CNCL

Communication Networks Class Library
Edition $Revision: 0.27 $, 1995

by M. Junius, M. Büter, D. Pesch, and others

Copyright © 1993-1995 Communication Networks, Aachen University of Technology, Germany.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into other languages, under the above conditions for modified versions.

1 Introduction to the CNCL

CNCL is a C++ library created at Communication Networks, Aachen University of Technology, Germany.

The main objective of this class library is to provide a common base for all C++ applications created at Communication Networks. Therefore, CNCL is both a class library featuring generic C++ classes as well as a simulation library with strong points in random number generation, statistics, and event driven simulation.

All globally visible classes and type definitions feature a "CN" prepended to their name to avoid collision with other class libraries, e.g. GNU libg++.

1.1 CNCL Class Hierarchy

`CNCL`	CNCL Static Members and Functions
`CNObject`	Root of the CNCL Hierarchy
`CNClass`	Class Description
`CNParam`	Abstract Parameter Base Class
`CNRNG`	Abstract Random Number Generator Base Class
`CNACG`	Additive RNG
`CNFiboG`	Fibonacci RNG
`CNFileG`	Data File RNG
`CNLCG`	Linear Congruence RNG
`CNMLCG`	Multiple Linear Congruence RNG
`CNTausG`	Tausworth RNG
`CNRndInt`	Random Integers
`CNRandom`	Abstract Random Distribution Base Class
`CNBeta`	Beta Distribution
`CNBinomial`	Binomial Distribution
`CNDeterm`	Deterministic Distribution
`CNDiracTab`	Distribution from Table
`CNDiscUniform`	Discrete Uniform Distribution
`CNErlang`	Erlang-k Distribution
`CNGeometric`	Geometric Distributed Random Numbers
`CNHyperExp`	HyperExp Distributed Random Numbers
`CNHyperGeom`	HyperGeom Distributed Random Numbers
`CNMDeterm`	Random Mix of Deterministic Distributions
`CNNegExp`	Negative Exponential Distribution
`CNNormal`	Normal Distribution
`CNLogNormal`	LogNormal Distribution
`CNRayleigh`	Rayleigh Distribution
`CNRice`	Rice Distribution
`CNPoisson`	Poisson Distribution
`CNRandomMix`	Mix of Several `CNRandom` Distributions
`CNTab`	Distribution from Table
`CNInterTab`	Distribution from Table (Interpolated Values)
`CNUniform`	Uniform Distribution
`CNWeibull`	Weibull Distribution
`CNStatistics`	Abstract Statistics Evaluation Base Class
`CNMoments`	Moments Evaluation
`CNLRE`	LRE Base Class
`CNLREF`	Evaluation by LRE F(x)
`CNLREG`	Evaluation by LRE G(x)
`CNDLREF`	Evaluation by Discrete LRE F(x)
`CNDLREG`	Evaluation by Discrete LRE G(x)
`CNBatches`	Evaluation by Batch Means
`CNAVLTree`	AVL balanced binary search tree
`CNAVLNode`	Node of AVL tree
`CNDLList`	Doubly Linked List of Objects
`CNDLObject`	Node of DoubleyLinked List
`CNDLIterator`	Iterator of Doubly Linked List
`CNQueue`	Abstract Queue Base Class
`CNQueueFIFO`	FIFO Queue
`CNQueueLIFO`	LIFO Queue
`CNQueueRandom`	Random Queue
`CNQueueSPT`	SPT Queue

`CNPrioQueueFIFO`	Priority Queue
`CNSink`	Sink
`CNStack`	Stack
`CNJob`	Standard Job for Queues
`CNDLObject`	
`CNEvent`	Generic Event
`CNNamed`	
`CNEventHandler`	Abstract Base Class for Event Handlers
`CNEventExploder`	Send Events to multiple EventHandlers
`CNEventList`	List of Events
`CNEventBaseSched`	
`CNEventScheduler`	Event Scheduler
`CNEventHeapSched`	Event Scheduler using a heap
`CNSimTime`	Simulation Time
`CNArray`	Abstract Array Base Class
`CNArrayObject`	Array of Pointer to `CNObject`
`CNArrayChar`	Array of `Char`
`CNArrayDouble`	Array of `Double`
`CNArrayFloat`	Array of `Float`
`CNArrayInt`	Array of `Int`
`CNArrayLong`	Array of `Long`
`CNArray2`	Base class for 2-dimensional arrays
`CNArray2Char`	Char array class
`CNArray2Double`	double array class
`CNArray2Float`	float array class
`CNArray2Int`	int array class
`CNArray2Long`	long array class
`CNArray2Object`	CNObjPtr array class
`CNKey`	Abstract Base Class for Keys
`CNKeyString`	String Key
`CNKeyInt`	Integer Key
`CNHashTable`	Abstract Hash Table Base Class
`CNHashStatic`	Hash Tables with Static Capacity
`CNHashDynamic`	Hash Tables with Dynamic Capacity
`CNHashIterator`	Sequential Iterator for Hash Tables
`CNManager`	Object Management Frontend
`CNCoord`	2-Dimensional Coordinates
`CNICoord`	2-Dimensional Integer Coordinates
`CNString`	Character String
`CNFormInt`	Integer as CNStrings
`CNFormFloat`	Doubles as CNStrings
`CNNamed`	Object with Name
`CNInt`	Integer derived from CNObject
`CNDouble`	Double derived from CNObject
`CNPipe`	UNIX Pipe
`CNSelect`	UNIX Select Interface
`CNNamed`	
`EZD`	Base Class for EZD Graphic Objects
`EZDDrawing`	Interface to EZD Drawings
`EZDPushButton`	Interface to EZD Push-Button

EZDWindow	Interface to EZD Windows
EZDDiagWin	Extra window with x-y diagram
EZDTextWin	EZD window for easy text display
EZDObject	Interface to EZD Object
EZDDiag	x-y diagram as an EZDObject
EZDBlock	Block with small rectangles for bit display
EZDPopUp	Interface to EZD popup menu
EZDQueue	Graphical Representation of a Queue
EZDServer	Graphical Representation of a Server
EZDText	EZD Object with Text
EZDTimer	Graphical Representation of a Timer
CNFClause	Clause of a fuzzy rule
CNFRule	Fuzzy rule
CNNamed	
CNFVar	Fuzzy variable
CNFRuleBase	Rule base and fuzzy inference engine
CNFSet	Fuzzy set abstract base class
CNFSetArray	Fuzzy set based on array with membership values
CNFSetLR	Fuzzy set with L and R functions
CNFSetTrapez	Fuzzy set with trapezium function
CNFSetTriangle	Fuzzy set with triangle function
CNFNumTriangle	Fuzzy numbers (triangle)
CNSDLProcess	SDL Process (dynamic)
CNSDLProcessData	SDL Process Description
CNSDLTimer	SDL Timer
CNSDLChannel	SDL Channel
CNEventHandler	
CNSDLManager	SDL Machine
CNSDLEnvironment	SDL Environment (system)
CNEvent	
CNSDLSignal	SDL Signals
CNReaderTbl	Table for adress of reader-function
CNPIO	persistent stream Object IO-formatting
CNObject	
CNPstream	abstract base class for persistent stream class
CNPiostream	persistent iostream format
CNObjectID	ID-Managment for persistent classes
CNPInt	class persistent CNInt
CNP<type>	Other persistent classes

1.2 Common CNCL Member Functions

CNCL requires that all classes have a common set of member functions available. These functions provide runtime type checking and type information, creation of objects via class descriptions, and safe type casts.

The common member functions are:

`CNClassDesc` *CLASS*`::class_desc() const;`

This function returns a pointer to the class description object, which must be available for every class in the CNCL hierarchy. This pointer is used for runtime type information.

`bool` *CLASS*`::is_a(CNClassDesc desc) const;`

This function allows runtime type checking. It returns `TRUE` if the queried object is type compatible with class `desc`.

`void` *CLASS*`::print(ostream &strm = cout) const;`

`void` *CLASS*`::dump(ostream &strm = cout) const;`

These functions output the object to the given stream `strm`. The function `print()` is defined in greater detail in the derived classes. The function `dump()` is intended for debug purposes.

The functions above are defined as pure virtual functions in the top-level class `CNObject`. They are required in every derived class.

Furthermore the following static member functions are required to allow object creation via the class description and safe type casts:

CLASS`*` *CLASS*`::cast_from_object(CNObject *obj);`

This function does a safe (*CLASS* `*`) type cast. It checks on type compatibility between the object passed with the `obj` pointer and *CLASS*. If this is a not true, an error message is printed and the program terminates.

The type checking may be disabled by defining the preprocessor macro `NO_TYPE_CHECK`, e.g. by supplying `-DNO_TYPE_CHECK` on the compiler's command line.

Example:

```
XYZ *px;                // CNClass XYZ derived from CNObject
ABC *pa;                // CNClass ABC derived from CNObject
DEF *pd;                // CNClass DEF derived from ABC
CNObject *po;

po = new DEF;           // Type compatible (C++ standard)
pd = (DEF *)po;         // The traditional way
pd = DEF::cast_from_object(po); // The CNCL way

pa = new DEF;
pd = DEF::cast_from_object(pa); // o.k.
px = XYZ::cast_from_object(pa); // Error
```

`CNObject *`*CLASS*`::new_object(CNParam *param = NIL);`

This function creates an object of type *CLASS*, optionally passing a pointer to a parameter object to the constructor.

