Last Reviewed: 3/98

ObjectCenter ChangeLog:

ObjectCenter Versions 2.1.0, 2.1.1, and 2.2.0

Table of Contents

This document provides details on all new features added to the ObjectCenter product since version 2.1.0. This includes all new features for versions 2.1.0, 2.1.1, and 2.2.0. It also describes the changes made between versions 2.0 and 2.1, specifics on the ObjectCenter Documentation and product limitations.

NOTE: All features added to each release of ObjectCenter are included in the releases made available later. For example, features added to ObjectCenter Version 2.1.0 are still available in ObjectCenter Version 2.1.1 and 2.2.0 unless otherwise documented.

More details about the ObjectCenter releases can be found within the *ObjectCenter Release Notes* as well as the *ObjectCenter Platform Guides* for each release. Refer to the main **ObjectCenter page** on CenterLine's website to access these documents relative to your specific release of ObjectCenter.

New Features in ObjectCenter Version 2.2.0

We have added the following new features in Version 2.2.0:

- Support for Solaris 2.5, 2.5.1, and 2.6 and SunOS 4.1.4 operating systems.
- Support for Sun UltraSPARC workstations
- HP 9000 Series 700 workstations running HP-UX 10.01, 10.10 and 10.20.
- Bug fixes
- New install and licensing procedures

All features supported in ObjectCenter Version 2.1.1 are also supported in ObjectCenter Version 2.2.0.

Like Version 2.1.1, ObjectCenter Version 2.2.0 supports the following SUN and HP Compilers (in addition to CenterLine-C and CenterLine-C++):

- **clcc** (CenterLine-C)
- **CC** (CenterLine-C++)
- Sun/SPARC-C (all versions)
- Sun/SPARC-C++ Versions 2.0.1 and 3.0.1 **ONLY**
- **gcc** v2.5.8 **ONLY**
- FORTRAN
- HP-C (all versions)
- HP C++ Compilers **cfront-based Versions ONLY**

Refer to the **Product Compatibility Matrix**, the **ObjectCenter Version 2.2.0 Release Notes** or the **ObjectCenter Version 2.2.0 Platform Guide** on CenterLine's website (**www.centerline.com**) for additional product support/compatibility information.

Licensing and Installation Enhancements in ObjectCenter 2.2.0

With the release of ObjectCenter Version 2.2.0, we have enhanced the installation process once again. Instead of running a **tar** command followed by the **RUN_ME** script, we provide a single **install.sh** script to perform these two tasks for you in one single step. To reflect these changes, we have also updated the ***Installing CenterLine Evaluations*** and ***Installaing and Managing CenterLine Products*** guides.

Additionally, we provide a later release of the FlexLM licensing software. ObjectCenter Version 2.2.0 now uses FlexLM Version 5.0a.

New Features in ObjectCenter Version 2.1.1

SUN Platform Support

This release adds support for the following:

- The Solaris 2.4 operating system
- The Sun SPARCompiler C Version 3.0.1
- Sun SPARC 5 workstations running SunOS 4.1.3_UI or Solaris 2.3 or 2.4

Support for Solaris 2 non-PIC Shared Libraries

We have added support for shared libraries that were not built for position independent loading to the Solaris 2.x platform. You can now load shared libraries with non-PIC relocations, including the Sun **XGL** and **XIL** graphics libraries.

Online Documentation

Several of the ObjectCenter manuals are now available online. To access the new online documentation, select **Manual Browser** from the **Browsers** menu on any primary window, click on the "?" button in the Main Window, or issue the **cldoc** command from a shell.

In the left panel of the **Library** window are one or more collections of books. Click on the name of a collection to display the names of all the books in that collection in the **Books** panel. You can open a specific book by double-clicking on its name, or selecting its name and clicking the **Open** button.

You can perform a simple search from the *Dyna*Text **Library** window, and you can perform more complex searches from a book window. Select **Forms** from the **Search** menu to see which other search forms are available. Use the **Next**, **Previous**, and **Go Back** buttons to navigate through the book.

Underlined text is hot - clicking on it scrolls the window to the section of the book referenced, or opens a new window if the reference is to another book.

You can create a history of your movement through the book by selecting **New Journal** from the **File** menu and selecting **Start Record** in the dialog that pops up.

Print sections by selecting **Print** from the **File** menu and highlighting the sections you want to print. Export sections to a file by selecting **Export** from the **Edit** menu, highlighting the sections you want to export, selecting **Content** as the **Export format**, providing a filename, and clicking the

Export button.

Figures, tables, and tips are shown as icons. Double-click to open them.

Tools.h++ Version

The Rogue Wave Tools.h++ class library binary distributed with ObjectCenter Version 2.1.1 and all subsequent versions, including Version 2.2.0, is Tools.h++ version 6.1.

CenterLine does not have any plans at this time to upgrade the version of the Rogue Wave Tools.h++ class library binary within later releases of ObjectCenter. Customers wishing to use a later release of the Rogue Wave Tools.h++ class library binary must obtain an upgrade through Rogue Wave directly.

Customers running ObjectCenter Version 2.2.0 (or any later release in the future), who have a **current support contract** in place for their ObjectCenter licenses, will receive full support from CenterLine should an issue arise while using a later version of the Rogue Wave Tools.h++ class library binary.

New Features in ObjectCenter Version 2.1.0

Thread Support on Solaris 2.3

We've added support for threaded applications on the Solaris 2.3 platform in process debugging mode (**pdm**), with the ability to debug threads in executables and a graphical **Thread Browser** to show the status of all the threads in your program. We have also added a thread-safe **libC** (the C++ library).

In process debugging mode, the **Thread Browser** gives you information about the threads and lightweight processes in your program. This information includes a list of all threads, and the state of each thread. The state information includes the function the thread is executing, the execution state (for example, **running**, **sleeping**) of the thread, and the start function for the thread.

