' section of the *C++ Target Guide*).

The platform-specific sections of this guide deal with the minimum requirements to run the Toolset on each platform (see System Requirements in this chapter, and System Requirements in the 'Getting Started with Unix' chapter). In addition to the basic memory requirement to run the Toolset, additional memory will usually be required to build and run models. The amount of memory required will depend on the size of the model. This section will deal with trying to estimate the memory requirements for models.

Memory consumption for ObjecTime models varies with many factors. The memory consumption varies with the number of the various design objects used, such as:

- actor classes
- protocol classes
- data classes
- configurations and their attributes
- packages
- actor references
- bindings
- ports
- states
- transitions
- choice points
- events

- functions
- inclusions
- ESVs
- MSCs and their contents
- probes and their attributes
- etc.

Memory consumption also varies with the amount of code (RPL, C and C++) entered in transitions, guard conditions, functions and inject messages, as well as strings entered in places such as configuration parameters and the description field in property editors.

Perhaps not so obvious, memory consumption also varies with things like:

- the number of equivalences defined in a model
- the number of requirements links
- the number of inflection points on bindings/transitions
- the depth of inheritance class hierarchies
- the number of excluded objects in a subclass
- the types of edits performed on a model in the current session
- the number and types of windows open on various components in the model
- and so on.

Memory consumption will also increase once a model has been compiled. Given all of these variables, it is very difficult to give an exact formula for the memory requirement of any given model. Instead, we have studied several typical models to offer an estimation of the memory required at a macroscopic level.

Typical model memory usage

We have studied several typical ObjecTime models. While the data given here is typical for the models we have studied, care should be taken in how these numbers are used. In particular, it is possible in cases of some specific models to obtain sizes which are off by an order of magnitude from the numbers presented here, especially if the model in question does not meet our definition of a “typical” model.

The models which we have studied are typically characterized as follows:

- X Actor classes
- between 0.4*X and 0.7*X Protocol classes
- between 0.7*X and 3*X Data classes
- on average each actor class has between 70 and 200 uncommented lines of code

Given this characterization, the in-memory size of these models is typically between 40 KB and 130 KB times X (for example, if X is 100 for a particular model, then the toolset will require between 4 MB and 13 MB to store the model in memory). Note that this sizing is done before compilation.

Memory usage in operations

Operations such as activation, passivation and merging will make copies of various internal data structures before and after the operation. At times, these operations will make complete copies of the ObjecTime model being operated on. Because of this, these operations will increase memory requirements during the execution of the operation (often this increase is equal to and sometimes double the memory required to hold the model in memory). Keep in mind that this memory is only required for the duration of the operation and will be released back to the ObjecTime memory pool, for use by other operations/models, once the operation is complete. Note that, at this time, memory allocated by ObjecTime Developer from the underlying OS will remain allocated until the Toolset quits. The Toolset will not dynamically release unused memory back to the OS.

Context vs. update memory usage

Another issue related to memory usage is that relating to contexts and updates. When contexts are being used, the typical work scenario has the designer activate the context and then create an update from the context, after which all edits are performed on this update. It should be noted that contexts and updates are the same size while in memory, and creating an update from a context essentially creates a duplicate copy of the context. One method to avoid this duplication, and therefore decrease memory consumption, is to remove the context from memory and work only on the update.

Two procedures for doing this are as follows:

- start with an activated context in memory with an update created from it
- passivate the update
- delete the update
- delete the context
- activate the update
- save and exit the session
- restart the session

or alternately:

- start with an activated context in memory with an update created from it
- passivate the update
- abandon the session
- restart a clean/empty session
- activate the update

Through the use of this technique, only one copy of the classes are stored in memory. Note that this will have the side effect of causing the update to be associated with 'TheContext' rather than the originating context. One consequence of this new association is that if the original context had a library associated with it; this information is lost. Putting this library path into TheContext's library entry will serve as a workaround for this situation. One other consequence of the association with TheContext is that the 'Show Changes' menu item on the Update menu will now return different results than it would have when the update was associated with the original context.

Model file sizes

Passivated (file) versions of models (updates and contexts) are smaller than the in-memory size of the model. The in-memory size of a model is anywhere between 2 and 5 times the size of the passivated file. As is usual for memory sizing issues, this is an approximation of the size ratios. It is possible to have different ratios.

> **Note:** This ratio only applies to passivated updates and contexts. Since project files contain references rather than the objects themselves (as do updates and contents), they will be much smaller in size.

Summary

The results discussed above will now be presented in a more compact form. Keep in mind that this discussion applies to models characterized by our "typical" model definition, and that actual memory usage may vary from the numbers presented here.

1) Given a typical model with X actor classes, the in-memory size of this model will be:

Y = between 40KB * X and 130KB * X

2) If both a context and update of this model are to be stored in memory, this would require 2*Y of memory. By removing the original context, only Y of memory would be required.

3) Operations involving this model may require an additional Y to 2Y of memory needed only for the duration of the operation. This memory is not released back to the OS but will be available for use by other toolset operations and model storage.

4) The passivated file (update or context) size of this model should be between Y/2 and Y/5.

Microsoft Visual SourceSafe (MSVSS)

The following notes apply to using Microsoft Visual SourceSafe (MSVSS) as a library system:

1) The ObjecTime library scripts for Microsoft Visual SourceSafe (MSVSS) do not allow the use of the 'multiple checkouts' feature of MSVSS.

2) ObjecTime scripts currently set the file type to binary format regardless of the setting within the toolset. Explicitly changing the file type to text, will cause problems with some of the library scripts.

3) If a file is manually deleted from an ObjecTime project in MSVSS without setting the 'Destroy Permanently' option, and later a user tries to create a file with the same name, then the destroyed file will be restored which will leave ObjecTime in an out of sync state.