It is used by the class description `CNClass` to create new objects.

Every class in CNCL requires a class description object and a pointer constant pointing to this class description. Following the CNCL convention, the description object is named *CLASS*`_desc` and the pointer is named `CN_`*CLASS*.

Example:

```
// Describing object for class XYZ
static CNClass XYZ_desc("XYZ", "$Revision: 0.27 $", XYZ::new_object);

// "Type" for type checking functions
CNClassDesc CN_XYZ = &XYZ_desc;
```

1.3 `CNgenclass` Script

To generate new classes for the CNCL hierarchy in a convenient way, the utility `CNgenclass` is provided. It generates a class framework with all the functions required by CNCL.

Usage:

`CNgenclass` *name base*

The required parameters are the name of the class to be created and the name of the base class. The result are two files in the current directory: *name*`.h` (class header file) and *name*`.c` (class implementation file).

Example:

```
CNgenclass MyClass CNObject
```

creates the files `MyClass.h` and `MyClass.c`. Please note that a leading "`CN`" is removed from the file names.

2 The Basic Classes of the CNCL Hierarchy

The following classes constitute the basics of CNCL. They provide the framework for runtime type checking, class descriptions, object management, and error handling.

2.1 CNCL — CNCL Static Members and Functions

SYNOPSIS

```
#include <CNCL/CNCL.h>
```

TYPE

None

BASE CLASSES

None

DERIVED CLASSES

CNObject

RELATED CLASSES

None

DESCRIPTION

The `CNCL` class contains only static members and functions for the class library's parameters and error handling. All classes in the CNCL hierarchy can directly access the static member functions, other code can call them via `CNCL::`*`func`*`()`.

ERROR HANDLING

The class `CNCL` provides common functionality for error handling. This is used by all classes to issue an error message or warning and terminate the program, if desired. The CNCL library also installs a `matherr()` handler using `CNCL::error()` for displaying an appropiate error message.

`enum ErrorType { err_fatal, err_abort, err_warning, err_ignore, err_default };`
The setting of the CNCL error handling:

`err_abort`

`err_fatal`

> Print error message, then terminate the program. (To do so, CNCL calls the exit handler installed with `set_exit_handler()`.) `err_abort` is like `err_fatal` but calls `abort()` for termination resulting in a core dump.

`err_warning`

> Print error message with "warning" prefix.

`err_ignore`

> No error message.

`err_ignore`

No error, just the message for info.

`err_default`

The default setting of the CNCL error handling. The default is `err_fatal` (may be set with `set_error()`).

The following member functions of `CNCL` can be used to manipulate the error handling.

`ErrorType get_error();`

Returns the current error handling type.

`ErrorType set_error(ErrorType err);`

Sets the error handling type to `err` and returns the previous value.

`static void set_exit_handler(void (*func)());`

Installs a function to be called on fatal errors. The default function will print a message and terminate the program by calling `exit()` (`err_fatal`) or `abort()` (`err_abort`).

`static void default_exit_handler()`

The CNCL default handler called on fatal errors.

ERROR MESSAGES

The following member function can be used to output error messages and/or terminate the program. Each of the functions accepts up to six `const char *` arguments, if the first one of these is `NIL`, then the default string (`"CNCL error: "`, `"CNCL warning: "`, `"CNCL: "` for `err_error`/`err_abort`, `err_warning`, `err_info`, respectively) is prepended to the output.

`void error(const char *msg, ...)`

Output error message, default error handling.

`void error(ErrorType err, const char *msg, ...)`

Output error message, error handling as specified by `err`.

`void fatal(const char *msg, ...)`

Output error message, fatal error handling.

`void warning(const char *msg, ...)`

Output error message, warning error handling.

`void info(const char *msg, ...)`

Output message.

`ostream& msg()`

Returns a reference to an output stream. Output to this stream will be appended to the next `error()`, `fatal()`, `warning()`, `info()` message. This is actually a `strstream` with a maximum capacity of `CNCL::STR_BUF_SIZE`.

UTILITIES

Outside the `CNCL` class the following constants and types are defined:

`NIL` The null pointer defined as `0`.

`TRUE` The boolean value true defined as `1`.

`FALSE` The boolean value false defined as `0`.

`bool` A boolean data type, actually `typedef int bool`.

2.2 CNObject — Root of the CNCL Hierarchy

SYNOPSIS

```
#include <CNCL/Object.h>
```

TYPE

`CN_OBJECT`

BASE CLASSES

CNCL

DERIVED CLASSES

CNClass, CNParam, ...

RELATED CLASSES

CNClass, CNParam

DESCRIPTION

`CNObject` is the actual base of the CNCL inheritance tree. It must be supported by all derived classes.

`virtual CNClassDesc class_desc() const;`

Returns the class description (pointer to instance of CNClass) for runtime type information.

`virtual bool is_a(CNClassDesc desc) const;`

Returns TRUE if the queried object is of type `desc`, else FALSE.

`virtual void print(ostream &strm = cout) const = 0;`

Output object to stream.

`virtual void dump(ostream &strm = cout) const = 0;`

Output object to stream for debug purpose.

`virtual int store_on(CNPstream &);`

`virtual int storer(CNPstream &);`

These functions are support functions for persistent objects, so they are not implemented in all derived classes. They are yielding a warning if they are called for a non-persistent object.

UTILITIES

`Object.h` defines the following operators for easily writing objects to streams:

```
ostream &operator << (ostream &strm, const CNObject &obj);
ostream &operator << (ostream &strm, const CNObject *obj);
```

Write object to stream using the `print()` member function. It is safe to output a null pointer `obj`, in this case "`(NIL)`" is printed.

2.3 CNClass — Class Description

SYNOPSIS

```
#include <CNCL/Class.h>
```

TYPE

`CN_CLASS`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNObject

DESCRIPTION

`CNClass` is used for the type description objects. For each class in the CNCL hierarchy there is a corresponding object of type `CNClass`. A pointer to this object is used for the CNCL runtime type information.

Constructors:

`CNClass(char *name, char *version, CNObject *(*func)(CNParam *p));`

The constructor of `CNClass` takes three arguments: the class' name, the class' version and a pointer to the static `CLASS::new_object()` member function of classes that provide this functionality. The function pointer may be `NIL` for classes that do not provide this, such as abstract base classes.

In addition to the member functions required by CNCL, `CNClass` provides:

`const char *name() const;`

`const char *get_name() const;`

Returns the class name.

`const char *version() const;`

`const char *get_version() const;`

Returns the class version.

`CNObject *new_object(CNParam *param = NIL) const;`

`CNObject *new_object(Param &param) const;`

Creates new objects via the corresponding class' static member function `new_object()`. The `param` object is used to pass optional arguments to the constructor.

`static CNClass *cast_from_object(CNObject *obj);`

Safes `(CNClass *)` type cast.

2.4 CNParam — Abstract Parameter Base Class

SYNOPSIS

```
#include <CNCL/Param.h>
```

TYPE

`CN_PARAM`

BASE CLASSES

CNObject

DERIVED CLASSES

RELATED CLASSES

CNClass

DESCRIPTION

`CNParam` is the abstract base for parameter classes used to pass constructor parameters to objects in a general way. This will be used in the forthcoming CNCL object management.

3 Random Numbers

CNCL provides a variety of random number generators, ranging from the very simple linear congruence generator to more sophisticated ones. Also included is a random number generator which reads random data from a file.

Random numbers are a crucial base of every simulation; most of the pseudo random number generators included in CNCL have their faults and short-comings.

The base random number generators are used by the random distribution classes to generate random numbers with the desired distribution.

3.1 CNRNG — Abstract Random Number Generator Base Class

SYNOPSIS

```
#include <CNCL/RNG.h>
```

TYPE

`CN_RNG`

BASE CLASSES

CNObject

DERIVED CLASSES

CNACG, CNFiboG, CNFileG, CNLCG, CNMLCG, CNTausG

RELATED CLASSES

CNRandom

DESCRIPTION

`CNRNG` is the abstract base class for all CNCL random number generators. It defines the common interface.

Constructors:

```
CNRNG();
CNRNG(CNParam *param);
```

Initializes `CNRNG`.

In addition to the member functions required by CNCL, `CNRNG` provides:

```
unsigned long as_long();
```

This function returns an unsigned integer in the range 0 ... 2^31-1. It uses the as_long32() function from the actual CNRNG to draw a random number and truncates it down to 31bit.

```
virtual unsigned long as_long32() = 0;
```

This function is used to draw a random number from the actual CNRNG (derived from the `CNRNG` class). The result is an unsigned integer in the range 0 ... 2^32-1 provided the class is able to produce 32bit random numbers.

```
virtual bool has_long32() = 0;
```

This function tells whether the actual CNRNG is able to produce 32bit integer values or not.

```
virtual void reset() = 0;
```

Resets the CNRNG to its initial state.

`float as_float();`

`double as_double();`

These functions draw a random number in the range 0 ... 1 and return the result as a `float` or `double` value.

3.2 CNACG — Additive RNG

SYNOPSIS

```
#include <CNCL/ACG.h>
```

TYPE

`CN_ACG`

BASE CLASSES

CNRNG

DERIVED CLASSES

None

RELATED CLASSES

CNRandom

DESCRIPTION

`CNACG` is the additive random number generator class.