At any given time, the **Thread Browser** focuses on a single thread or light-weight process (**LWP**), known as the "current active entity." You control execution of the current thread with the **cont**, **next**, **nexti**, **step**, and **stepi** commands. You can display a traceback of the thread execution stack with the **where** command. You can also perform these operations on another thread at the break level by making it the current active entity. To make another thread the current active thread, you use the **thread** command with the new thread number as an argument.

ObjectCenter's **pdm** supports threads and threaded applications, where the types of threads that can be used are **Solaris-based** thread types. **POSIX** and other thread types are not supported by ObjectCenter. Solaris is the only operating system where threads are supported. ObjectCenter for HP-UX or SunOS do not support threads.

For more information about debugging threaded applications, see the **thread**, **threads**, and **thread support** entries in the **Manual Browser**.

Emacs Main Window

If you're an **FSF GNU Emacs 19** user, you can start up an ObjectCenter session from within Emacs.

First you need to add the following lines to your **.emacs** file:

```
(setq load-path (cons ".../lib/lisp" load-path))
(load "clipc")
```

The **M-x objectcenter** command starts the environment as a subprocess of Emacs, with the menus from the **CenterLine Main Window** replacing the menus at the top of your **Emacs window**. Edit the path name supplied if you want to run a different version of ObjectCenter. You can give a project name as an argument. To invoke ObjectCenter in process debugging mode, use the **-pdm** switch. Edit your code directly in the **Source area** at the top of the **Emacs Main Window**.

All the browsers are available with the same commands and menu items you use when you invoke ObjectCenter from a shell. A separate **Button Panel** window, including your own user-defined buttons, can be launched from the **Browsers** menu in the **Main Window**.

For more information, see the **emacs integration** entry in the *ObjectCenter Reference Guide*.

Printing from the Cross-Reference and Inheritance Browsers

You can now print the contents of the **Cross-Reference** and **Inheritance** Browsers to a postscript file. When you click on the **Print...** button in the **Browser**, the **Print from Browser** dialog box is displayed. You can specify what paper size to use, the title of the printout, and the name and location of the output file. The default location is your current directory. You can also specify how many pages the output should be printed on. For example if you specify an output page height of **2** and width of **3**, ObjectCenter resizes the output to fit on six sheets of paper.

Frequently Asked Questions in the Manual Browser

We've taken the questions that customers asked most often and put the answers in the **Manual Browser**. Click on the "?" button in the **Main Window** to open the **Manual Browser**.

Select ***ObjectCenter User's Guide*** from the list of books, and then select ***Appendix A, "Frequently Asked Questions"***, from the **Table of Contents** panel. Click on the "+" icons in the **Table of Contents** panel to view the topics covered. You'll find topics such as *"How do I fix unresolved symbols?"* and *"I'm having trouble loading with make"*.

Workspace Improvements

We have made several changes to enhance performance and behavior in the **Workspace**. The most important changes are described here. In addition to the changes described below, you can now do the following in the **Workspace**:

- Set actions on variables defined in the **Workspace**
- Define **constructors** of nested classes
- Define **macros** to have the value **0**
- Declare a single-declaration **extern** function without enclosing the single declaration in braces
- Define a **main** with a single line and set breakpoints on it.

New Command: `load_header`

We have added a new command, **load_header**, for loading the definitions from header files into ObjectCenter. Use the **load_header** command to load non-local header files without specifying a path, and to load multiple header files into a single module. If you want to load a header file in your working directory or path, you can use use the **load** command to load it.

The **load_header** command replaces the **#include <header_file.h>** syntax. There were several problems related to the use of **#include** in the **Workspace**. For example, because definitions were parsed one at a time, if an error was encountered while parsing a header file, previous definitions were not undefined. As a result, users often had to issue an **unload** workspace command before they could reload a header file.

In addition, the **Workspace** does not match the separate compilation model of C and C++. To enable the **Workspace** to function as a debugger, definitions in a header file included in the **Workspace** are visible across modules. As a result, using **#include** in the **Workspace** occasionally causes ObjectCenter to pick up incorrect definitions from included files.

The new **load_header** command lets you load a header file into the environment in a separate module, using the paths defined in the **load_flags** and **sys_load_cflags** and **sys_load_cxxflags** options to locate the file. You can use **load_header** to load system header files such as **iostream.h** without specifying the path to the file.

You can also load multiple header files into a single module with **load_header**. For example, if the local header file **rect.h** has dependencies on definitions in **math.h** and **limits.h**, you can load all three header files into a single module with this command:

```
-> load_header <math.h> <limits.h> "rect.h"
```

For complete syntax and usage information, please refer to the **load_header** entry in the **Manual Browser**.

We have provided a new option, **workspace_include**, to provide backwards compatibility for users who have existing project files that use the **#include** syntax. We do not recommend that you set this option for new projects. The option is described in the **load_header** and **options** entries in the **Manual Browser**.

`-dd=off` Default for Header Files

ObjectCenter now loads all header files (files with the suffix **.h** or **.H**) wit hteh switch **-dd=off** unless you specify **-dd=on** on the **load** or **load_header** command line or in the **load_flags** option.

In general, when you load header files, you will want to load them without demand-driven code generation so that the class data and member functions are available. Suppose you've loaded a source file, **List.C**, with **-dd=on**, the **List.C** includes **List.h**. Because you loaded the **List.C** with demand-driven code generation turned on, only the classes and member functions defined in **List.h** that are actually used in **List.C** will be visible to the **Class Examiner** and otehr browsers. To see all the classes and member functions defined in **List.h**, load the header file:

```
-> load List.h
```

ObjectCenter loads the header file with demand-driven code generation turned off by default.

`edit workspace` Enabled

You can now save code you define in the **Workspace** more easily. During a ObjectCenter session, all C++ definitions you enter are stored in a **Workspace** scratchpad. The **edit workspace** command lets you save the scratchpad to a file, by default **workspace.C**, and then edit the file.

For example, suppose you create a class in the **Workspace**. You can create stubs for other classes and external functions called by the class, and then invoke the methods in the class to test them.

