INTRODUCTION

This is the Installation Guide for the NIH Class Library (previously known as the "OOPS" Class Library) Revision 3.0.

The NIH Class Library is intended to be portable to a UNIX system compatible with either System V or 4.2/4.3BSD and which supports the AT&T C++ translator Release 2.00, Release 2.1, or other compatible C++ compiler. We have ported and tested this library on the following systems:

Sun-3 with SunOS 3.5
Sun-3 with SunOS 4.0
Sun-4 with SunOS 4.0

Send comments to:

Keith Gorlen
Building 12A, Room 2033
Computer Systems Laboratory
Division of Computer Research and Technology
National Institutes of Health
Bethesda, MD 20892

phone: (301) 496-1111
Internet : kgorlen@alw.nih.gov
uucp: uunet!nih-csl!kgorlen

GUIDE TO THIS DISTRIBUTION KIT

The NIH Class Library distribution kit consists of a main directory and the following subdirectories:

`errfac`	Error Message Facility source files
`lib`	Source files for the basic library classes
`test`	Test suite for the basic library classes
`vector`	Source files for the Vector classes
`vectest`	Test suite for the Vector classes
`ex`	Example programs for the book

The main directory is refered to as `NIHCL` in the following discussion, but may be placed anywhere.

Most subdirectories have files named `MAKEFILE` and `Makefile`. The `MAKEFILE` is used by the installation procedure, and should work with both the System V and BSD version of the `make` utility. The fancier `Makefile` is used for development, and may not work under BSD.

SUMMARY OF STEPS IN INSTALLING THE NIH CLASS LIBRARY

1. Update C++ system library and include files
2. Edit `NIHCL/Makefile`
3. Edit `NIHCL/lib/nihclconfig.h`
4. Build and install error message facility*
5. Build NIHCL basic classes, Vector classes, and test suite
6. Test basic classes and Vector classes
7. Build NIHCL basic classes, Vector classes, and test suite with multiple inheritance support
8. Test basic classes and Vector classes with multiple inheritance support
9. Install class libraries*
10. Build example programs
11. Test example programs

* root permission may be required

INSTALLING THE NIH CLASS LIBRARY

1. Update C++ system library and include files

No updates to R2.00 of the AT&T C++ Translator are required. However, if you are using R2.1, be sure to make the changes documented in the section *COMPILING UNDER AT&T C++ TRANSLATOR RELEASE 2.1* in the *NIH Class Library Release Notes*.

2. Edit NIHCL/Makefile

Edit `NIHCL/Makefile` to change make variables as needed for your environment. Here are the settings shipped with the distribution kit:

```
# C++ compiler
CC = CC

# C++ debug switch
CCDEBUG =
#CCDEBUG = -g

# C++ flags
# NOTE: Disable +p option when compiling with AT&T R2.1
#CCFLAGS = +p
#CCFLAGS =

# C++ include files
```

```
I = /usr/include/CC

# If using BSD
SYS = BSD
# If using System V
#SYS = SYSV

# Compile with nested types
# (works with AT&T R2.1 and GNU C++)
NESTED_TYPES =
#NESTED_TYPES = -DNESTED_TYPES

# Disable AT&T R2.0/R2.1 bug work-around code
BUGDEFS =
#BUGDEFS = -DBUG_bC2728 -DBUG_38 -DBUG_39 -DBUG_OPTYPECONST
# Defining BUG_TOOBIG disables code that
# prevents C compiler "yacc stack overflows" error
#BUGDEFS = -DBUG_bC2728 -DBUG_38 -DBUG_39 -DBUG_OPTYPECONST
-DBUG_TOOBIG

# Enable debug code
DEBUGDEFS =
#DEBUGDEFS = -DDEBUG_OBJIO -DDEBUG_PROCESS

# Flags for ln
#LNFLAGS =
LNFLAGS = -s

# If using "patch"
MAIN = _main.c_p
# If using "munch"
#MAIN = _main.c_m

# Target library for installation of Error Facility
LIB_ID = libC

# Target Directories for Installation

# directory for libnihcl.a
NIHCLLIBDIR = /usr/local/lib
# directory for NIHCL include files
NIHCLINCDIR = /usr/include/nihcl
# directory where ${LIB_ID}.a resides
CLIBDIR = /usr/local/lib/C++R2.0
# directory for errgen utility
ERRGENDIR = /usr/local/bin
# directory for errgen table file
```

```
ERRTABDIR = /usr/local/lib
# directory for errlib.h and errors.h
ERRINCDIR = $I
```

3. Edit NIHCL/lib files

3.1 Edit `nihclconfig.h`

The NIH Class Library source is configured for your system by setting flags in `NIHCL/lib/nihclconfig.h` which specify the machine model and operating system (UNIX variant).

To configure the NIH Class Library for one of the not yet implemented options, at least all of the parameters appearing in `nihclconfig.h` will have to be defined for that option.