Constructors:

```
CNACG(unsigned long seed = 0, int size = 55);
CNACG(CNParam *param);
```

Initializes `CNACG`.

In addition to the member functions required by CNCL, `CNACG` provides:

```
virtual unsigned long as_long32();
```

Draws a random number. The result is an unsigned integer in the range 0 ... 2^32-1.

```
virtual bool has_long32();
```

Returns TRUE because the CNACG is able to produce 32bit integer values.

```
virtual void reset();
```

Resets the CNACG to its initial state.

3.3 CNFiboG — Fibonacci RNG

SYNOPSIS

```
#include <CNCL/FiboG.h>
```

TYPE

`CN_FIBOG`

BASE CLASSES

CNRNG

DERIVED CLASSES

None

RELATED CLASSES

CNRandom

DESCRIPTION

`CNFiboG` is the Fibonacci random number generator class.

Constructors:

```
CNFiboG(CNParam *param);
CNFiboG(unsigned long init = 54217137);
```

Initializes `CNFiboG` with a 97 elements circular queue and initial seed.

In addition to the member functions required by CNCL, `CNFiboG` provides:

```
virtual unsigned long as_long32();
```

Draws a random number. The result is an unsigned integer in the range 0 ... 2^32-1.

```
virtual bool has_long32();
```

Returns TRUE because CNFiboG is able to produce 32bit integer values.

```
virtual void reset();
```

Resets the CNFiboG to its initial state.

```
void seed(unsigned long);
void seed(unsigned long *);
```

Reinitializes the generator with a 97 elements circular queue and seed.

3.4 CNFileG — Data File RNG

SYNOPSIS

```
#include <CNCL/FileG.h>
```

TYPE

`CN_FILEG`

BASE CLASSES

CNRNG

DERIVED CLASSES

None

RELATED CLASSES

CNRandom

DESCRIPTION

`CNFileG` is a data file random number generator class. It reads random numbers from a disk file, e.g. data from PURAN2.

Constructors:

```
CNFileG(char *filename, bool par = FALSE);
CNFileG(CNParam *param);
```

Initializes `CNFileG` with data file `filename` and sets parity check if required, e.g. parity check for PURAN2.

In addition to the member functions required by CNCL, `CNFileG` provides:

```
virtual unsigned long as_long32();
```

Draws a random number. The result is an unsigned integer in the range 0 ... 2^32-1.

```
virtual bool has_long32();
```

Returns TRUE because the CNFileG is able to produce 32bit integer values.

```
virtual void reset();
```

Resets the CNFileG to its initial state.

```
void newfile(char *filename, bool par = FALSE);
```

Opens a new file for reading random number data and indicates if data requires parity check.

```
unsigned int wrong_parity();
```

Gets the number of wrong parity checks while reading data from a file, e.g. data from PURAN2.

3.5 CNLCG — Linear Congruence RNG

SYNOPSIS

```
#include <CNCL/LCG.h>
```

TYPE

`CN_LCG`

BASE CLASSES

CNRNG

DERIVED CLASSES

None

RELATED CLASSES

CNRandom

DESCRIPTION

`CNLCG` is a linear congruence random number generator class.

Constructors:

```
CNLCG(unsigned long seed = 929);
CNLCG(CNParam *param);
```

Initializes `CNLCG` with initial `seed`.

In addition to the member functions required by CNCL, `CNLCG` provides:

```
virtual unsigned long as_long32();
```

Draws a random number. The result is an unsigned integer in the range 0 ... 2^31-1.

```
virtual bool has_long32();
```

Returns FALSE because LCG produces only 31bit integer values.

```
virtual void reset();
```

Resets the CNLCG to its initial state.

```
unsigned long seed();
void seed(unsigned long);
```

Gets/sets the CNLCG seed value.

3.6 CNMLCG — Multiple Linear Congruence RNG

SYNOPSIS

```
#include <CNCL/MLCG.h>
```

TYPE

`CN_CNMLCG`

BASE CLASSES

CNRNG

DERIVED CLASSES

None

RELATED CLASSES

CNRandom

DESCRIPTION

`CNMLCG` is a multiple linear congruence random number generator class combining the results of two different CNLCGs

Constructors:

```
CNMLCG(long seed1 = 0, long seed2 = 1);
CNMLCG(CNParam *param);
```

Initializes `CNMLCG` with two seeds.

In addition to the member functions required by CNCL, `CNMLCG` provides:

```
virtual unsigned long as_long32();
```

Draws a random number. The result is an unsigned integer in the range 0 ... 2^31-1.

```
virtual bool has_long32();
```

Returns FALSE because CNMLCG produces only 31bit intger values.

```
virtual void reset();
```

Resets the CNMLCG to its initial state.

```
long seed1();
void seed1(long);
long seed2();
void seed2(long);
```

Gets/sets the two seed values.

```
void reseed(long, long);
```

Sets both seed values.

3.7 CNTausG — Tausworth RNG

SYNOPSIS

```
#include <CNCL/TausG.h>
```

TYPE

`CN_TAUSG`

BASE CLASSES

CNRNG

DERIVED CLASSES

None

RELATED CLASSES

CNRandom

DESCRIPTION

`CNTausG` is a Tausworth random number generator class.

Constructors:

```
CNTausG();
CNTausG(CNParam *param);
```

Initializes `CNTausG`.

In addition to the member functions required by CNCL, `CNTausG` provides:

```
virtual unsigned long as_long32();
```

Draws a random number. The result is an unsigned integer in the range 0 ... 2^32-1.

```
virtual bool has_long32();
```

Returns TRUE because CNTausG is able to produce 32bit integer values.

```
virtual void reset();
```

Resets the CNTausG to its initial state.

3.8 CNRndInt — Random Integers

SYNOPSIS

```
#include <CNCL/RndInt.h>
```

TYPE

`CN_RNDINT`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNRndInt` generates uniform distributed random integers in a given interval. The result is the same as provided by the `CNDiscUniform` distribution, but `CNRndInt` is more efficient.

BEWARE: do NOT use the `CNLCG` RNG as a base generator for `CNRndInt`.

Constructors:

```
CNRndInt();
CNRndInt(CNParam *param);
CNRndInt(long low, long high, CNRNG *gen);
CNRndInt(long high, CNRNG *gen);
CNRndInt(CNRNG *gen);
```

Initializes a `CNRndInt` with base RNG `gen` and upper/lower interval limits `low`/`high`.

In addition to the member functions required by CNCL, `CNRndInt` provides:

```
CNRNG *generator() const;
CNRNG *generator(CNRNG *gen);
```

Gets/sets the base CNRNG used by `CNRndInt`.

```
long low() const;
long high() const;
long low(long x);
long high(long x);
```

Gets/sets the upper/lower interval limits.

```
long operator()();
long operator()(long high);
long operator()(long low, long high);
long as_long();
long as_long(long high);
long as_long(long low, long high);
```

Draws a `long` random integer. Interval limits may be passed as optional parameters.

```
int as_int();
int as_int(long high);
int as_int(long low, long high);
```

Draws a `int` random integer. Interval limits may be passed as optional parameters.

3.9 CNRandom — Abstract Random Distribution Base Class

SYNOPSIS

```
#include <CNCL/Random.h>
```

TYPE

`CN_RANDOM`

BASE CLASSES

CNObject

DERIVED CLASSES

CNBeta, CNBinomial, CNDeterm, CNDiracTab, CNDiscUniform, CNErlang, CNGeometric, CNHyperExp, CNHyperGeom, CNInterTab, CNLogNormal, CNMDeterm, CNNegExp, CNNormal, CNPoisson, CNRandomMix, CNRayleigh, CNRice, CNTab, CNUniform, CNWeibull

RELATED CLASSES

CNRNG

DESCRIPTION

`CNRandom` is the abstract base class for all CNCL random number distributions. It defines a common interface to access to all derived RNG classes in a common way.

Constructors:

```
CNRandom(CNRNG *gen);
CNRandom(CNParam *param);
```

Initializes `CNRandom` with a base random number generator.

In addition to the member functions required by CNCL, `CNRandom` provides:

```
CNRNG *generator();
```

Returns a pointer to the actually used CNRNG.

```
void generator(CNRNG *gen);
```

Sets the CNRNG used by `CNRandom` to `gen`.

```
virtual double operator() () = 0;
```

Draws a random number from the distribution. This operator must be defined in the derived classes.

3.10 CNBeta — Beta Distribution

SYNOPSIS

```
#include <CNCL/Beta.h>
```

TYPE

`CN_BETA`

BASE CLASSES

CNRandom

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNBeta` is a class for generating beta distributed random numbers.

Constructors:

```
CNBeta();
CNBeta(CNParam *param);
CNBeta(long a, long b, CNRNG *gen);
```

Initializes a `CNBeta` distribution with a base random number generator `gen` and the parameters `a` and `b`.

In addition to the member functions required by CNCL, `CNBeta` provides:

```
virtual double operator() ();
```

Draws a beta distributed random number.

3.11 CNBinomial — Binomial Distribution

SYNOPSIS

```
#include <CNCL/Binomial.h>
```

TYPE

`CN_BINOMIAL`

BASE CLASSES

CNRandom

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNBinomial` is a class for generating binomial distributed random numbers.