After testing your class, you can use **edit workspace** to create a file containing the code you defined in the **Workspace**. You can enter your own name for the file or accept the default, **workspace.C**.

```
-> edit workspace

Appending all workspace definitions to a file.
Default filename is "workspace.C" in the current directory.
Please specify a filename, press Return to accept default,
or <CTRL-D> to abort:
```

If you want to test a particular set of definitions, edit the file so that it contains the definitions you want to test. Then use the **unload workspace** command to unload all the definitions and objects you created in the **Workspace**, and use the **source** command to load the definitions in your saved file back into the **Workspace**. Note that the **source** command will report errors if you've unloaded any definitions that the saved file depends on.

If you want to use the new file as source code, add any **#include** lines you need and remove any extraneous lines. For example, some ObjectCenter commands, such as **whatis**, will appear in the file.

> **NOTE:** When using code developed in the **Workspace**, remember that **static** functions and variables and private and protected member functions are visible at global scope in the **Workspace**. As a result, you may have to add **friend** functions or classes or make **static** functions externally visible to use the **Workspace** sources as a separate file.

`unload workspace` Improved

The **unload workspace** command now only undefines user-defined variables. Previously, when you used the **unload workspace** command, all variables were undefined, including predefined macros such as **__CENTERLINE__**. ObjectCenter now resets predefined macros after an **unload workspace**.

Creating Objects for Workspace Classes with Virtual Functions

You can now create objects of **Workspace** classes which contain virtual functions or which are derived from classes that contain virtual functions.

Previously, if you defined a class with a virtual function in the **Workspace**, and then tried to create an object of the type of that class, ObjectCenter reported unresolved references.

Overloading and Name-Resolution

ObjectCenter is now able to find multiple functions of the same name in multiple compilation units.

Previously, if you loaded a file that contained an overloaded function, and then defined another function with the same name in the **Workspace**, ObjectCenter was unable to find the functions defined in the source file. In rare circumstances, ObjectCetner may still generate spurious errors because it cannot do true multiple-module overloading. In these cases, we recommend you use **casts** to unambiguously establish the type of the arguments to the function.

HP `clcc` Compiler

CenterLine's C compiler (**clcc**) is now available on the HP platform. The CenterLine-C compiler is an ANSI C optimizing compiler designed to achieve small code size. The compiler is also compliant with K&R C and is link compatible with Sun and HP compilers.

Enhancing pdm Debugging

In addition to support for debugging threaded applications, we've made some other improvements to pdm. The process debugging mode in ObjectCenter 2.1.0 is based on Version 4.12 of **gdb** and takes advantage of its new features.

Error Browser Layout Improvements

We've streamlined both the appearance and the performance of the **Error Browser**. We've moved some of the buttons from the bottom of the **Browser** to the side and added **Edit**, **Reload**, and **Build** buttons to the side button panel to make it easier to fix and reload your code. We removed the redundant buttons used to suppress warning messages, so now you use just the **Suppress** menu to control suppressions. The new **Error Browser** uses less memory and performs faster.

Changing the Size of Your Windows

We've added Motif-style resizeable panes to the **Main Window** and **Thread Browser**. For example, to change the relative size of the **Source** panel and **Workspace**, place your cursor on the pane control sash, which is a square box at the bottom of the **Source** panel pane, just under the **Error Sentinel**. Hold down the **Left** mouse button and drag the sash up or down to resize the panes.

X Resource Changes

The Graphical User Interface in this release of ObjectCenter was built with a new release of the **OI** toolkit. This change made it necessary to set more specific font and color resources for some elements of the GUI. You can override some of these settings by using the switches described in **Table 20** in the online *ObjectCenter Reference Guide* (online) or by changing your X resources as described in ***"Modifying X Resources"*** in the *ObjectCenter Reference*.

ANSI C Code Generation in ObjectCenter

ObjectCenter is based on the USL C++ Language System. The USL C++ Language System uses **cfront** to translate C++ source code into C intermediate code before compilation. Because ObjectCenter is based on the USL C++ Language System, ObjectCenter generates C intermediate code internally in order to interpret C++ source code in the ObjectCenter environment.

Previously, the type of C intermediate code ObjectCenter generated was limited to K&R C. We have now enabled ObjectCenter to generate either K&R or ANSI C intermediate code. The default type of C intermediate code generation is K&R C.

Usually, generating K&R C intermediate code causes no problems.

Problems can occur, however, in the ObjectCenter environment with some programs having object code produced by a compiler that generated ANSI C intermediate code if the programs have functions that pass **float**, **short**, or **char** arguments.

If you get incorrect results because ObjectCenter generates K&R C intermediate code by default, you can select ANSI C intermediate code generation.

You can choose ANSI C intermediate code when invoking ObjectCenter by using a new command-line switch (**-backend_ansi**). You can also toggle between the two types of code generation during a session by setting or unsetting the new **backend_ansi** option (**setopt backend_ansi** or **unsetopt backend_ansi**).

We have also provided an ANSI C **libC** (the C++ library) for CenterLine's C++ compilation system. The K&R C **libC** is the default. To use the ANSI C **libC**, use the **+a1** switch. The libraries are installed in **CenterLine/<arch>/lib/a0/libC.a** and **CenterLine/<arch>/lib/a1/libC.a**. See the **CC** entry in the **Manual Browser** for information about the **+a0** and **+a1** switches.

For more information about K&R C and ANSI C intermediate code generation, see the **Manual Browser** entry **"code generation"**.

HP Softbench 3.2 Support

The ObjectCenter integration with **SoftBench** supports **Softbench** messages through CenterLine's Application Programming Interface, or API, using a Gateway to translate the messages. With this release we've added a new switch to the **objectcenter** and **codecenter** commands that starts up the Gateway automatically on the HP platform.

To use your CenterLine environment and Softbench together, first start up Softbench. Then invoke ObjectCenter with the **-softbench** switch:

```
% objectcenter -softbench
```

This command sends a **START** message to the **SoftBench Tool Manager**, which places ObjectCenter in the **Tool Manager's window**.