4) The same user (on the same or different systems) should not perform more than one MSVSS library interface operation (even in different MSVSS projects at the same time). Two or more library interface operation might collide with each other and cause incorrect results and error messages. (PR4292)

Limits

Model Limits (RPL, C and C++)

The total number of ports + SAPs + ESVs + actorRefs per actor class <= 256. This includes inherited components.

RPL Code Editor

- 2000 lines per code segment

RPL Limits

- 256 method variables (arguments and temp vars)
- 256 literals (strings, numbers, symbols, characters, message selectors, referenced class names, IF-THEN/ELSE/WHILE/FOR bodies)
- 256 levels of nested IF/WHILE/FOR statements
- There is a limit on the size of the compiled code generated for the inside of IF/WHILE/FOR statements. In practice this is not a problem, but is still a possible limit.

Simulation Services Library Limits

- A maximum of 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.

Special Notes and Reminders

- ObjecTime Developer conventions for environment variables are as follows:

  General: Toolset uses host conventions - $<name> in UNIX and %<name>% in NT

  Exceptions:

  Environment Browser supports $ for both UNIX and NT. It does not support %

  Package path supports $ for both UNIX and NT. It does not support %
- To improve performance when you do a merge, especially with large systems, do NOT select “Cancelable” which is the default on the Merging dialog. If you do select “Cancelable” ObjecTime will make a copy of the entire update and merge into this copy. If the merge is successful, the copy will replace the original. This will cause the entire system to be recompiled the next time you compile, as if you had selected “Recompile”. Thus merging in as little as one class can cause your entire system to be recompiled. Therefore, we generally recommend that you deselect the “Cancelable” option.
- The external editor started up by ObjecTime must be a window based editor. On Unix, for example, one must use “xterm -e vi” instead of “vi”. On Windows NT, one must use “Wordpad” instead of “edit.com” for example.
- Version 1.1 of an object in a library is often used as a dummy placeholder to reserve the object name within the library. You will get inconsistent error messages if you try to merge a placeholder version 1.1 class, package or requirement from a library, for example, “Error occurred when extracting a version”, “Error extracting a requirement”, “Error reading from <requirement>”, “Error checking requirement definition header”, or “Error bad header for requirements file”. (PR1560)

- Library check in and check out of a class will cause a refresh of any library browsers open on that library — **once for each class accessed**. To speed operations where many classes are either checked in or checked out, close the library browsers before beginning the operation.
- ObjecTime does not guarantee proper operation when an image (.otd) file is created in one directory or on one host platform and is then opened in a different directory or host platform. If you need to move updates between directories or platforms, you should passivate your updates and activate them into a different workspace at the destination location.
- 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 an actor, the 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.
- ObjecTime currently assumes that library scripts on Windows NT are written in Perl. On Windows NT, each library script is invoked using the command '`perl -w <script_name_and_arguments>`'.
- Users should not rename the ObjecTime image files. For example, renaming the image file "ObjecTime5.2.otd" to "MyImage.otd" will prevent the image from being loaded.
- In order to support building the linear form parser on multiple platforms, GNU Bison (version 1.25 and later) becomes the default parser generator and flex (version 2.5.4 and later) the default scanner-generator. Both of these tools are publicly available on a number of sites on the Internet.
- ObjecTime uses a very simple algorithm for comparing version numbers. It assumes that a version number is a series of segments separated by periods (for example, '1.main.3'). Each segment can be either a (positive) integer or a string. The comparison of two version numbers is a pairwise comparison of the segments. If the two segments being compared are all digits, then the expected numeric comparison is done (for example, '1.30' is higher than '1.4'). If either of the segments contains characters other than digits, then a string comparison is done (for example, '1.30' is earlier than '1.4a'). The comparison also assumes that both version numbers have the same number of segments. If this is not true, then it will only compare the smaller number of segments and return an answer based on that.

Perl Information

- Perl plays a number of roles in ObjecTime Developer 5.2.1.

  1. To produce a dependency list for Target Services Library source files to permit recompilation of the libraries by customers following any customization/configuration change of the Target Services Library. Perl must be in the search path for dependency list (a.k.a. "depends" file) generation to work. A new "depends" file can be generated on the host where ObjecTime is run or on any other host where Perl is already installed.

  2. To pre-process our generated make files on UNIX hosts in preparation for their use in building models for Windows NT targets using the Visual C++ compiler.

  3. To permit execution of our make files on Windows NT targets when compiling and linking on that platform using Visual C++.

Perl is included with the ObjecTime release. The version of Perl in use at the time of release was 5.002 beta3 on SunOS, Solaris, HPUX 10, IRIX and AIX. On Windows NT, the version in use at the time of release is 5.003_07. On QNX, the version supplied for QNX is “5.002 with DEBUGGING”

To run the Perl scripts provided with ObjecTime, make sure the path is properly set up on your Unix or Windows NT platform.

Building a Model with VC50 Debugging Information

NEW in 5.2: For Windows NT users that are developing with the Visual C++ 5.0 tools, ObjecTime Developer now offers integration with the Visual C++ source debugger. This permits setting and clearing of transition code source breakpoints at run-time within ObjecTime Developer.

Before you compile your ObjecTime model:

- Ensure DevStudio’s bin and sharedIde\bin are in your path.
- Read ‘Pure Windows NT Installation’ under ‘Supported Network Configurations’ in the Introduction of this Guide.
- Read Appendix E, Integrating Developer Studio on Windows NT, in the *C++ Language Guide*.

Chapter 11

Troubleshooting

Troubleshooting Unix

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.

CD read errors

If you are installing from a CD-ROM drive across the network and you are using a fast CD-ROM device, you may see some tar read errors during the installation process. To avoid the problem, either copy the CD contents to a local disk drive and run the installation from there, or run `setup.sh` from the machine to which the CD-ROM drive is connected.

Incorrect key mappings

If you are running ObjecTime from an HP workstation or from an NCD X-terminal, and some keys on your keyboard are not working. “Optional settings” on page 48 for more information.