Constructors:

```
CNBinomial();
CNBinomial(CNParam *param);
CNBinomial(int n, double u, CNRNG *gen);
```

Initializes a `CNBinomial` distribution with a base random number generator `gen`, and the parameters `n` and `u`.

In addition to the member functions required by CNCL, `CNBinomial` provides:

```
int n();
int n(int xn);
double u();
double u(double xu);
```

Gets/sets the values for n and u.

```
virtual double operator() ();
```

Draws a binomial distributed random number.

3.12 CNDeterm — Deterministic Distribution

SYNOPSIS

```
#include <CNCL/Determ.h>
```

TYPE

`CN_DETERM`

BASE CLASSES

CNRandom

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNDeterm` is a class for generating deterministic "random" numbers, i.e. it always generates the same value. This is useful for mixed distributions.

Constructors:

```
CNDeterm();
CNDeterm(CNParam *param);
CNDeterm(double value, CNRNG *gen);
```

Initializes a `CNDeterm` distribution with `value`.

In addition to the member functions required by CNCL, `CNDeterm` provides:

```
double value();
double mean();
double value(double x);
double mean(double x);
```

Gets/sets the value.

```
virtual double operator() ();
```

Draws a deterministic (i.e. not random) number.

3.13 CNDiracTab — Distribution from Table of CDF

SYNOPSIS

```
#include <CNCL/DiracTab.h>
```

TYPE

`CN_DIRACTAB`

BASE CLASSES

CNRandom

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNDiracTab` generates random numbers according to a table with the distribution function.

Constructors:

```
CNDiracTab();
CNDiracTab(CNParam *param);
CNDiracTab(CNDiracTabEntry *tab, long length, CNRNG *gen);
```

Initializes a `CNDiracTab` distribution with a base random number generator `gen` and a table `tab` with length `length`.

Example:

```
CNDiracTabEntry datafield[] = {{ 0.1, 2 },{ 0.3, 4 }, { 0.6, 6 }};
CNDiracTab example(datafield, 3, &generator);
```

In addition to the member functions required by CNCL, `CNDiracTab` provides:

```
virtual double operator() ();
```

Draws a random number.

3.14 CNDiscUniform — Discrete Uniform Distribution

SYNOPSIS

```
#include <CNCL/DiscUniform.h>
```

TYPE

`CN_DISCUNIFORM`

BASE CLASSES

CNRandom

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNDiscUniform` is a discrete uniform distribution with interval limits `low` and `high`. It generates discrete (i.e. integer) values.

Constructors:

```
CNDiscUniform();
CNDiscUniform(CNParam *param);
CNDiscUniform(long low, long high, CNRNG *gen);
```

Initializes a `CNDiscUniform` distribution with a base random number generator `gen` and the parameters `low` and `high`.

In addition to the member functions required by CNCL, `CNDiscUniform` provides:

```
long low();
long low(long x);
long high();
long high(long x);
```

Gets/sets the values for the interval limits.

```
virtual double operator() ();
```

Draws a discrete uniform distributed random number.

3.15 CNErlang — Erlang-k Distribution

SYNOPSIS

```
#include <CNCL/Erlang.h>
```

TYPE

`CN_ERLANG`

BASE CLASSES

CNRandom

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNErlang` is a class for generating Erlang-k distributed random numbers.

Constructors:

```
CNErlang();
CNErlang(CNParam *param);
CNErlang(double mean, double variance, CNRNG *gen);
```

Initializes a `CNErlang` distribution with a base random number generator `gen`, mean value `mean`, and variance `variance`.

In addition to the member functions required by CNCL, `CNErlang` provides:

```
double mean();
double mean(double x);
double variance();
double variance(double x);
```

Gets/sets the values for mean and variance.

```
virtual double operator() ();
```

Draws a Erlang-k distributed random number.

3.16 CNGeometric — Geometric Distributed Random Numbers

SYNOPSIS

```
#include <CNCL/Geometric.h>
```

TYPE

`CN_GEOMETRIC`

BASE CLASSES

CNRandom

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNGeometric` is a class for generating geometric distributed random numbers.

Constructors:

```
CNGeometric();
CNGeometric(CNParam *param);
CNGeometric(double mean, CNRNG *gen);
```

Initializes a `CNGeometric` distribution with a base random number generator `gen` and the value `mean` as a parameter for the random generator.

NOTE: Please do NOT take this parameter as the distrbution's mean ('MEAN') value. This can be calculated as:

```
MEAN = 1 / ( 1 - mean )
```

In addition to the member functions required by CNCL, `CNGeometric` provides:

```
double mean();
double mean(double x);
```

Gets/sets the values for mean.

```
virtual double operator() ();
```

Draws a geometric distributed random number.

3.17 CNHyperExp — Hyperexponential Distribution

SYNOPSIS

```
#include <CNCL/HyperExp.h>
```

TYPE

`CN_HYPEREXP`

BASE CLASSES

CNRandom

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNHyperExp` is a class for generating hyperexponential distributed random numbers. `CNHyperExp` is a mixed distribution of two negative exponential distributions (H2).

Constructors:

```
CNHyperExp();
CNHyperExp(CNParam *param);
CNHyperExp(double p, double m1, double m2, CNRNG *gen);
```

Initializes a `CNHyperExp` distribution with a base random number generator `gen`, the mixprobability `p` and the intensity parameters `m1` and `m2`.

In addition to the member functions required by CNCL, `CNHyperExp` provides:

```
virtual double operator() ();
```

Draws a hyperexponential distributed random number.

3.18 CNHyperGeom — Hypergeometrical Distribution

SYNOPSIS

```
#include <CNCL/HyperGeom.h>
```

TYPE

`CN_HYPERGEOM`

BASE CLASSES

CNRandom

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNHyperGeom` is a class for generating hypergeometrical distributed random numbers.

Constructors:

```
CNHyperGeom();
CNHyperGeom(CNParam *param);
CNHyperGeom(double mean, double variance, CNRNG *gen);
```

Initializes a `CNHyperGeom` distribution with a base random number generator `gen`, mean value `mean` and variance `variance`.

In addition to the member functions required by CNCL, `CNHyperGeom` provides:

```
double mean();
double mean(double x);
double variance();
double variance(double x);
```

Gets/sets the values for mean and variance.

```
virtual double operator() ();
```

Draws a hypergeometrical distributed random number.

3.19 CNInterTab – Distribution from Table of CDF (Interpolated)

SYNOPSIS

```
#include <CNCL/InterTab.h>
```

TYPE

`CN_INTERTAB`

BASE CLASSES

CNTab

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNInterTab` is a class for generating random numbers from a table. It works as `CNTab`, but the values are interpolated between the table entries.

Constructors:

```
CNInterTab();
CNInterTab(CNParam *param);
CNInterTab(double *tab, long length, CNRNG *gen);
```

Initializes a `CNInterTab` distribution with a base random number generator `gen`, a table with the samples `tab` and the length of the table `length`.

Example:

```
double datafield[]={ 2, 4, 6, 8, 10, 12 };
CNInterTab ex(datafield, 6, generator);
```

In addition to the member functions required by CNCL, `CNInterTab` provides:

```
virtual double operator() ();
```

Draws a random number.

3.20 CNLogNormal — Log-normal Distribution

SYNOPSIS

```
#include <CNCL/LogNormal.h>
```

TYPE

`CN_LOGNORMAL`

BASE CLASSES

CNNormal

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNLogNormal` is a class for generating logarithmic normal distributed random numbers.

Constructors:

```
CNLogNormal();
CNLogNormal(CNParam *param);
CNLogNormal(double mean, double variance, CNRNG *gen);
```

Initializes a `CNLogNormal` distribution with a base random number generator `gen`, mean value `mean` and variance `variance`.

In addition to the member functions required by CNCL, `CNLogNormal` provides:

```
double mean();
double mean(double x);
double variance();
double variance(double x);
```

Gets/sets the values for mean and variance.

```
virtual double operator() ();
```

Draws a logarithmic normal distributed random number.

3.21 CNMDeterm — Random Mix of Deterministic Values

SYNOPSIS

```
#include <CNCL/MDeterm.h>
```

TYPE

`CN_MDETERM`

BASE CLASSES

CNRandom

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNMDeterm` is a class for generating a random mix of two deterministic values.

Constructors:

```
CNMDeterm();
CNMDeterm(CNParam *param);
MDeter(double m, double d1, double d2, CNRNG *gen);
```

Initializes a `CNMDeterm` distribution with a base random number generator `gen`, a mixing parameter `m` and two values `d1` and `d2`.

In addition to the member functions required by CNCL, `CNMDeterm` provides:

```
virtual double operator() ();
```

Draws a random number.

3.22 CNNegExp — Negative Exponential Distribution

SYNOPSIS

```
#include <CNCL/NegExp.h>
```

TYPE

`CN_NEGEXP`

BASE CLASSES

CNRandom

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNNegExp` is a class for generating negative exponential distributed random numbers.

Constructors:

```
CNNegExp();
CNNegExp(CNParam *param);
CNNegExp(double mean, CNRNG *gen);
```

Initializes a `CNNegExp` distribution with a base random number generator `gen` and mean value `mean`.