The CenterLine environment and the **SoftBench Tool Manager** send messages to each other through the Gateway. For example, ObjectCenter sends an **EDIT-WINDOW** message to the **Broadcast Message Server** when it needs to bring up an editor. The **SoftBench** editor loads and displays the file.

> **NOTE:** Pretty much all versions of Softbench compiler are supported by ObjectCenter. Any issues using a specific release beyond version 3.2, please contact CenterLine Technical

Support at (781) 444-8000 or via email at **support@centerline.com**.

To terminate the Gateway and remove ObjectCenter from the **Tool Manager window**, use the ObjectCenter **quit** command.

For more information about CenterLine's API or Gateway, enter **man CLIPC** or **man CenterLine API** in the **Workspace**, or **man clms_gateway** in a shell.

Tools.h++ Class Library Object Code

We provide a binary version of the **Tools.h++** class library from Rogue Wave with this release of ObjectCenter. The **Tools.h++** class library is a C++ foundation class library that includes support for single, multibyte, and wide character strings, time and date handling, internationalization, and persistant storage.

Among the classes in the **Tools.h++** library are template-based classes including **hash tables**, **stacks**, and **dictionaries**, the **RWFile** class to encapsulate standard file operations, and **generic** and **Smalltalk-like** collection classes. The **Tools.h++** class library is installed in the directory **CenterLine/rwtool**.

Rogue Wave **Tools.h++** classes are unsuitable for examination with the ObjectCenter **Data Browser** because of their sophistication. To enable you to examine their contents, we have provided a Rogue Wave Tools.h++ integration that dumps a readable representation of the contents of **RWCollectable** classes into the **Run Window**.

The integration is extensible to your own classes and subclasses and works in both component debugging mode (**cdm** or the ObjectCenter Interpreter) and process debugging mode (**pdm**). Preparation, usage and capabilities are slightly different in the two modes. The integration is installed in the directory **CenterLine/unsupported/src/rwtool**.

Each ObjectCenter license entitles one developer to use the **Tools.h++** class library. If you have ObjectCenter floating licenses, and more developers wish to use **Tools.h++** at your site, than the number of licenses you have purchased, please contact your CenterLine Sales Representative at (781) 444-8000 or via email at **info@centerline.com** to register additional users.

Licensing and Installation Enhancements

To make the installation process easier, we've made some changes to the **RUN_ME** script and made the licensing error messages more informative. We've also updated our installation manuals, ***Installing CenterLine Evaluations*** and ***Installing and Managing CenterLine Products***.

Changes

This section describes improvements made in ObjectCenter in the point

Between Version 2.0.0 and Version 2.1.0

releases since Version 2.0.0.

Memory Leak Detection

Memory leak detection identifies potential memory leaks by reporting on the memory that the program allocates while running and fails to free before exiting.

The memory leak detection report lists leaks by the size of the memory allocated and identifies where the program allocated the memory in the stack trace. In addition, it shows the number of times the leak occurred there. For more information, see the **memory leak detection** entry in the **General Topics** category.

Run Window Changes

By default, the input and output of your program previously went to the terminal in which you invoked ObjectCenter without spawning an independent **Run Window** and without returning control to the shell.

Now, ObjectCenter by default opens a separate **Run Window** for your program's input and output and returns control to the shell in which you invoked ObjectCenter. A CenterLine program called **clxterm** creates this **Run Window**, which is a standard version of **xterm**, the **X11** terminal emulator.

To avoid creating the separate **Run Window** and avoid returning control to the shell, use the **-no_run_window** switch when you invoke ObjectCenter. The program's input and output goes to the shell in which you invoked ObjectCenter. Using the **-no_run_window** switch means you are unable to interrupt ObjectCenter and unable to place it in the background. This option is intended for debugging applications that need specific terminal support rather than a generic terminal such as **xterm**.

> **NOTE:** Avoid starting ObjectCenter in the background using the **-no_run_window** switch. Your program could have undesirable input/output behavior.

To create a separate **Run Window** and avoid returning immediate control to the shell, use the **-no_fork** switch. With **-no_fork**, control returns when you enter the suspend character (usually **^Z**) in the shell or exit ObjectCenter. After you type the suspend character in the shell, you must type **bg** to enable your program to direct output again to the **Run Window**. Without **-no_fork**, the shell prompt comes back immediately.

By default, issuing the **run** or **start** command deiconifies the **Run Window**. To prevent deiconifying the **Run Window**, use the **win_no_raise** option. Setting this option prevents the deiconification of the **Run Window** when you issue **run** or **start**.

The **close-window** (**f.delete**) **window-manager** operation now iconifies the **Run Window**. This is to prevent the accidental destruction of the **Run Window**.

We added unique resource names for the **Run Window**, generic terminals, and the **vi** Edit window.

- The **Run Window** resource is:

  ObjectCenter*RunWindow.xterm-resource

- The generic-terminal resource is:

  ObjectCenter*Terminal.xterm-resource

- The **vi** Edit window is:

  ObjectCenter*EditWindow.xterm-resource

Scrollbars are now the default in the **Run Window** and in the generic terminal windows. Disabled scrollbars continue to be the default for the **vi** Edit window.

For more information about setting resources for these windows, see the ***"Run and Edit Windows"*** entry in the **Manual Browser**.

`clxterm` Resources in `app-defaults`

ObjectCenter used to require you to use **.Xdefaults** for setting **clxterm** resources. This is no longer true. ObjectCenter's **clxterm** now reads the **app-defaults** files of the Graphical User Interface as well as **.Xdefaults**.

This means you can put all GUI-related resources (those for the user interface itself as well as for **clxterm**) into the same **app-defaults** file. You no longer have to split them up into two different files, nor do you have to use your **.Xdefaults** file for program-specific resources.

The app-defaults file for ObjectCenter is:

$XAPPLRESDIR/ObjectCenter

See the **Xresources** entry in the **Manual Browser** for more information.