SCCS/RCS files missing

Upon starting up ObjecTime, you may receive several messages indicating that several SCCS and/or RCS commands cannot be found. If you are not using the ObjecTime library system, then this is of no concern. Otherwise, you may wish to speak with your UNIX administrator to see about obtaining the missing files.

Cannot allocate color

This problem can occur if there are insufficient color resources for all of the X Window System applications that you are currently running. Typically a message will be issued by the X Window System on start-up of an application that cannot obtain the required colors. In this case, we recommend that you start up the X Window System with a static color palette.

For Sun OpenWindows 3.2, this can be done by using the following command when starting up:

```
openwin -dev /dev/fb staticvis
```

Configuring OpenWindows for use with a Gray-Scale Screen

The following indicates the options that should be used when starting up OpenWindows:

OpenWindows 3.2:

-dev /dev/fb grayvis staticvis

OpenWindows 3.3:

-dev /dev/fb defclass StaticGray grayvis

Font problems

If a dialog is presented which indicates that fonts cannot be found, then the font path was likely not set correctly. Please see “Starting ObjecTime Developer 5.2.1” on page 50. Refer to the step where fonts are set using the xset command.

Setup of Fonts for X-Terminals

The ObjecTime fonts will not be set properly if the user is on an X-Terminal which obtains its boot files from a file server that does not have access to the $OBJECTIME_HOME/fonts directory. In this case, the fonts should be copied to the file server from which the X-Terminal obtains its boot files. Alternately, the X-Terminal can be changed to obtain its boot files from the same file server upon which the Developer release files have been installed.

Setup of Fonts for PC and Mac Based X-Terminal Software Packages

To use the ObjecTime fonts with your X-Terminal package, you will first need to convert the delivered fonts into a format acceptable to your software. Then, you will need to install the fonts and associate the font aliases with the font files. The steps are as follows:

1. **Converting fonts** - Most X-Terminal software packages have a utility to convert fonts in ‘bdf’ format to an acceptable format. Each software package is different in the exact procedure to do this. Consult your manual on how to do this. You will find a ‘bdf’ version of the ObjecTime fonts in the ‘fonts’ directory, which is inside the ObjecTime installation directory ($OBJECTIME_HOME/fonts/bdf). This directory contains the five font files in bdf format. Use your X-Terminal’s conversion utility to convert the bdf files into the form used by your software.
2. **Installing fonts** - Refer to your X-Terminal’s manual for the procedure to install the newly created fonts. For some packages, you copy the files to a specific directory, for others, you select menu items to install the font files.
3. **Associating font aliases with font files** - The ObjecTime code refers to the fonts by the ISO standard font name. These are long cryptic strings that describe the font. This information is not part of

the font file, therefore your X-Terminal will have to be told about this information in a separate step. This information is usually found in the 'fonts.alias' files for standard X fonts (e.g., see $OBJECTIME_HOME/fonts/sun/fonts.alias). This information is entered by associating the short font name (e.g., 'otl10r') with the long ISO string. The method for doing this varies with the different software packages. Some packages have menu items and dialog boxes to manually enter the information. Other packages allow you to place the information in a file. Refer to your manual for the exact procedure for doing this. The aliases you need to specify are as follows:

font alias

otl10b -objectime-otl-bold-r-normal--12-100-75-75-p-60-iso8859-1

otl10s -objectime-otl-ultrabold-r-normal--12-100-75-75-p-60-iso8859-1

otl10r -objectime-otl-normal-r-normal--12-100-75-75-p-60-iso8859-1

otl10t -objectime-otl-ultralight-r-normal--12-100-75-75-p-60-iso8859-1

otl10i -objectime-otl-normal-i-normal--12-100-75-75-p-60-iso8859-1

You need to type these aliases exactly with no spaces. The best way to enter these strings is probably to open up one of the 'fonts.alias' files with a text editor, and copy the strings directly from the file into the file or dialog boxes associated with your X package.

Note that when using Mac/PC based X terminal packages, you probably don't want to use the Unix 'xset +fp ...' command to set the font path, as this may cause problems when starting up ObjecTime.

Font Installation Diagnostics

Once the fonts have been installed, you can perform these diagnostics to verify that they are installed correctly.

Try the Unix command '**xlsfonts | grep otl**' and verify that the output is:

-objectime-otl-bold-r-normal--12-100-75-75-p-60-iso8859-1

-objectime-otl-normal-i-normal--12-100-75-75-p-60-iso8859-1

-objectime-otl-normal-r-normal--12-100-75-75-p-60-iso8859-1

-objectime-otl-ultrabold-r-normal--12-100-75-75-p-60-iso8859-1

-objectime-otl-ultralight-r-normal--12-100-75-75-p-60-iso8859-1

otl10b

otl10i

otl10r

otl10s

otl10t

If not, then the ObjecTime fonts cannot be found. Retry the font installation procedure. Once the fonts are found, then the Unix commands:

xfd -fn otl10b

xfd -fn otl10i

xfd -fn otl10r

xfd -fN OTL10s

xfd -fn otl10t

can be used to open up windows on the five fonts to verify that they look correct. Here is a quick description of the characters in the fonts to verify that you have the correct font:

otl10b - bold characters: upper/lower case, four arrows and ‘ent’

otl10i - italic font

otl10r - upper/lower case characters, special icons (document, check mark, stop sign, yield sign, etc.)

otl10s - icon only font: document, check mark, stop sign, yield sign, etc.

otl10t - template font: each cell (character) is really a multi-character bitmap (e.g. ‘<keyword>’, ‘<type-name>’, ‘<literal>’)

If the fonts look correct, there should be no problem running ObjecTime.

- **After starting my first ObjecTime Developer session, the fonts look like they don’t have spaces between the words**.