In addition to the member functions required by CNCL, `CNNegExp` provides:

```
double mean();
double mean(double x);
```

Gets/sets the values for mean.

```
virtual double operator() ();
```

Draws a negative exponential distributed random number.

3.23 CNNormal — Normal Distribution

SYNOPSIS

```
#include <CNCL/Normal.h>
```

TYPE

`CN_NORMAL`

BASE CLASSES

CNRandom

DERIVED CLASSES

CNRayleigh, CNRice

RELATED CLASSES

CNRNG

DESCRIPTION

`CNNormal` is a class for generating normal (Gaussian) distributed random numbers.

Constructors:

```
CNNormal();
CNNormal(CNParam *param);
CNNormal(double mean, double variance, CNRNG *gen);
```

Initializes a `CNNormal` distribution with a base random number generator `gen`, mean value `mean` and variance `variance`.

In addition to the member functions required by CNCL, `CNNormal` provides:

```
double mean();
double mean(double x);
double variance();
double variance(double x);
```

Gets/sets the values for mean and variance.

```
virtual double operator() ();
```

Draws a normal distributed random number.

3.24 CNPoisson — Poisson Distribution

SYNOPSIS

```
#include <CNCL/Poisson.h>
```

TYPE

`CN_POISSON`

BASE CLASSES

CNRandom

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNPoisson` is a class for generating Poisson distributed random numbers.

Constructors:

```
CNPoisson();
CNPoisson(CNParam *param);
CNPoisson(double mean, CNRNG *gen);
```

Initializes a `CNPoisson` distribution with a base random number generator `gen` and mean value `mean`.

In addition to the member functions required by CNCL, `CNPoisson` provides:

```
double mean();
double mean(double x);
```

Gets/sets the values for mean.

```
virtual double operator() ();
```

Draws a Poisson distributed random number.

3.25 CNRandomMix — Mix of Several CNRandom Distributions

SYNOPSIS

```
#include <CNCL/RandomMix.h>
```

TYPE

CN_RANDOMMIX

BASE CLASSES

CNRandom

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

CNRandomMix generates random numbers from a mix of several CNRandom distributions.

Constructors:

```
CNRandomMix();
CNRandomMix(CNParam *param);
CNRandomMix(CNRandomMixEntry *tab, int length, CNRNG *gen);
```

Initializes a CNRandomMix distribution with a base random number generator gen and a table of several distributions.

The following example shows how to use a CNRandomMixEntry table with 3 different distributions and their corresponding probabilities for a CNRandomMix distribution:

```
CNLogNormal ex1(10, 0.5, &generator1);
CNNormal    ex2(12, 0.3, &generator2);
CNHyperExp  ex3(10, 0.5, &generator3);

CNRandomMixEntry tab[] = {
    { 0.1, &ex1 }, { 0.5, &ex2 }, { 0.4 &ex3 }
};

CNRandomMix ex(tab, 3, &generator);
```

In addition to the member functions required by CNCL, CNRandomMix provides:

```
virtual double operator() ();
```

Draws a random number.

3.26 CNRayleigh — Rayleigh Distribution

SYNOPSIS

```
#include <CNCL/Rayleigh.h>
```

TYPE

`CN_RAYLEIGH`

BASE CLASSES

CNNormal

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNRayleigh` is a class for generating Rayleigh distributed random numbers.

Constructors:

```
CNRayleigh();
CNRayleigh(CNParam *param);
CNRayleigh(double variance, CNRNG *gen);
```

Initializes a `CNRayleigh` distribution with a base random number generator `gen` and variance `variance`.

The variance is the one passed to the `CNNormal` base class, not the actual value of the Rayleigh distribution.

In addition to the member functions required by CNCL, `CNRayleigh` provides:

```
virtual double operator() ();
```

Draws a Rayleigh distributed random number.

3.27 CNRice — Rice Distribution

SYNOPSIS

```
#include <CNCL/Rice.h>
```

TYPE

`CN_RICE`

BASE CLASSES

CNNormal

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNRice` is a class for generating Rice distributed random numbers.

Constructors:

```
CNRice();
CNRice(CNParam *param);
CNRice(double mean, double variance, CNRNG *gen);
```

Initializes a `CNRice` distribution with a base random number generator `gen`, mean value `mean` and variance `variance`.

The mean value and variance are those passed to the `CNNormal` base class, not the actual values of the Rice distribution.

In addition to the member functions required by CNCL, `CNRice` provides:

```
virtual double operator() ();
```

Draws a Rice distributed random number.

3.28 CNTab — Distribution from Table of CDF

SYNOPSIS

```
#include <CNCL/Tab.h>
```

TYPE

`CN_TAB`

BASE CLASSES

CNRandom

DERIVED CLASSES

CNInterTab

RELATED CLASSES

CNRNG

DESCRIPTION

`CNTab` requires a table with samples.

Constructors:

```
CNTab();
CNTab(CNParam *param);
CNTab(double *tab, long length, CNRNG *gen);
```

Initializes a `CNTab` distribution with a base random number generator `gen`, a table of values `tab`, and length `length`.

Example:

```
double datafield[]={ 2, 4, 6, 8, 10, 12 };
CNTab ex(datafield, 6, &generator);
```

In addition to the member functions required by CNCL, `CNTab` provides:

```
virtual double operator() ();
```

Draws a random number.

3.29 CNUniform — Uniform Distribution

SYNOPSIS

```
#include <CNCL/Uniform.h>
```

TYPE

`CN_UNIFORM`

BASE CLASSES

CNRandom

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNUniform` generates uniform distributed random numbers within the interval limits `low` and `high`.

Constructors:

```
CNUniform();
CNUniform(CNParam *param);
CNUniform(double low, double high, CNRNG *gen);
```

Initializes a `CNUniform` distribution with a base random number generator `gen` and interval limits `low` and `high`.

In addition to the member functions required by CNCL, `CNUniform` provides:

```
double low();
double low(double x);
double high();
double high(double x);
```

Gets/sets the values for the limits.

```
virtual double operator() ();
```

Draws a uniform distributed random number.

3.30 CNWeibull — Weibull Distribution

SYNOPSIS

```
#include <CNCL/Weibull.h>
```

TYPE

`CN_WEIBULL`

BASE CLASSES

CNRandom

DERIVED CLASSES

None

RELATED CLASSES

CNRNG

DESCRIPTION

`CNWeibull` generates Weibull distributed random numbers with a form factor `alpha` and a scale factor `beta`.

Constructors:

```
CNWeibull();
CNWeibull(CNParam *param);
CNWeibull(double alpha, double beta, CNRNG *gen);
```

Initializes a `CNWeibull` distribution with a base random number generator `gen` and parameters `alpha` and `beta`.

In addition to the member functions required by CNCL, `CNWeibull` provides:

```
double alpha();
double alpha(double x);
double beta();
double beta(double x);
```

Gets/sets the values for alpha and beta.

```
virtual double operator() ();
```

Draws a Weibull distributed random number.

c -*-texinfo-*-

4 Statistical Evaluation in CNCL

CNCL provides essentially four methods for a statistical evaluation of simulation results:

- Moments: mean, variance, coefficient of variation etc., without error measure (`CNMoments`)
- Limited-Relative-Error (LRE): distribution function, complementary distribution function, and local correlation coefficient with error measure (`CNLREF`, `CNLREG`)
- Discrete-Limited-Relative-Error (DLRE): same as LRE, but only for discrete (complementary) distribution functions with well–known x–values
- Batch-Means: distribution function with relative Bayes error, histogram; mean (with relative Bayes error) and variance of the group means (`CNBatches`).

All classes are derived from the base class `CNStatistics`.

4.1 CNStatistics — Abstract Statistics Base Class

SYNOPSIS

```
#include <CNCL/Statistics.h>
```

TYPE

`CN_STATISTICS`

BASE CLASSES

CNObject

DERIVED CLASSES

CNMoments, CNLRE, CNBatches

RELATED CLASSES

None

DESCRIPTION

`CNStatistics` is the base class of all statistics classes. It defines the common interface.

Constructors:

```
CNStatistics();
CNStatistics( CNParam *param );
```

Initializes `CNStatistics`.

The evaluation phases and types supported by `CNStatistics` are:

```
enum Phase { INITIALIZE=0, ITERATE=1, END=2 };
```

Different phases of a statistical evaluation with their settings.

```
enum Type { DF=0, CDF=1 };
```

`DF` as density function, `CDF` as complementary density function.

In addition to the member functions required by CNCL, `CNStatistics` provides:

```
virtual void put( double ) = 0;
```

Input function of the class.

```
virtual double mean() const = 0;
```

Returns the mean value of the input sequence.

```
virtual double variance() const = 0;
```

Returns variance or the input sequence.

```
virtual long trials() const = 0;
```

Returns number of input values.

`virtual double min() const = 0;`

Returns minimum input value.

`virtual double max() const = 0;`

Returns maximum input value.

`virtual boolean end() const = 0;`

Returns `TRUE` if end of evaluation is reached else `FALSE`.
NOTE: In case of a `CNMoments` evaluation, the return value is always `FALSE`.

`virtual void reset() = 0;`

Resets the evaluation.