More Flexible Resource Setting

The ObjectCenter Graphical User Interface used to require that you use a model-specific resource setting to set the colors of some objects. This is no longer true. You can set colors of objects in the same way in **Motif** and in **OPEN LOOK**. This is the syntax for the Graphical User Interface:

ObjectCenter*Color*OI_scroll_text.@text.Background: color

This is the syntax for the **Workspace**:

ObjectCenter*Color*Workspace.@text.Background: color

The *ObjectCenter Reference* (in the section called **"Setting resources for scrolling text objects"** on p. 409) states that you must use a model-specific resource setting to set the colors of objects. Although this is no longer a requirement, you can choose to follow the instructions in that section if you want a certain standard set of localizations for **Motif** users, and a different set for **OPEN LOOK** users.

New Resource For Control Buttons

We have added a new resource that can improve performance by reducing the **X11** server traffic that results from dimming the control buttons in the GUI. This resource is especially valuable when running ObjectCenter with slow X servers or low-speed connections such as X over serial lines.

In previous releases of ObjectCenter, the control buttons dimmed as soon as the component debugger became busy. The new resource enables you to specify the length of time that the debugger must be busy before the control buttons on the GUI dim. By default, the buttons on the GUI dim when the debugger has been busy for 1.15 seconds:

ObjectCenter*dimButtonsWhenDebuggerBusy: 1.15

You can change the value of this resource in the site-wide application defaults file for ObjectCenter, or in your local **.Xdefaults** file. The value can be:

- The string **Always** if you want the buttons to dim as soon as the debugger is busy.
- The string **Never** if you never want the buttons to dim.
- Any positive floating-point number, to indicate the number of seconds you want to elapse before the buttons start dimming.

Support of Direct Pasting into Emacs

Previously, the Graphical User Interface of ObjectCenter did not handle **CUT_BUFFER0**, the text transfer mechanism that **GNU Emacs Versions 18 and 19** use to paste text from other applications. If you wanted to copy text from ObjectCenter and paste it into an **emacs** buffer, you had to run the **xcutsel** program to act as an intermediary.

With this version of ObjectCenter, running **xcutsel** is unnecessary. The user interface now automatically exports text to **CUT_BUFFER0** whenever you highlight text in labels, entry fields, and multi-line text objects. You can paste this text directly into emacs in the same way you paste text from an **xterm** into **emacs**.

Emacs-like Keybindings

These shell-like and Emacs-like keybindings are now available in ObjectCenter Version 2.1 by default:

Control-a	**Beginning of line**
Control-e	**End of line**
Control-b	**Backward character**
Control-f	**Forward character**
Meta-b	**Backward word**
Meta-f	**Forward word**
Control-n	**Next line**
Control-p	**Previous line**
Control-d	**Delete next character**
Control-u	**Delete to beginning of line**
Control-k	**Delete to end of line**
Control-w	**Delete previous word**

In **Motif**, some windows may use **Meta + B** and **Meta + F** as menu mnemonics, rendering them unavailable in text objects.

As a result of this change, the information in the X resources entry in the *ObjectCenter Reference* about setting translations for underlying objects in your **.Xdefaults** file is now obsolete.

User-Defined Commands Access Line Numbers

A number of customers said they missed the **L** (current **Source** area line number) facility that ObjectCenter used to provide in user-defined commands. Without **L**, writing certain kinds of user-defined commands was impossible. For example, you were unable to write a command to select a line in the **Source** area and set a breakpoint or action on that line.

Although ObjectCenter Version 2.1 allows no editing in the **Source** area and the concept of current **Source** area line number does not really apply, we have provided an equivalent facility that works in terms of the **Source** area text selection.

These four new keywords are now available in user-defined commands:

$first_selected_line
$first_selected_char
$last_selected_line
$last_selected_char

The **$first_selected_line** and **$last_selected_line** keywords provide you with the starting and ending line numbers of the **Source** area's current text selection. Lines are numbered beginning with **1**. If no text is selected in the **Source** area, both of these keywords return **0**.

The **$first_selected_char** keyword provides the position of the first character selected on **$first_selected_line**. The **$last_selected_char** keyword provides the position of the last character selected on **$last_selected_line**. Character positions are numbered beginning with **1**, and tabs are considered to be a single character. If no text is selected in the **Source** area, both of these keywords return **0**.

Directing Output to the Workspace

You can direct your output to the **Workspace** by unsetting the **win_io** option.

We recommend that you keep the **win_io** option set, however, for complicated programs that use curses-style input and output. Unsetting **win_io** has the following limitations:

- Controlling-**tty** semantics are unavailable in the **Workspace**.

  This means that **tcgetpgrp** / **tcsetpgrp** and **tty**-generated signals will not work as expected.

- If your program affects the **tty** mode, it may affect the **Workspace** output.

- The **tty** mode may not be preserved across **Workspace** interactions. For example, when you continue from a breakpoint, the **tty** settings may not be the same as when you stopped.

To unset the **win_io** option, enter this in the **Workspace**:

```
unsetopt win_io
```

Your output will go to the **Workspace** at your next **reinit**, whether it is an implicit **reinit** (for example, when you issue the **run** command) or an explicit **reinit** (by issuing the **reinit** command).

Template File Definition Extensions

By default, ObjectCenter used to require files defining templates to have an extension of either **.c** or **.C**. If you wanted to use an extension other than **.c** or **.C**, you ahd to change map files. The reason ObjectCenter had these default extensions was that they were a USL C++ Language System convention. Release 3.0.2 of the USL C++ Language System no longer has this convention. Accordingly, ObjectCenter no longer requires by default that files defining templates have these extensions.

New CC Switches

We introduced the following new switches to ObjectCenter's C++ compiler command (**CC**). For a description of these switches, and the complete list of switches to ObjectCenter's C++ compiler command, please refer to the **CC manual page**.

- **-dryrun**
- **-ec string**
- **-el string**
- **-gdem**
- **-ispace**
- **-ispeed**
- **-nCenterLine**
- **-ptdpathname**
- **-pt1**
- **-pti**
- **-ptk**
- **-ptmpathname**
- **-ptopathname**
- **-v**

New Feature for -pg Switch

Used for profiling with **gprof**, the **-pg** switch of ObjectCenter's C++ compiler command (**CC**) now automatically passes the **-pg** switch to the backend C compiler and the **-Bstatic** switch to the linker to request that it link with a static library.