Possible cause: New fonts conflict with your previous installation of ObjecTime Developer 5.X, used through NFS software.

Solution: Point your NFS client software to use newly installed fonts and removed previously compiled ones. Restart the system and restart ObjecTime Developer session.

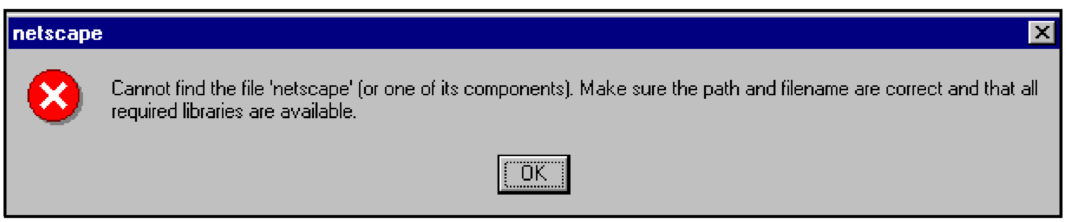
Socket connections:

- **Cannot open socket connection to external Layer Service master.**

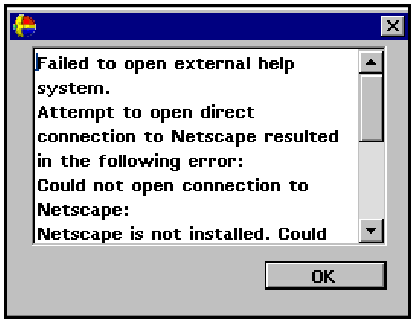
Upon initialization of a model that contains SAPs, the toolset times out for each socket connection attempt (one for each SAP) and displays an error dialog of "Cannot open socket connection to external Layer Service master." If there is a large number of SAPs, the user interface has the appearance of hanging. It does not respond to mouse input or nor are windows refreshed until all attempts are completed. This is caused when either the toolset or the rtsController runs out of socket descriptors. To fix this simply increase the number of available socket descriptors using the “limit” command before starting ObjecTime.

Online Help

If the UNIX platform in which the toolset is running, does not have a browser installed, the following error message will be displayed when the toolset is launched.



You must install a compatible browser for online help to function. Please refer to “Installing Netscape Navigator” on page 43 in this guide for installation instructions. If you had previously been running an earlier version of Netscape, it may be necessary to uninstall the older version of Netscape prior to installing version 4.04 from the ObjecTime Developer CD. When a help request is issued from the toolset without a validly configured browser, the following message will be displayed. Once again,



please check to ensure either Netscape Navigator has been configured properly as per the instructions in the “Installing Netscape Navigator” on page 43 in this guide.

License Server Upgrades

If you need to replace the ObjecTime license server, or perform a disk replacement on the server, it will be necessary to have new license keys generated by ObjecTime support. After the upgrade and ObjecTime installation have been completed, please refer to “License Registration” on page 58 in this guide for further information.

Troubleshooting Windows NT

Screen flicker

If the colors in ObjecTime Developer 'flicker', when switching between applications then your system is set for 256 colors. Increase the number of colors in your Display settings.

Install/Uninstall Problems

- **Install will not proceed for non-Administrators.**

The user doing the install must be in the Administrator group to run ObjecTime Developer 5.2.1 Install. There is a concept of Administrator privileges on the System in NT, rather than network administrator. To add the user administrator privileges, you have to login as an administrator for the system (not a network administrator) and run "User Manager" utility in Start Menu\Programs\Administrative Tools(Common). Select Administrators group and add the user to the group. Refer to Windows NT documentation for further details.

- **Uninstall leaves incrementally installed ObjecTime files on the disk**.

Always run the uninstall program before re-installing ObjecTime Developer. If components have been incrementally installed they will not be removed by the uninstall but must be removed manually.

- **Install fails trying to create rtsController.exe**

If you try to install into a directory that previously held an ObjecTime installation and one (or more) of the executables is STILL RUNNING then this error will occur. Simply do the following:

1. Reboot the workstation.
2. Delete the partial install (ensure that all the files are deleted).
3. Start the install again.

- **Font installation error**.

If the ObjecTime installation procedure returns an error (see below) on Font install, it is due to Windows NT not allowing the installer to remove previously installed fonts.

You may do one of the following:

- reboot or reinstall ObjecTime
- follow the instructions in the dialog (after the next reboot)

- **Uninstall of “old” ObjecTime Release causes run failure of 5.2.1 in the following ways:**.
  - Loss of license manager from system
  - Loss of environment settings for OBJECTIME_HOME and PATH inclusing of %OBJECTIME_HOME%\BIN\WINNT4

  This is due to problems with multiple ObjecTime Releases sharing the same registry entries. This only happens if you need to uninstall a previous release of ObjecTime Developer after installing a new one. To avoid this problem, use the following procedure to uninstall a previous release

  - For the scenario where you have:
    - installed OT5.1
    - then installed OT 5.2.1

    if you want to uninstall OT 5.1, you should:

  1 Do a user setup in OT 5.1.

  2 Uninstall OT 5.1.

  3 Do a user setup on OT 5.2.1.

- **Listbox is empty**

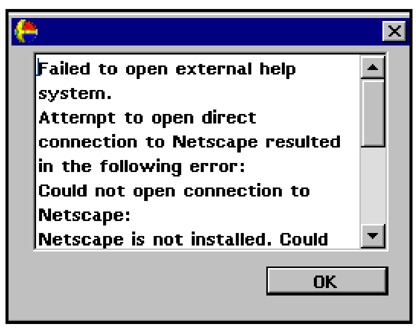
  During setup the user can go to the Directory Browser to select the destination directory for ’ObjecTime Install’. When the user enters the browser a second time (for example: click Cancel and click Browse again) sometimes the user will not see mapped network drives, the listbox will be empty. This is a known limitation related to the InstallShield software, the installation utility used by ObjecTime.