`virtual Phase status() const = 0;`

Returns the state of evaluation;

4.2 CNMoments — Moments Evaluation

SYNOPSIS

```
#include <CNCL/Moments.h>
```

TYPE

`CN_Moments`

BASE CLASSES

CNStatistics

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

The `CNMoments` class yields the moments of an input sequence:

- mean,
- variance and relative variance (squared coefficient of variation),
- 2nd and 3rd zero moment,
- 3rd central moment,
- deviation and relative deviation (coefficient of variation),
- skewness.

Constructors:

`CNMoments( CNParam *param )`

`CNMoments( char *new_name = "no name" )`

Initializes a `CNMoments` evaluation with `new_name` as name of evaluation.

In addition to the member functions required by CNCL and `CNStatistics`, `CNMoments` provides:

`double mean() const;`

Returns mean of the input values.

`double variance() const;`

Returns variance.

`double M_2() const;`

Returns 2nd moment.

`double M_3() const;`

Returns 3rd moment.

`double Z_3() const;`

Returns 3rd central moment.

`double skewness() const;`

Returns skewness.

`double relative_variance() const;`

Returns relative variance (squared coefficient of variation).

`double relative_deviation() const;`

Returns relative deviation (coefficient of variation).

4.3 CNLREF, CNLREG — Evaluation by LRE

SYNOPSIS

`#include <CNCL/LREF.h>` Distribution Function F(x)

`#include <CNCL/LREG.h>` Complementary Distribution Function G(x)

TYPE

`CN_LREF`
`CN_LREG`

BASE CLASSES

CNStatistics

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

The `CNLREF` and `CNLREG` classes provide the statistical evaluation of random sequences by the LRE algorithm. Results are the distribution (d.f.) and the complementary distribution function (c.d.f.) respectively, the local correlation coefficient, and the error of each discovered point. The simulation run time is controlled by a predefined maximum error regarding to the local correlation of the input values. For further information refer to "Effective Control of Simulation Runs by a New Algorithm for Correlated Random Sequences" by F. Schreiber, AEUe Vol. 42, pp. 347-354, 1988.

Constructors:

```
CNLREF( CNParam * param );
```

```
CNLREF( double MIN=0.01, double MAX=0.99, double MAX_ERR=0.05, int LEVEL=100,
Scale SCALE=CNLRE::LIN, int MAXSORT= 0, char* NAME = "no name", char* TEXT = "no text";)
```

Initializes a `CNLREF` evaluation.
Parameters:

`MIN, MAX`	limits of d.f. and c.d.f respectively
`MAX_ERR`	maximum error of d.f. and d.f respectively
`LEVEL`	number of levels
`SCALE`	scale of ordinate (`CNLRE::LIN` or `CNLRE::LOG`)
`MAXSORT`	maximum size of an internal sort array
`NAME`	allows to name the evaluation.

`TEXT` a short explanation of the evaluation.

```
CNLREG( CNParam * param )
```

```
CNLREG( double MIN=0.01, double MAX=0.99, double MAX_ERR=0.05, int LEVEL=100,
Scale SCALE=LIN, int MAXSORT= 0, char* NAME = "no name", char* TEXT = "no text");
```

Initializes a `CNLREG` evaluation. The parameters are the same as described above.

In addition to the member functions required by CNCL and `CNStatistics`, `CNLREF` and `CNLREG` provide:

```
Phase status() const;
```

Returns the current state of the evaluation (`INITIALIZE`, `ITERATE` or `END`) .

4.4 CNDLRE — Evaluation by Discrete–LRE

SYNOPSIS

`#include <CNCL/DLREF.h>` Distribution Function F(x)

`#include <CNCL/DLREG.h>` Complementary Distribution Function G(x)

TYPE

`CN_DLREF`
`CN_DLREG`

BASE CLASSES

CNStatistics

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

The Discrete Limited Relative Error algorithm (DLRE or LRE III) is a special LRE for evaluating discrete random variables. The possible values must be known before the evalution is initialized, because F(x) resp. G(x) of each value will be estimated. So, the DLRE is not a better histogramm with an error measure. The implementation is a generalized version of the in ... described algorithm.

If a test value does not agree with any x–interval, the following warning is printed: `Warning: Wrong x value in DLRE[FG]::put !`.

Constructors:

```
CNDLREF( CNParam * param );
```

```
CNDLREF( double XMIN, double XMAX, double INT_SIZE, double MAX_ERR, double PRE_FIRST,
char * NAME, char * TEXT, unsigned long MAX_NRV);
```

```
CNDLREF( double * XVALUES, long LEVEL, double MAX_ERR, double PRE_FIRST, char * NAME,
char * TEXT, unsigned long MAX_NRV);
```

Initializes a `CNDLREF` evaluation. There are two types of the initializing: The first constructor (second of the list above) should be used for equidistant x–values and the socond for non–equidistant x–values.

Parameters:

`XMIN, XMAX`

Minimal and maximal x–values, whose F(x) resp. G(x) shall be estimated.

`INT_SIZE` Size of an interval of the x–axis

`XVALUES` Pointer to an array of doubles, which includes the x–values.

`LEVEL` The number of array elements.

`MAX_ERR` maximum error of d.f. and d.f respectively

`PRE_FIRST`
Predecessor of the first test value. That is necessary because of the correlation; you should not think about it too long, since a false value causes only a very small error in the measurement.

`NAME` allows to name the evaluation.

`TEXT` a short explanation of the evaluation.

`MAX_NRV` The maximal number of test values.

```
CNDLREG( CNParam * param )
```

```
CNDLREG( double XMIN, double XMAX, double INT_SIZE, double MAX_ERR, double PRE_FIRST,
char * NAME, char * TEXT, unsigned long MAX_NRV);
```

```
CNDLREG( double * XVALUES, long LEVEL, double MAX_ERR, double PRE_FIRST, char * NAME,
char * TEXT, unsigned long MAX_NRV);
```

Initializes a `CNDLREG` evaluation. The parameters are described above.

In addition to the member functions required by CNCL and `CNStatistics`, `CNDLREF` and `CNDLREG` provide:

```
Phase status() const;
```

Returns the current state of the evaluation (`INITIALIZE`, `ITERATE` or `END`).

4.5 CNBatches — Evaluation by Batch Means

SYNOPSIS

```
#include <CNCL/Batches.h>
```

TYPE

`CN_Batches`

BASE CLASSES

CNStatistics

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

The class `CNBatches` provides the statistical evaluation of random sequences by the Batch-Means method. Results are the distribution (d.f.) and complementary distribution function (c.d.f.) respectively, a histogram of relative frequencies, Bayes-error and confidence intervals of each point of the d.f and the c.d.f. respectively, and of the estimated mean and the variance of the group means.

The evaluation needs at least n * h values; n is the number of batches and h the size of the batches. The evaluation is aborted earlier if the relative Bayes error of the estimated mean is lower than a given error. For further information refer to "Principles of Discrete Event Simulation" by G.S. Fishman, J. Wiley & Sons, New York, 1978.

Constructors:

```
CNBatches();
```

```
CNBatches( CNParam *param );
```

```
CNBatches( double BOTTOM=0.0, double TOP=100.0, double MAX_ERR=0.2, long NOG=10,
long SOG=1000, long NOI=10, short CONF=95, char *NAME="no name",
char *TEXT="no text" );
```

Initializes a `CNBatches` evaluation.The Parameters are:

`BOTTOM`
`TOP` lower resp. upper limit of evaluated values; values beyond these limits are only counted;

`MAX_ERR` defined maximum error

`NOG` number of groups (batches)

`SOG`	size of one group (batch)
`NOI`	number of intervals
	WARNING: if using more than one `CNBatches` evaluations, all objects must be initialized with the same `NOI`!!!
`NAME`	
`TEXT`	allows to name and to explain the evaluation.

In addition to the member functions required by CNCL and `CNStatistics`, `CNBatches` provides:

`virtual Phase status() const;`

Returns the current state of evaluation (`INITIALIZE`, `ITERATE` or `END`).

`double bayes_err();`

Returns the relative Bayes error of the mean.

`double sigma();`

Returns the deviation of the group means; can also be used as an error measure of the mean, e.g. its relative Bayes error.

`double rd_conf();`

Returns the confidence interval of the mean.

`virtual void print( Type type = CNStatistics::DF, ostream &strm = cout ) const;`

The first argument determines between the output of d.f (`CNStatistics::DF`, default) and c.d.f. (`CNStatistics::CDF`), the second is an user defined output stream.

5 Container Classes

CNCL's hierarchical concept and the runtime type information make it possible to provide generic containers, i.e. container that can contain any object. CNCL generic containers classes work with pointers to `CNObject`, a class with which all classes in the CNCL hierarchy are type-compatible.

5.1 CNAVLTree — AVL balanced binary search tree

SYNOPSIS

```
#include <CNCL/AVLTree.h>
```

TYPE

`CN_CNAVLTREE`

BASE CLASSES

CNObject

DERIVED CLASSES

none

RELATED CLASSES

CNAVLNode

DESCRIPTION

`CNAVLTree` is realizing a generic AVL tree. It contains nodes derived from `CNAVLNode` and organizes them in a balanced binary search tree. Such Nodes can be added to the tree, searched for and be removed.

`CNAVLTree` is an abstract base class. It provides the general algorithms needed for AVL trees, but it makes no assumptions on what kind of key the nodes will be sorted by. Usable AVL trees require derived classes that specify the key that will be used.