New Environment Variable

A new environment variable to ObjectCenter's C++ compiler, **CENTERLINE_CC_VERBOSE**, causes **CC** to display messages to aid users in setting the **ccC** environment variable correctly.

If you use the default backend C compiler (**clcc**), when you use either the **-ansi** or **-pg** switch, the appropriate switches are supplied to the C compiler automatically. If you set the environment variable **ccC** to use a different C compiler, **CENTERLINE_CC_VERBOSE** warns you that you may need to supply an additional switch when you use the **-ansi** or **-pg** switch. Set **CENTERLINE_CC_VERBOSE** to **0** to disable the warnings.

Two New pdm Options and Switches

We have added two new options and switches in process debugging mode (**pdm**). The new options are **class_as_struct** and **full_symbols**.

The option **class_as_struct** has these characteristics:

Type: Boolean
Default Value: FALSE
Commands Affected: debug

When **class_as_struct** is false, ObjectCenter reads the class debugging information produced by the compiler. This enables ObjectCenter to provide full class information when needed.

When **class_as_struct** is true, ObjectCenter ignores class debugging information produced by the compiler. This causes ObjectCenter to treat classes as C structures. For example, the **whatis** command will only display data members and not member functions. Setting this option will give shorter initialization time but less debugging information for classes.

You can set the value of **class_as_struct** by adding this line to your **.pdminit** file:

```
setopt class_as_struct
```

You can also set it by issuing this command in the **Workspace** before issuing the **debug** command:

```
-> setopt class_as_struct
```

Use the **unsetopt** command to reset the **class_as_struct** option to false.

The option **full_symbols** has these characteristics:

Type: Boolean
Default Value: FALSE
Commands Affected: debug

When **full_symbols** is false, ObjectCenter reads only part of the debugging information to shorten initialization time. Additional debugging information will be read as needed, such as when you issue the **whatis** or **list** command. When **full_symbols** is true, ObjectCenter reads all the debugging information at initialization.

You can set the value of **full_symbols** by adding this line to your **.pdminit** file:

```
setopt full_symbols
```

You can also set it by issuing this command in the **Workspace** before issuing the **debug** command:

```
-> setopt full_symbols
```

Use the **unsetopt** command to reset the **full_symbols** option to false.

For more information about options, including setting, unsetting, and displaying them in the **Workspace**, see the **options** entry in the **Manual Browser** or the *ObjectCenter Reference*.

The new switch **-class_as_struct** corresponds to the **class_as_struct** option. The new switch **-full_symbols** corresponds to the option **full_symbols**. For example, with the following command, you can invoke ObjectCenter in **pdm** mode with **full_symbols** set:

```
% objectcenter -pdm -full_symbols
```

New Switches For `contents` and `link` Commands

We have added a switch called **-ascii** to the **contents** command and a switch called **-list** to the **link** command.

The **contents -ascii** command displays the output of the **contents** command in the **Workspace** instead of invoking the **Project Browser**. The **link -list** displays the library link order in the **Workspace**. This switch is useful for diagnosing link-order related problems in the interpreter.

New Scope for `unsuppress` Command

We have added **everywhere** as a new scope argument for the **unsuppress** command.

You use **everywhere** in combination with a ObjectCenter violation number (error or warning) to unsuppress a ObjectCenter violation while you are debugging.

The scope argument **everywhere** unsuppresses a violation wherever you had suppressed it without a location-specific scope argument. (A location-specific argument specifies a line number, file, directory, function, library, or identifier).

If you had used a **location-specific** scope to suppress the violation to begin with, the violation stays suppressed at that location if you use **unsuppress** with **everywhere**. To unsuppress the violation at that location, you must use the **unsuppress** command with the **location-specific** scope you had used to suppress it.

This is the syntax for global unsuppression of a violation with the number **num**:

```
unsuppress num everywhere
```

Alternatively, you can unsuppress all occurrences of a violation with one command regardless of whether you suppressed them with a **location-specific** argument. This is the syntax:

```
unsuppress num
```

For more information about suppressing and unsuppressing violations, see the **suppress** and **unsuppress** entries in the **Manual Browser** or *ObjectCenter Reference*.

Mnemonics

Each ObjectCenter primary window provides **Motif**-style mnemonics for almost every menu item on its menu bar.

Mnemonics are not available for items that you can create and destroy on the fly during a session, such as items on the **User Defined** submenu of the **ObjectCenter** menu in the **Main Window**.

A menu item with a mnemonic has one of its letters underlined, usually the first one. To select an item with mnemonics,

1. Press the **Meta** key and the underlined-letter key at the same time, to display the menu.
2. Press the underlined-letter key of the menu item.

New Warning Message for `free (0)`

ObjectCenter has added the following new warning message for **free(0)**:

Warning #106 Freeing NULL pointer

ObjectCenter used to give this warning message for **free(0)**:

#95 (Cannot free memory address (not within data space))

The reason for the new warning is that **POSIX** and **SVID** allow **free(0)**. With the separate warning message, you can suppress the warning separately and, thus, run **POSIX / SVID** compliant code without losing the ability to check bad calls to **free()**.

Here is an example of the new warning:

```
"d410fix.c":432, d4_10_3(), Freeing NULL pointer
(Warning #106)
431: #if ANSI && D410B
432: free(NULL);
433: #endif /* ANSI */
```

Use of **free(0)** is not portable.

Options Browser has Longer Editable Fields

In the **Options Browser**, the maximum input lengths of text entry fields used to be 512 characters. We have increased this maximum to 1000 characters.

If you need an editable field longer than 1000, you must use the equivalent **Workspace** command rather than the Graphical User Interface to perform the task. For example, if you want to set an option's value to more than 1000 characters, you must use **setopt** in the **Workspace**. The **Options Browser**, however, displays only the first 1000 characters of the value you set.

Longer Line Limit in the Source Area

The longest line the **Source** area could display used to be 500 characters. Now, the **Source** area can display up to 10,000 characters per line.