  To work around the problem, press then network button on the browser and press cancel in the network dialog to go back to the Browser.

Online Help

If the Windows NT platform in which the toolset is running, does not have a browser installed, the following error message will be displayed when the toolset is launched.

You must install a compatible browser for online help to function. Please refer to “Installing Netscape Navigator” on page 23 or “Configuring for use with Internet Explorer 4.0” on page 24 in this guide for installation instructions. If you had previously been running an earlier version of Netscape, it may be necessary to uninstall the older version of Netscape prior to installing version 4.04 from the ObjecTime Developer CD. When a help request is issued from the toolset without a validly configured browser, the following message will be displayed. Once again, please check to ensure either Netscape Navigator or



Internet Explorer has been configured properly as per the instructions in the “Installing Netscape Navigator” on page 23 or “Configuring for use with Internet Explorer 4.0” on page 24 in this guide.

Compilation problems:

Compile fails on valid C++ model for TargetRTS or SimulationRTS with VC++ 5.0 / 6.0

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 that the INCLUDE and LIB environment variables are set (for example, they could be set for the user who installed VC++, but not set for another user). Set the environment variables. Refer to the VC++ 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.

Windows NT Compilation Command Line Limits

In 5.2.1: 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, Visual C++ 6.0, 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, compilation options, the local working directory and include directories. The only compiler that has a measurable limit is Microtec's VRTX compiler. 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 4 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.

MSVSS Library problems

MSVSS Library Interface fails to find scripts or project

Possible cause: ObjecTime library directory incorrect for Windows NT.

Solution: In order for ObjecTime to recognize a directory as a valid library to be accessed with the script interface, the directory name must end with ‘.otlib’. This directory will sometimes be referred to as a ‘script library’. A script library must contain an entry that is a directory containing the scripts required to interact with that library. **This subdirectory must be named ‘.objectime_scripts_dir’ for UNIX and ‘objectime_scripts_dir_nt’ for Windows NT.**

Visual SourceSafe is a project-oriented library system as opposed to the directory-oriented library system assumed by ObjecTime. In order to use Visual SourceSafe, an extra step is necessary to map the directory name to a project name. **This is done by creating a file named ObjecTimeMSVSSProject and placing it in the .otlib directory (not the scripts subdirectory). This file should specify the name of the project used within the Visual SourceSafe library for any classes/packages that are placed in this library**. An example of this file is provided in the directory %OBJECTIME_HOME%\bin\LibraryInterface\forMSVSS.

MSVSS Library Interface commands fail to execute with the message ’Cannot execute MSVSS command ’

Possible cause: MSVSS binaries are not on the path.

Solution: Add MSVSS binaries directory to the path and restart ObjecTime Developer session.

MSVSS Library Interface commands fail to execute with a message ‘Cannot create project <Project Name> ’

Possible cause: You are not configured as a SourceSafe user.

Solution: Each user has to be configured, before using SourceSafe, through SourceSafe Administrator. Request from your MSVSS administrator to add you as a user and restart ObjecTime Developer session.

MSVSS Library Interface commands fail to execute with the message ’Could not open ObjecTimeMSVSSProject’

Possible cause: ObjecTimeMSVSSProject file is missing from the library directory.

Solution: Each MSVSS library directory should contain this file to point ObjecTime Developer to which MSVSS project to use. Sample of the file is available in <OBJECTIME_HOME \bin\Library-Interface\forMSVSS>. Copy the file into the library directory and, if desired, modify the default project name stored in it.

DLL loading problem

On starting ObjecTime Developer on Windows NT, user sees the error message “ObjecTime encountered an error while attempting to load a dynamic link library called: EMERG.DLL.”

Possible cause: The user has overridden the environment variable OBJECTIME_HOME by defining a user variable in the system environment.

Solution: Installation automatically sets the OBJECTIME_HOME variable for the NT user. The user is therefore advised not manually set the variable.

Mailing exception files

Windows Messaging must be installed before either exception files or comment files are automatically mailed. The format supported is SMTP (internet format addresses).

Starting ObjecTime Developer

The image file found in the working directory cannot be renamed from ObjecTime5.2.otd. If it is renamed, it cannot be started by double-clicking or by other means and it must be renamed to the original name.

Troubleshooting License Manager

On Windows NT, the License Manager utilities fail to execute from the command prompt

License manager utilities must be invoked from the ObjecTime command prompt available from the ObjecTime command group in the Start Menu. This is because the commands rely on certain environment variables, which will not be set in a normal command console but will be set by the “ObjecTime Developer Command Prompt”.

License manager fails when running on a stand-alone Windows NT machine

If the license keys do not work for a stand-alone system, and the logfile indicates that they are not valid, then the IP address used to generate the keys may be different than the address for the network card installed on the machine. If this happens, make sure the IP address supplied to ObjecTime for key generation is the one for the installed network card.

> **Note:** a stand-alone system must have an ethernet card installed in order to be able to run the license manager.

License file corruption

If licensing suddenly fails, and the logfile indicates that the license files are not valid, then the license files may have become corrupted. The solution is to reinstall the license key files with the activateKey command. It is recommended that the original keys from ObjecTime always be retained in case this happens.

It is also recommended that the account used to install the license keys be the same one used to run the license manager.

Dialup networking conflicts

If licensing on a stand-alone machine suddenly stops working, then this may be an interaction with dialup networking. If dialup networking is activated, the dynamically assigned IP address may conflict with the IP address of the network card. This can cause the license manager to think it is running on a machine for which the license keys are not valid.

If this happens, dialup networking can be deactivated and the problem should be corrected. If it is required that dialup networking be used on the same system as the one running the license manager, then the system administrator should configure dialup networking to use a static IP address that is the same as the one on the installed network card.

This problem can be avoided if the network card is installed before the dialup networking. If installation is done in this order, the activation of dialup networking should not conflict with the IP address obtained by the license manager from the network card.