The NIH Class Library should configure itself automatically for the following machines:

sun/mc68000
sun/sparc

Classes `Process`, `HeapProc`, `StackProc`, `Scheduler`, `Semaphore`, and `SharedQueue` have some machine-specific dependencies and will not work unless the `SETJMP()/LONGJMP()` functions are properly defined. See the *NIH Class Library Release Notes* for directions on porting the `Process` classes.

3.2 Edit `Object.h`

The file `Object.h` defines three versions of a preprocessor macro named `STRINGIZE`, which forms some symbol names by concatenating the class name argument with other strings. Each version does this a different way. The version for use with ANSI C preprocessors, conditionalized on the symbol `__STDC__`, uses `##` for concatenation. If you are not using an ANSI C preprocessor, defining the symbol `BS_NL` in `Object.h` selects the version that uses the sequence `\`<newline> as the concatenation separator, which seems to work with most System V UNIX systems. If you do not define `BS_NL`, you get the version of DEFINE_CLASS that uses an empty comment sequence (`/**/`), which works with most Berkeley UNIX systems.

`Object.h` should require no editing on most systems.

4. Install error message facility

(Skip this step for MASSCOMP/RTU)

```
su     (if installing in protected directory)
make errorfacility
```

This builds an error message registery facility and error processing library similar to `errcom` and the 3E library routines on the MASSCOMP.

The `errgen` program reads a `.err` file to determine a facility name, and then reads the file `${ERRTABDIR}/errgen_tab` to lookup the number assigned to that facility. The facility number determines the high-order bits of the error numbers which `errgen` assigns, assuring that error numbers used by different libraries do not coincide. Errgen produces a `.h` file containing error symbols and their assigned numbers, and a `.c` file containing a table of error messages and formatting information.

This step creates a module containing the error handling library routines named `errors.o` and adds it to `${CLIBDIR}/${LIB_ID}.a`, and it copies the files `errlib.h` and `errors.h` into the directory `${ERRINCDIR}.`

The test program testerr on `NIHCL/errfac` verifies that the error facilities have been built correctly. It returns the first and last error defined in the file `testerrs.err`.

5. Build the NIH Class Library, Vector classes, and test suite

```
make
```

6. Test the NIH Class Library

```
make verify
```

This runs the test suite and compares the output of each test program with the contents of a `.v` file containing the correct output. If the program runs correctly, you'll see the message "No differences encountered". Some tests such as `date`, `identdict`, `process`, `random`, `stack`, and `tim` produce output to the terminal. `date` outputs yesterday's, today's, and tomorrow's date. `identdict` dumps an identity dictionary. `random` prints out a list of random numbers. `stack` prints out a `CLTNEMPTY` error message to test error reporting, and `tim` prints out the current date and time.

The `error` test program frequently fails to compare because its output depends upon memory addresses that change from implementation to implementation. `error` should differ only in the object address printed in the `CLTNEMPTY` error message.

The output of `fdset` depends upon the maximum number of allowable file descriptors on your system. The test output was generated under SunOS 4.0, which has a limit of 64 file descriptors.

Several tests that print floating point numbers may fail to compare due to formatting differences.

The byte size of the object printed by ex8-1 may vary for different systems. The test output was produced by a Sun-3.

7. Build the NIH Class Library, Vector classes, and test suite with multiple inheritance support

```
make cleantest
make mi
```

8. Test the NIH Class Library, Vector classes, and test suite with multiple inheritance support

```
make verify
```

This runs the same tests as in Step 6, with similar results.

9. Install the NIH Class Libraries

```
su     (if installing in protected directory)
make install
```

The NIH Class Library archives `libnihcl.a`, `libnihclmi.a`, `libnihclvec.a`, and `libnihclvecmi.a` are copied to `${NIHCLLIBDIR}` and `ranlib` is executed on the libraries. All header files for basic classes are copied to directory `${NIHCLINCDIR}`.

10. Build example programs

```
make examples
```

11. Test example programs

```
make exverify
```

TROUBLE SHOOTING

YACC stack overflows

Some test programs may fail to compile because they are too complicated for your C compiler and get a "yacc stack overflow". Either increase the table space in your C compiler or simplify the program by breaking it up into separate functions. The inline copy constructors that Release 2.0 automatically generates are frequently the source of this error.

Explicitly defining non-inline copy constructors solves the problem. See the *Release Notes* for further details.

Problems with class Exception

Test programs error and except test class `Exception`, the the NIH Class Library exception handler. If these programs fail to perform correctly suspicion can be directed to the performance of system functions `setjmp()` and `longjmp()`.

Problems with Process classes

Progams `process` and `stackproc` test the NIH Class Library co-routine mechanism (classes `Process`, `HeapProc`, `StackProc`, and `Scheduler`), the object queue (class `SharedQueue`) and semaphore (class `Semaphore`). These are machine-dependent and rely on the presence of `alloca()`, which all systems do not provide, and on `setjmp()`/`longjmp()` being implemented by saving/restoring all machine registers, which is not always the case for all systems either. If the `process` or `stackproc` tests fail to compile, link, or run, check your system's implementations of `alloca()` `setjmp()`, and `longjmp()` -- you may need to implement your own versions.