Variable Prevents Unexpected Behavior

Rapid signal delivery occurs when you use **setitimer(2)** to set an interval timer to expire after less than about 300 milliseconds or when a separate process sends signals to the ObjectCenter process via **kill(2)** at intervals of less than about 300 milliseconds.

We added **CENTERLINE_RAPID_SIGNALS**, a new Boolean environment variable, to ensure robust behavior in the presence of rapid signal delivery. In the presence of rapid signal delivery, ObjectCenter can experience unexpected behavior, such as reporting that the user program generated a **SIGSEGV** or **SIGBUS** at some unknown location.

If you strongly suspect that rapid signal delivery is causing unexpected behavior, set **CENTERLINE_RAPID_SIGNALS** to **TRUE** before invoking ObjectCenter.

Do not set this variable unless you suspect a problem with rapid signal delivery; doing so would cause a serious performance problem.

ObjectCenter Documentation

This section describes the documentation for ObjectCenter. ObjectCenter Version 2.1.1 comes with hardcopy documentation and online documentation. ObjectCenter Version 2.2.0 comes with online documentation only. Hardcopy manuals for ObjectCenter v2.2.0 are available for a separate charge. In addition, we provide USL C++ Language System documentation and Rogue Wave Tools.h++ documentation with any release of ObjectCenter.

Online Documentation

Access online versions of most ObjectCenter manuals by selecting **Manual Browser** from the **Browsers** menu or by typing **cldoc** in a shell. The following manuals are available online:

- *ObjectCenter User's Guide*
  A task-based description of ObjectCenter, explaining how to use the graphical user interface to load, manage, run, and debug programs within ObjectCenter. An appendix to the online *User's Guide* contains *Frequently Asked Questions : Answers to some of the questions most often asked of CenterLine Technical Support*.

- *ObjectCenter Reference*
  A complete reference for ObjectCenter, containing an alphabetical listing and description of ObjectCenter commands, functions, and informational topics. Appendices to the online *ObjectCenter Reference* include *About The ObjectCenter Release* (this guide, formerly called *About This Release*) and *ObjectCenter Platform Guides* containing platform-specific information about ObjectCenter. The *ObjectCenter Platform Guides* are also available in HTML form

on the main **ObjectCenter page** of our website.

- *ObjectCenter Tutorial*
  A step-by-step introduction to ObjectCenter features.

- *CenterLine-C Programmer's Guide*
  Information about the CenterLine-C compiler. Information related to the CenterLine-C++ Compilation System (**CC**) can be found in the *ObjectCenter Reference*.

In addition to the online documentation described above, the following information is available:

- Access context-sensitive help in the GUI version of ObjectCenter by moving the cursor over the item you want information about and pressing **F1** or the **Help** key if your keyboard has one, or by selecting **"On Context"** from the **Help** menu and moving the ? cursor over the item and clicking. A **Help** window appears describing that item.

  If you have bound the **F1** key to a window manager operation, you are unable to access context-sensitive help with the **F1** key.

- Access information on a variety of topics from the **Help** menu, which appears on every primary window.

- Access information about a command by typing help in the **Workspace** followed by the name of the command.

- Access any entry in the *ObjectCenter Reference* by typing **man** and the name of the entry in the **Workspace**.

Where there are differences between *ObjectCenter Reference* and the **Manual Browser**, the information in the **Manual Browser** is more up-to-date.

The online documentation available outside the ObjectCenter environment is in this directory:

> **path/oc_2.0.0/<arch>/docs**
>
> The word **path** represents the path to the CenterLine directory, and **<arch>** is a platform-specific directory, for example **sparc-sunos4**, **sparc-solaris2**, **pa-hpux8**, **i486-svr4**, **powerpc-aix** or **m88k-svr4**.

The **online** directory contains a file called **README**, which describes the files in this directory. Among the files are:

- **bugs.open**, which describes the known bugs, limitations, and workarounds for ObjectCenter.

- **bugs.fixed**, which describes bugs fixed since the most recent version of ObjectCenter

  NOTE: Some of the above listed files/directories may not be available for your version of ObjectCenter. Additionally, the **README** file in the **online** directory may not be available for your version of ObjectCenter.

Hardcopy Documentation

This is the hardcopy documentation that comes with ObjectCenter:

- *ObjectCenter Read Me First Release Bulletin*
  The latest hardcopy information, containing any updates necessary to other hardcopy documentation.

- *Installing and Managing CenterLine Products*
  How to install ObjectCenter and administer it, including how to troubleshoot licensing problems.

- *ObjectCenter Tutorial*
  A step-by-step introduction/overview to ObjectCenter features.

- *CenterLine-C Programmer's Guide*
  Information about the CenterLine-C compiler. Information about the CenterLine-C++ Compilation System (**CC**) can be found within the *ObjectCenter Reference*.

- *Tools.h++ - Introduction and Reference Manual*
  Information about the Tools.h++ foundation class library from Rogue Wave.

USL Documentation

ObjectCenter also comes with the following C++ Language System documentation:

- *AT&T C++ Language System Product Reference Manual*
  We ship this complete manual with ObjectCenter. This manual provides a complete definition of the C++ language supported by Release 3.0 of the C++ Language System.

- *AT&T C++ Language System Library Manual*
  We ship this complete manual with ObjectCenter. This manual describes class libraries shipped with Release 3.0: the **iostream** library, **complex** library, and **task** library. (We do not, however, support or supply the **task** library **libtask.a** itself.)

NOTE: If you purchase ObjectCenter Version 2.2.0, you will only receive with that purchase the above described documentation in online form. However, you will automatically receive hardcopies of the *Tools.h++ - Introduction and Reference Manual*, the *AT&T C++ Product Reference Manual* and the *AT&T C++ Library Manual* since these are not available within the online documentation. If you wish to have a hardcopy version of the manuals provided within the online documentation, they can be purchased separately.

Product Limitations

The following is a list of known current limitations with our product.