Logfile creation failure

Failure to create a license manager logfile can be caused by the failure to specify a full path name and file name. Relative paths will not work when specifying the logfile location and name.

Key activation failure

If the license manager key files are not created when the activateKey command is run, then make sure that a full path name is specified for the license key directory. Also, make sure that the account from which activateKey is being run has write permission for the license directory.

Inactive License Manager

Immediately following the installation of the license manager on Windows NT, the license manager will not be running. The license manager will be started automatically the next time the machine supporting the license manager is restarted. An alternative to restarting the machine is to start the license manager manually. This can be done using the ElanLM control panel accessible from the control panel window.

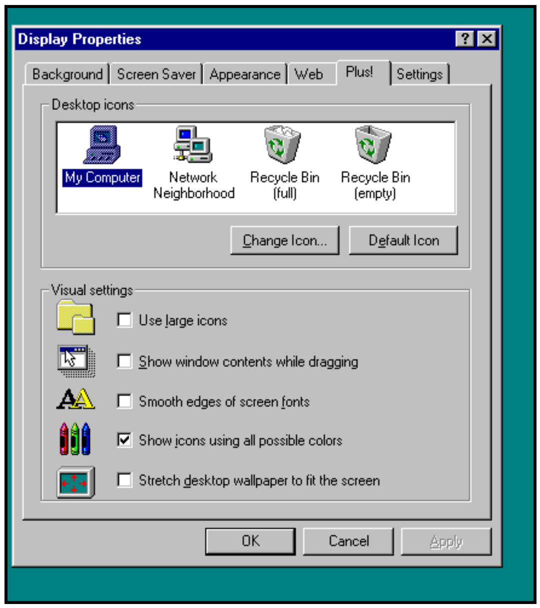
The `TZ` environment variable should be set to a valid value. Otherwise when the time changes between daylight savings time and standard time, the license files will become invalid and will have to be reinstalled using `activateKey`.

License Server Upgrades

If you need to replace the ObjecTime license server, or perform a disk replacement on the server, it will be necessary to have new license keys generated by ObjecTime support. After the upgrade and ObjecTime installation have been completed, please refer to “License Registration” on page 58 in this guide for further information.

ICON Display

If the ObjecTime icon which is displayed under Windows NT does match the documentation, you may need to increase the number of colors which Windows NT uses to display the program icons. In order to correct, you will need to bring up Display Properties from the Desktop of your NT workstation and check the box as displayed in the following figure.



Appendix A

Developer 5.2.1 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 Developer 5.2.1 directory and its sub-directories contain all the individual files that comprise the particular release. Some of the files and directories included here are:

`<INSTALL>/Developer5.2.1 (this is the top level)`

`Help`

This directory and its sub-directories contains the on-line Help, as well as an on-line version of the HTML conversions of all ObjecTime manuals (complete with hypertext links).

`Training`

This directory and its sub-directories contain model updates for the Tutorial (RPL, Batch, C and C++ examples) together with those for additional user exercises.

`specials`

This directory will contain any special patch patches that may be issued for your installation.

`image/ObjecTime5.2.otd`

This is called the image or session file, and is a combination of all the code and data corresponding to the executing ObjecTime program. This file will be copied by every user (through the use of the create_objectime_dir shell script on UNIX, or through the Launcher under Windows NT) into the user's own private directory. All ObjecTime models will be stored automatically in this file whenever the session is saved. Note that you should also save each model Update, using the ObjecTime passivation mechanism, for backup purposes.

`bin`

The bin directory holds the ObjecTime executables and various scripts. The three main executables/scripts for running ObjecTime are described below (ObjecTimeVM.*, objectime, create_objectime_dir). Additional scripts include copy_objectime_dir, otdebug, otprint, objectime_viewer. Executables/scripts for licensing are: ObjecTimeKeyInfo, startLicenseManager, killLicenseManager, killUserLicense, licenseInfo, activateKey, serverUsageReport.

$OBJECTIME_HOME/bin also contains subdirectories for each of the supported workstation platforms.

`bin/*/ObjecTimeVM.*`

These are the modified ObjectWorks/Smalltalk virtual machines for various platforms. These files are executed in conjunction with the ObjecTime5.2.otd file.

`bin/*/objectime`

This is a UNIX shell script which you use to invoke the ObjecTime toolset. It automatically selects the appropriate virtual machine for execution depending on the type of workstation it is invoked from.

`bin/create_objectime_dir`

This is a UNIX shell script which is used to create a directory which contains an ObjecTime session file. ObjecTime is always executed from this created directory.

> **Note:** In Windows NT, the function of 'bin/objectime' and 'bin/create_objectime_dir' are combined into a single file 'bin/winnt4/ObjecTime5.2.1.exe'.

`license`

This directory contains various files containing encrypted information and is used by the License Manager in order to ensure that ObjecTime is being executed according to the End-User License Agreement.

`ModelExamples`

Various examples illustrating the use of ObjecTime features are included here. Each update has instructions on its use within the Properties Editor of its Update Browser.

`C++/SimulationRTS`

This directory contains all of the source code, makefiles and other files required by the C++ Simulation Services Library.

`RPL`

This directory contains files used by the RPL browsers in ObjecTime.

`fonts/*`

This directory contains directories containing required text fonts.

`linearForm`

This directory contains yacc specifications for the ObjecTime linear form output used to store classes in a library or directory.

`tools`

This directory contains shared libraries which allow ObjecTime to integrate with source code debuggers on different target platforms.

ntsetup

This directory contains the setup programs for installing ObjecTime on Windows NT.

> **Note:** In Windows NT, the directory ‘ntsetup’ is used to perform a remote setup. By accessing this directory, which contains the file ‘setup.exe’, you can setup from a remote machine using an existing installation.

Versions

This directory contains packaging and version information..