Constructors:

```
CNAVLTree();
CNAVLTree(CNParam *param);
```

Initializes an empty AVL tree.

Destructor:

```
~CNAVLTree();
```

Deletes the tree and all `CNAVLNode` nodes still in the tree.

In addition to the member functions required by CNCL, `CNAVLTree` provides:

```
bool add(CNAVLNode*);
```

Adds a node to the tree. If the key the node is sorted by is already existing in the current AVL tree, `FALSE` is returned. otherwise `TRUE` is returned.

`CNAVLNode *find();`

Searches for a key. This is an abstract functions that may only be called by subclasses of `CNAVLTree`. The subclasses have to provide new `find()` functions that manage the key to search for.

Returns a pointer to the found `CNAVLNode` on success and `NIL` on failure (key not found).

`CNAVLNode *remove();`

Similar to `find()`, but the returned node is also removed from the tree.

`bool empty();`

Returns `TRUE` if the tree is empty.

`void delete_all();`

Deletes all nodes still in the tree.

`CNAVLNode *find_first();`

Returns a pointer to the first node in the tree, i.e. the node with the lowest key. Returns `NIL` if the tree is empty.

`CNAVLNode *remove_first();`

Similar to `find_first()`, but also removes the returned node.

`CNAVLNode *get_root();`

Returns the root node of the tree or `NIL` is the tree is empty.

`unsigned long length() const;`

Returns the number of nodes in the tree.

The following example shows parts of a derived class. It uses simple `long` keys to sort and find nodes. See also the example for `CNAVLNode`.

```
class IntAVLTree : public CNAVLTree
{
    friend class IntAVLNode;

  // ...

  public:  /***** Public interface ************************************/

    virtual CNAVLNode *find(long key);
    virtual CNAVLNode *remove(long key);

  private:  /***** Internal private members *****************************/

    long searchkey;

  // ...
};

// ...

CNAVLNode *IntAVLTree::find(long key) {
    searchkey = key;
    return CNAVLTree::find();
};
```

```
CNAVLNode *IntAVLTree::remove(long key) {
    searchkey = key;
    return CNAVLTree::remove();
};
```

5.2 CNAVLNode — Node for CNAVLTree

SYNOPSIS

```
#include <CNCL/AVLNode.h>
```

TYPE

`CN_AVLNODE`

BASE CLASSES

CNObject

DERIVED CLASSES

none

RELATED CLASSES

CNAVLTree

DESCRIPTION

`CNAVLNode` is an abstract base class for nodes of AVL trees.

Derived classes must be implemented for different key types. They must be able to compare their keys between two node and between a node and the serached key of the tree.

Constructors:

```
CNAVLNode();
CNAVLNode(CNParam *param);
```

Initializes a new AVL node.

In addition to the member functions required by CNCL, `CNAVLNode` provides:

```
virtual int compare(CNAVLNode*) = 0;
```

Compares the node's key with another node's key. Returns `-1` if the current node's key is lower, `1` of it's higher and `0` if it's equal to the other node's key.

```
virtual int find(CNAVLTree*) = 0;
```

Compares the node's key with the current search key. Returns `-1` if the current node's key is lower, `1` of it's higher and `0` if it's equal to the searched key.

Derived tree classes must provide a possibility for nodes to get the currently searched tree key, e.g. as a member variable in the tree structure.

```
CNAVLNode *left();
```

Returns the root of the left sub-tree (lower keys) or `NIL` if there is no left sub-tree.

```
CNAVLNode *right();
```

Returns the root of the right sub-tree (higher keys) or `NIL` if there is no right sub-tree.

The following example shows parts of a derived class. It uses simple `long` keys to sort and find nodes. See also the example for `CNAVLTree`.

```
class IntAVLNode : public CNAVLNode
{
  public:  /***** Constructors ****************************************/

    IntAVLNode(long key): key_(key) {};

  public:  /***** Public interface ************************************/

    virtual int compare(CNAVLNode*); // compare node with another one
    virtual int find(CNAVLTree*);    // compare node with searched key
    long get_key() { return key_; };

  private:  /***** Internal private members *****************************/

    long key_;

  // ...

};

// ...

int IntAVLNode::compare(CNAVLNode *n) {
    long c = IntAVLNode::cast_from_object(n)->key_;
    if (key_ < c) return -1;
    if (key_ > c) return  1;
    return 0;
};

int IntAVLNode::find(CNAVLTree *t) {
    long c = IntAVLTree::cast_from_object(t)->searchkey;
    if (key_ < c) return -1;
    if (key_ > c) return  1;
    return 0;
};
```

5.3 CNDLList — Doubly Linked List of Objects

SYNOPSIS

```
#include <CNCL/DLList.h>
```

TYPE

`CN_DLLIST`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNDLObject, CNDLIterator

DESCRIPTION

`CNDLList` is a doubly linked list that can contain any CNCL compatible object.

Constructors:

```
CNDLList();
CNDLList(CNParam *param);
```

Initializes the list to empty state.

Destructors:

```
~CNDLList();
```

Deletes the linked list and all `CNDLObject` nodes. It does NOT delete the objects referenced by the nodes.

In addition to the member functions required by CNCL, `CNDLList` provides:

```
CNDLObject *first() const;
```

Returns the first node in the list or `NIL` if the list is empty.

```
CNDLObject *last() const;
```

Returns the last node in the list or `NIL` if the list is empty.

```
CNDLObject *next(CNDLObject *link) const;
```

Returns the next node in the list, where `link` points to the current node. This may be `NIL` if the next node doesn't exist.

```
CNDLObject *prev(CNDLObject *link) const;
```

Returns the previous node in the list, where `link` points to the current node. This may be `NIL` if the previous node doesn't exist.

`bool empty() const;`

Checks if `DLList` is empty.

`unsigned long length() const;`

Returns the length (number of nodes) of this `DLList`.

`CNDLObject *append(CNObject *obj);`

`CNDLObject *append(CNObject &obj);`

Adds a new node to the end of the list, referencing the object `obj`. It returns the node allocated for the object.

`CNDLObject *append(CNDLObject *obj);`

Adds an already allocated node to the list. It returns the pointer `obj`.

`CNDLObject *prepend(CNObject *obj);`

`CNDLObject *prepend(CNObject &obj);`

Adds a new node to the start of the list, referencing the object `obj`. It returns the node allocated for the object.

`CNDLObject *prepend(CNDLObject *obj);`

Adds an already allocated node to the list. It returns the pointer `obj`.

`CNDLObject *delete_object(CNDLObject *pos);`

Deletes the node `pos` from the list. It returns the next node. This function deletes the node (`CNDLObject`) from the list, but it does NOT delete the object referenced by the node.

`CNDLObject *remove_object(CNDLObject *pos);`

Removes the node `pos` from the list. It returns the next node. This function does NOT delete the linked list node (`CNDLObject`) and it does NOT delete the object referenced by the node, either.

`void delete_all();`

Deletes all nodes from the linked list and initializes the list to empty. The objects referenced by the nodes are NOT deleted.

`void delete_all_w_obj();`

Deletes all nodes from the linked list, initializes the list to empty, AND deletes the referenced objects.

`CNDLObject *insert_before(CNDLObject *pos, CNObject *obj);`

`CNDLObject *insert_before(CNDLObject *pos, CNObject &obj);`

Creates a new node for `obj` and inserts it into the list before node `pos`. It returns the new node.

`CNDLObject *insert_before(CNDLObject *pos, CNDLObject *obj);`

Inserts an already allocated node into the list before node `pos`. It returns `obj`.

`CNDLObject *insert_after(CNDLObject *pos, CNObject *obj);`

`CNDLObject *insert_after(CNDLObject *pos, CNObject &obj);`

Creates a new node for `obj` and inserts it into the list after node `pos`. It returns the new node.

`CNDLObject *insert_after(CNDLObject *pos, CNDLObject *obj);`

Inserts an already allocated node into the list after node `pos`. It returns `obj`.

`bool ok();`

Checks the list for consistency. It returns `TRUE`, if the list is o.k.

5.4 CNDLObject — Node of Doubly Linked List

SYNOPSIS

```
#include <CNCL/DLObject.h>
```

TYPE

`CN_DLOBJECT`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNDLList, CNDLIterator

DESCRIPTION

`CNDLObject` is a node in the `CNDLList` doubly linked list. It contains a pointer to the next and previous node, plus a pointer to the referenced object.

Constructors:

```
CNDLObject();
CNDLObject(CNParam *param);
CNDLObject(CNObject *obj);
```

Initializes `CNDLObject` and optionally sets a referenced object.

```
CNDLObject(const CNDLObject &link);
```

Will issue a fatal error message. `CNDLObject`s cannot be copied!

`CNDLObject`s have a private destructor and can therefore only be allocated on the heap. Furthermore `CNDLObject`s cannot be copied; an attempt do so results in a runtime error.

In addition to the member functions required by CNCL, `CNDLObject` provides:

```
CNDLObject *next(CNDLObject *p);
CNDLObject *next();
CNDLObject *set_next(CNDLObject *p);
CNDLObject *get_next();
```

Gets/sets the pointer to the next node. It returns the current/old value.

```
CNDLObject *prev(CNDLObject *p);
CNDLObject *prev();
```

```
CNDLObject *set_prev(CNDLObject *p);
CNDLObject *get_prev();
```

Gets/sets the pointer to the previous node. It returns the current/old value.

```
CNObject *object(CNObject *obj);
CNObject *object();
CNObject *set_object(CNObject *obj);
CNObject *get_object();
```

Gets/sets the pointer to the referenced object. It returns the current/old value.

```
void delete_object();
```

Deletes the referenced object.