GUI Behavior

ObjectCenter currently does not allow you to change the placement of the scrollbar. In addition, you cannot set the scrollbar placement with a window manager X resource (such as **Motif**'s **XmNscrollLeftSide** or **XmNscrollRightSide** or **OPEN LOOK**'s **OpenWindows.ScrollbarPlacement**). This is because ObjectCenter is developed using the **Object Interface** (**OI**) toolkit, not the **Motif** or **OPEN LOOK** (**XView**) toolkits. The **OI** toolkit currently does not offer a resource to change scrollbar placement.

Caution if Editing `sys_load_cflags` or `sys_load_cxxflags`

ObjectCenter supplies custom versions of header files, such as **stdarg.h** and **varargs.h**. The ObjectCenter versions of the **stdarg.h** and **varargs.h** header files reside in the **CenterLine/include** directory.

The global **ocenterinit** file automatically supplies the following directories:

- To the **sys_load_cflags** option, it supplies:

  -Idirectories/include/ObjectCenter

- To the **sys_load_cxxflags** option, it supplies:

  -Idirectories/clc++/<arch-os>/incl

When **ocenterinit** supplies these directories, ObjectCenter includes the header files before the standard versions in **/usr/include**.

If you edit **sys_load_cflags**, be sure to keep the location of the **-Idirs/include/ObjectCenter** directory before other **-I** switches so this support for variable argument functions remains unchanged. If you edit **sys_load_cxxflags**, be sure to keep the location of the **-Idirectories/clc++/<arch-os>/incl** before other **-I** switches files.

ANSI Mode

ObjectCenter in **ansi** mode does not interpret the ANSI C **#define** correctly.

Interprets `#define` Incorrectly

In the following example, ObjectCenter produces the warning **Macro 'A' requires 1 parameters, but only 0 are given**, when the code should produce the output **HELLO**:

```
#define HELLO
#define A(x) #x
main() {
char *s = A(HELLO);
printf("%s\n", s);
}
```

This is because defining **HELLO** gives empty content to the definition and the parser never sees the argument. The work-around is to define **HELLO** as something, any value will do. For example:

```
#define HELLO 1
```

Support for Variable Argument Functions

ObjectCenter supports the use of variable function arguments loaded in source and object code. It supports both ANSI C (**stdarg.h**) and K&R C (**varargs.h**).

However, the following restriction applies to loading source code. You must not use a **structure** or **union** whose size is larger than **8 bytes** as a fixed argument in a variable-argument function. If you do, the interpreter computes the address of the arguments incorrectly.

Data Browser

Due to window size limitations in X, the **Data Browser** has a limit to the number of items it can contain. The limit is determined at run time. The font and model (**Motif** or **OPEN LOOK**) you use affect the limit. Using default fonts, you can create about 900 fields per data item under **Motif** and about 1500 under **OPEN LOOK**.

Manual Browser

You currently cannot view system **man** pages within the **Manual Browser** because the **Manual Browser** cannot format UNIX **man** pages. You can, however, view all the ObjectCenter entries.

Source Area

There is a limitation to the number of lines that you can list in the **Source** area. You can list files up to about 30,000 lines.

Limited Thread Support

ObjectCenter and ObjectCenter's C++ compiler (**CC**) currently support threads on the Solaris platform **ONLY** (Solaris 2.3 and _*possibly* later releases) as long as **Solaris-based** threads are defined. **POSIX** style and other thread types (such as GNU-based threads) are not supported. Threads are not supported at all on the HP and SunOS platforms. ObjectCenter's **cdm** does not support threads at all on any platform.

Static Functions in the Workspace

ObjectCenter now maintains all static initializers created for objects defined in the **Workspace**, so that in most cases you can **rerun** applications and the correct constructors will be called for those objects. However, if you **swap** a file from source to object code and **rerun** the program, ObjectCenter is unable to find constructors for objects declared in the **Workspace**. This limitation only occurs when re-executing code containing a **static** function by using **rerun** or by calling the function explicitly.

The following example illustrates the problem:

```
C++ 2 -> load -dd=off String.C
Loading (C++): -dd=off String.C
C++ 3 -> link
Linking from '/tmp_mnt/hosts/.../libC.sa.2.0' .... Linking
completed.
Linking from '/usr/lib/libc.sa.1.6' .... Linking completed.
C++ 4 -> String s4;
(class String *) 0x33e990 /* (class String) s4 */
C++ 5 -> rerun
Executing: a.out
Program exiting with return status = 0

Resetting to top level.
C++ 6 -> swap String.C
Unloading: String.C
Cannot open '/tmp_mnt/hosts/.../String.o'.
No such file or directory
Executing: /tmp_mnt/hosts/.../CC +d -g -c -
dd=off /tmp_mnt/hosts/.../String.C
/tmp_mnt/hosts/.../String.C:
Loading: String.o
C++ 7 -> rerun
Executing: a.out

----------------
"workspace":4, (workspace static initializer for s4), (Error
#156)
Calling undefined function String::String(void).
```

The workaround is to use the **load_header** command to load the definitions of classes you require. If you have received an error such as the one in the example, you can use the **unload workspace** command to remove existing definitions, then use the **history** command to help redefine functions and objects.

Limited Signal Handling Support

ObjectCenter does not support the **siginterrupt()** and **sigstack()** signal functions in component debugging mode.

ObjectCenter Version 2.1 and up supports calling **sigsetjmp()** and **siglongjump()** from an application running in component debugging mode with the following limitation:

> Every call to **sigsetjmp()** behaves as if the second argument were the value **0**, regardless of the actual value of the second argument. This means that a call to **siglongjmp()** will never restore the signal mask to the value it had at the time of the corresponding call to **sigsetjmp()**. In other respects, **sigsetjmp ()** and **siglongjmp()** behave exactly like **setjmp()** and **longjmp ()**.

There may be additional limitations in signal handling support on your platform. Please refer to the **"Anomalies"** section in your *Platform Guide*. For the *Platform Guide* related to your release of ObjectCenter, see the main **ObjectCenter Page** on CenterLine's website.