WebModelPublisher

This directory contains executables and scripts for the Web Model Publisher option if applied to your ObejcTime installation

Codegen

This directory contains executables and scripts relating to code generation utilities.

Appendix B

Known Limitations / Restrictions

Inconsistent compile state

It is possible, after some model and environment changes, to get into a state where the model compilation is failing even though the model should compile. In such circumstances, the only solution is to remove the generated files (C++/C, makefiles and dependency files) and start again. This problem occurs because dependency (.dep) files contain the last-known set of depended files, and will only be rewritten if one of these files (or the Inclusion Paths) changes. Consequently, the Makefile requires that all of the last-known set of depended files already exist.

Known examples of where model/environment changes will cause compile problems which require cleaning up generated files are the following:

- An external header file, or path to a header file, is renamed both externally and in the toolset, after a compile has been performed. This will cause the compile to fail during the dependency calculation phase of compilation. This is because dependency files referring to the old file/path will be used to calculate the new dependencies. Since the file no longer exists or has moved, the dependency creation script, makedepend will not find the file and will fail.
- The loadbuild paths set in the update properties editor are changed to point at a different directory after a compile has been performed. The generated dependency files will still point at the original loadbuild directory resulting in the wrong files being considered during model compilation.

To recover from the above situations, some of the generated files must be removed. To remove the generated files, the makefile in the update directory can be run by using the command "**make CLEAN_ALL.NOW"** from the update directory. This will remove the LF, C++, C and build directories, their contents and subdirectories. A subsequent re-compile from the toolset will re-generate these directories and their contents.

If VPATH is being used for build reuse, then the "Generate Changes Only" should be selected on the re-compile. This will bring the toolset/environment into a state consistent with the build context and the changes made in the toolset since the update was created from the context (or the project file was merged into the toolset).

Supported Platforms

- 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. (PR 1943)
- During compilation for 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. This warned message should be disregarded by the user.

External Layer

- In situations using External Layer short circuit connections with the TargetRTS, 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)

X11

X server bugs on HP-UX 10.20 in ObjecTime Developer for C++ and C

- We have discovered a few graphic-related bugs when using ObjecTime on HP-UX 10.20 X servers. Some of the ObjecTime code has been rewritten to work around these X server bugs, but there exists at least one minor graphic anomaly around which we cannot work. This problem happens when the ObjecTime window is partially covered by another X server window. In this state, when windows and dialogs are closed on the main ObjecTime window, the background may not be redrawn properly. A user workaround is to either collapse and then expand the main ObjecTime window to cause it to redraw when this graphic corruption occurs, or just to not use windows overlapping with the main ObjecTime window.

  Note that this problem has been found to occur on various versions of the X server for the HP-UX 10.20, and certainly occurs on the most recent version of the X server at the time of the ObjecTime 5.2.1 release.The bug is known to occur on the following patch level of the X server:

  PHSS_11628 s700_800 10.20 X/Motif Runtime July97 Periodic patch

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 open during code generation.
- 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)
- The Toolset display sometimes doesn't get completely updated when dragging another window in front of an open Toolset window. Thin background colored vertical lines might be left on the Toolset window. To refresh the window, maximize it and then restore its size. (PR3226)

- If the license manager is running locally on the system, the uninstall procedure does not remove the License Manager service from the system. The license manager can be disabled by the user through the system control panel.
- When using online help each selection of a menu item will open up a new copy of the default browser. This is due to a limitation in the browser interface on the Windows NT platform.
- 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. (PR 3834)

NEW as of 5.2: The following issues are new since ObjecTime Developer 5.2:

- **Pathnames:** ObjecTime Developer 5.2 can't be installed in a directory whose pathname contains a space. It can handle include or library directory names that contain spaces if they are enclosed in double quotes. (PR3833)
- **Cross Platform Access:** In ObjecTime Developer 5.2, when using PVCS libraries for Windows NT/ Unix cross-platform development, you can access Unix PVCS libraries from the Windows NT environment but you cannot access Windows NT PVCS libraries from the Unix environment.
- **Windows NT Compilation Limits:** For information on source compilation and linking limits, see "Windows NT Compilation Command Line Limits" on page 132 of this guide.

Working Directory

Note: Do not store files or updates in the ObjecTime Working Directory. Set up another directory where you can passivate your updates.

Merging

- Daemons placed on an unconnected transition point from the inside of a state, will be lost during a merge.

SimulationRTS

- The 'size' method on subclass ports in the SimulationRTS always returns the replication factor for the base class, even if the replication factor is different for the subclass. (PR3410)
- When using Windows NT to recompile the SimulationRTS you must use the makent.bat script supplied.

Class differences merging

To perform merging of changes from different versions of a class, it is recommended that the differences tool and CVM (class versions merge), accessible from the toolset menu, be used.

Some development environments support multi-way merging of classes. This facility is used to support simultaneous check out of a class. The results are then merged together. Multi-way merging takes into account what has been added and deleted, relative to a common ancestor version of the class, and constructs a class which contains all the appropriate changes. In some cases when two changes have been made in the same area, the tool signals a conflict which must be manually resolved. ClearCase's ClearMerge is one such utility which performs multi-way merging.

Some customers have asked about the possibility of using these external merge facilities on the storage linear form of ObjecTime models. ObjecTime cannot recommend this practice because of the possible corruptions which may result. Due to the structure of the model files, they are not amenable to textual merging, and such an attempt may result in a corrupted file. Corrupted files may not be readable by the toolset or they may be readable but result in incorrect code being generated.

To perform merging of changes from different versions of a class, it is recommended that the differences tool and CVM (class versions merge), accessible from the toolset menu, be used.

User interface

Shortcut Keys

- We have discovered minor problems when using ObjecTime's shortcut keys when running ObjecTime under various window managers. Some window managers use the same shortcut keys as ObjecTime. If a window manager has the same shortcut key as is used in ObjecTime, typing the shortcut key will cause the window manager to perform the action as defined by the window manager and the shortcut key will not be delivered to ObjecTime.

  In short, if you discover ObjecTime shortcut keys that either do not work or do something unexpected, then your window manager may be intercepting these shortcut keys. This problem can usually be fixed by modifying your window manager's shortcut key map. A few known problems with shortcut keys and possible fixes are listed below.
- The default configuration for OpenWindows 3.3 has the shortcut key “Meta-W” defined to close OpenWindows windows. When using ObjecTime, this default behavior will close (collapse) the ObjecTime window. To disable this behavior and use Meta-W to close windows inside ObjecTime, you can:
  - open the OpenWindows properties windows (from the background menu)
  - select category 'Keyboard'
  - set 'Keyboard Menu Equivalents' to 'Application Only' instead of the default 'Application + Window'
- The OpenWindows Virtual Window Manager (the window manager which simulates many virtual screens on one monitor) uses the Meta-<arrow key> shortcut keys to allow switching between screens. ObjecTime also uses these shortcut keys in the RPL editor for navigating between nodes. Therefore, if using the OpenWindows virtual window manager, you will not be able to use the Meta-<arrow key> shortcut keys in the RPL editor.

Batch Mode

- When merging in batch mode in ObjecTime Developer 5.2.1, incorrect syntax in the batch script file may cause system exceptions. For example, merge from `/home/user1/tests/OpManager.actor endmerge` would cause system exceptions to occur. The correct syntax for this operation is:

  merge from `/home/user1/tests OpManager.actor endMerge`
- When using the batch mode "selectOption" action, only language option names that do not contain spaces can be specified.

Library

- SourceSafe Library Management reports that all ObjecTime class types are binary, even if they really are in text mode. (PR4174)
- Library System: When submitting classes to a library, do not cancel the submission by clicking [Cancel]. Doing so might result in a bad state, where the Toolset indicates that the class is still checked out, but the library system thinks it's not checked out. (PR3117)
- The library system does not currently prevent several users from submitting two classes of different types (for example, actor class and data class), but with the same name, to the same library. This may cause a

system error upon subsequent merging of one of these classes from the library. Hence users should ensure the uniqueness of all class names submitted to the library.

- The RCS library mechanism allows a user to check out classes more than once through multiple updates open in the same session. (PR 1124)

  If a class is checked out more than once, unchecking it out will result in only the local update browser being updated.

  Note: SCCS libraries do not have this problem.

- Auto-mounted file systems can sometimes result a problem if there is no activity for a long time. This is specifically applicable when using auto-mounted libraries. Some systems remove the auto-mounted library after a certain period of inactivity. The workaround is to leave one shell with its current working directory in the library directory, that is, open a shell and cd to the library directory. The auto-mounter considers it in use and will not remove the file system.

  Note: There is an incompatibility with the generic use of PVCS and SCCS libraries at the same time. To use PVCS and SCCS together, use the library script interface to both and make sure that your Path provides access to all PVCS tools and to the SCCS command. Place PVCS before SCCS in the Path.

Emergency Passivation

- The *kill -USR1* facility may not be able to invoke the emergency save operation in the event of the X-Server crashing.

Memory Usage

- To ensure the most optimum use of memory, we recommend that users should periodically[1] passivate their updates and then re-activate them into a fresh ObjecTime session created via the script *create_objectime_dir*. This is especially needed if you have cancelled several activations of updates.

Platforms

SGI Machines

- The license server does not seem to recognize user IDs correctly, therefore running ‘licenseInfo’ shows user IDs for all license holders as ‘unknown’.
- You must use the pcf fonts with SGI machines.

AIX Machines

- A session saved on an AIX machine can not be used on any other platform. Updates or contexts should be used to transfer models from AIX machines to any other platform.

DOORS

- The elements in the ObjecTime design update should not contain any double quotation marks. This causes an error when the exported linear form for the update is imported into a DOORS Design Formal Module.

1. Weekly, or more often if memory consumption appears excessive.

- DOORS Integration Pack 2.2 will be required for compatibility with ObjecTime Developer 5.2.1.
- DOORS Integration Pack 2.1 is required for compatibility with ObjecTime Developer 5.2.
- Starting with DOORS Integration Pack 2.1, a file in:

  Unix: $DOORSHOME/bin/OT_Version

  or on **Windows NT:** %DOORSHOME%\bin\OT_Version

  OT_Version can be viewed to identify the version of the integration pack which is installed. If this file is missing, the installed integration pack is of a previous version.

Default Parser/Scanner Generator

- In order to support building the linear form parser on multiple platforms, GNU Bison (version 1.25 and later) is now the default parser generator and flex (version 2.5.4 and later) is now the default scanner generator. Both of these tools are publicly available on a number of sites on the Internet.

Help

- If you bring up help from within the Toolset, and your configured browser isn't running, the Toolset will start the browser for you. But when you exit the Toolset, the browser is left running. (PR4185).
- ObjecTime Developer passes several parameters to Netscape when starting the online help system. Therefore, if a system administrator wishes to use a script to invoke the Netscape executable, then these parameters must be passed to the executable.
- The link from the help index page always goes to the Table of Contents and skips the front matter. To review this material go to the document and select [Top].

Simulation and Target Compatibility

- When using recall/recallAll in OTD 5.2.1 do not use default arguments. Supply both arguments to be fully compatible with the Simulation RTS and the Target RTS.

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.

Simulation Timing

There are known problems using invokes with C++ actors under simulation timing, which even an all C++ model won't fix. This has to do with the single thread that C++ simulation actors run under, sharing the same stack. Using invokes under simulation timing can often result in cases where the C++ executable stack is not unwound in the proper order. To be safe, only RPL actors should be used with invokes under simulation timing.

ObjecTime®
www.objectime.com

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