5.5 CNDLIterator — Iterator of Doubly Linked List

SYNOPSIS

```
#include <CNCL/DLIterator.h>
```

TYPE

`CN_DLITERATOR`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNDLList, CNDLObject

DESCRIPTION

`CNDLIterator` is an iterator to traverse a `CNDLList` doubly linked list.

Constructors:

```
CNDLIterator();
CNDLIterator(CNParam *param);
```

Initializes `CNDLIterator`.

```
CNDLIterator(const CNDLList *new_list);
CNDLIterator(const CNDLList &new_list);
```

Initializes `CNDLIterator` with linked list `list`. The iterator is reset to the first element in the list.

In addition to the member functions required by CNCL, `CNDLIterator` provides:

```
void reset(const CNDLList *new_list);
void reset(const CNDLList &new_list);
void reset();
```

Resets the iterator to a new list `new_list` and/or sets the iterator to the first element in the list.

```
CNObject *object()
CNObject *get_object()
```

Gets the referenced object from the current iterator position. It returns the object or `NIL`, if none is available.

```
CNDLObject *position()
```

`CNDLObject *get_position()`

Gets the current iterator position (node in the list). It returns a pointer to the node or `NIL`, if none is available.

`CNObject *first_object();`

`CNObject *first();`

Sets the iterator to the first element in the list. It returns the referenced object or `NIL`, if none is available.

`CNObject *last_object();`

`CNObject *last();`

Sets the iterator to the last element in the list. It returns the referenced object or `NIL`, if none is available.

`CNObject *next_object();`

`CNObject *next();`

`CNObject *operator ++();`

`CNObject *operator ++(int);`

Moves the iterator to the next element in the list. It returns the current referenced object (the one BEFORE moving the iterator) or `NIL`, if none is available.

`CNObject *prev_object();`

`CNObject *prev();`

`CNObject *operator --();`

`CNObject *operator --(int);`

Moves the iterator to the previous element in the list. It returns the current referenced object (the one BEFORE moving the iterator) or `NIL`, if none is available.

The following examples show how to use a `CNDLIterator` object to traverse a linked list:

Forward:

```
CNDLList list;

...

CNDLIterator trav(list);
CNObject *obj;

while(obj = trav++)
{
    // Do something with obj ...
}
```

Alternate forward:

```
for(trav.reset(list); obj=trav.object(); trav.next())
{
    // ...
}
```

Backward:

```
for(trav.last(); obj=trav.object(); trav--)
{
    // ...
}
```

5.6 CNQueue — Abstract Queue Base Class

SYNOPSIS

```
#include <CNCL/Queue.h>
```

TYPE

`CN_QUEUE`

BASE CLASSES

CNObject

DERIVED CLASSES

CNQueueFIFO, CNQueueLIFO, CNStack, ...

RELATED CLASSES

none

DESCRIPTION

`CNQueue` is a queue of any CNCL compatible object.

Constructors:

```
CNQueue();
CNQueue(CNParam *param);
```

Initialize the queue.

In addition to the member functions required by CNCL, `CNQueue` provides the following abstract member function, which must be implemented by the derived classes:

```
virtual bool empty() const = 0;
```

Returns `TRUE`, if the queue is empty.

```
virtual bool full() const = 0;
```

Returns `TRUE`, if the queue is full.

```
virtual int length() const = 0;
```

Returns the actual queue length.

```
virtual void put(CNObject *obj) = 0;
void put(CNObject &obj);
```

Puts an object into the queue.

```
virtual CNObject *get() = 0;
```

Retrieves an object from the queue.

```
virtual void delete_all() = 0;
```

Deletes all objects in the queue.

5.7 CNQueueFIFO — FIFO Queue

SYNOPSIS

```
#include<CNCL/QueueFIFO.h>
```

TYPE

`CN_QUEUEFIFO`

BASE CLASSES

CNQueue, CNDLList

DERIVED CLASSES

none

RELATED CLASSES

none

DESCIPTION

`CNQueueFIFO` is a queue, implemented as a doubly linked list, that can contain any number (well, sort of ... ;-) CNCL compatible objects. The queueing strategy is FIFO (First In, First Out).

Constructors:

```
CNQueueFIFO();
CNQueueFIFO(CNParam *param);
```

Initialize the FIFO-queue to an empty state.

In addition to the member functions required by CNCL, `CNQueueFIFO` provides:

```
virtual bool empty() const;
```

Returns `TRUE`, if the queue is empty.

```
virtual bool full() const;
```

Always returns `TRUE`.

```
virtual int length() const;
```

Returns the actual queue length.

```
virtual void put(CNObject *obj);
```

Puts an object into the queue.

```
virtual CNObject *get();
```

Retrieves an object from the queue.

```
virtual void delete_all();
```

Deletes all objects in the queue.

5.8 CNQueueLIFO — LIFO queue

SYNOPSIS

```
#include <CNCL/QueueLIFO.h>
```

TYPE

`CN_QUEUELIFO`

BASE CLASSES

CNQueue

DERIVED CLASSES

None

RELATED CLASSES

CNQueueFIFO, CNStack

DESCRIPTION

`CNQueueLIFO` is a queue, implemented as a doubly linked list, that can contain any number (well, sort of ... ;-) CNCL compatible objects. The queueing strategy is LIFO (Last In, First Out).

Constructors:

```
CNQueueLIFO();
CNQueueLIFO(CNParam *param);
```

Initialize the LIFO-queue to an empty state.

In addition to the member functions required by CNCL, `CNQueueLIFO` provides:

```
virtual bool empty() const;
```

Returns `TRUE`, if the queue is empty.

```
virtual bool full() const;
```

Always returns `FALSE`.

```
virtual int length() const;
```

Returns the actual queue length.

```
virtual void put(CNObject *obj);
```

Puts an object into the queue.

```
virtual CNObject *get();
```

Retrieves an object from the queue.

```
virtual void delete_all();
```

Deletes all objects in the queue.

5.9 CNQueueRandom — Random queue

SYNOPSIS

```
#include <CNCL/QueueRandom.h>
```

TYPE

`CN_QUEUERANDOM`

BASE CLASSES

CNQueue

DERIVED CLASSES

None

RELATED CLASSES

CNQueueFIFO, CNQueueLIFO

DESCRIPTION

`CNQueueRandom` is a queue, implemented as a doubly linked list, that can contain any number (well, sort of ... ;-) CNCL compatible objects. The queueing strategy is Random, that means there is no ordering in the queue.

Constructors:

```
CNQueueRandom();
CNQueueRandom(CNParam *param);
CNQueueRandom(CNRNG *rng);
```

Initialize the Random-queue to an empty state. One may provide a base random number generator (recommended!), otherwise the queue provides its own.

In addition to the member functions required by CNCL, `CNQueueRandom` provides:

```
virtual bool empty() const;
```

Returns `TRUE`, if the queue is empty.

```
virtual bool full() const;
```

Always returns `FALSE`.

```
virtual int length() const;
```

Returns the actual queue length.

```
virtual void put(CNObject *obj);
```

Puts an object into the queue.

```
virtual CNObject *get();
```

Retrieves a random object from the queue.

`virtual void delete_all();`

Deletes all objects in the queue.

5.10 CNQueueSPT — SPT queue

SYNOPSIS

```
#include <CNCL/QueueSPT.h>
```

TYPE

`CN_QUEUESPT`

BASE CLASSES

CNQueue

DERIVED CLASSES

None

RELATED CLASSES

CNQueueFIFO, CNJob

DESCRIPTION

`CNQueueSPT` is a queue, implemented as a doubly linked list, that can contain any number (well, sort of ... ;-) CNCL compatible (restrictions see below) objects. The queueing strategy is SPT, that means jobs with the shortest processing time are delivered first. The processing time is read from the public identifier `length` of the `CNJob` object. SPT queues only accept objects derived from `CNJob`.

Constructors:

```
CNQueueSPT();
CNQueueSPT(CNParam *param);
```

Initialize the SPT-queue to an empty state.

In addition to the member functions required by CNCL, `CNQueueSPT` provides:

```
virtual bool empty() const;
```

Returns `TRUE`, if the queue is empty.

```
virtual bool full() const;
```

Always returns `FALSE`.

```
virtual int length() const;
```

Returns the actual queue length.

```
virtual void put(CNObject *obj);
```

Puts an object into the queue. Only objects derived from `CNJob` may be put into a SPT queue.

`virtual CNObject *get();`

Retrieves the object with the shortest processing time from the queue.

`virtual void delete_all();`

Deletes all objects in the queue.

5.11 CNPrioQueueFIFO — Queue with priority

SYNOPSIS

```
#include <CNCL/PrioQueueFIFO.h>
```

TYPE

`CN_PRIOQUEUEFIFO`

BASE CLASSES

CNQueue

DERIVED CLASSES

None

RELATED CLASSES

CNQueueFIFO, CNJob

DESCRIPTION

`CNPrioQueueFIFO` is a queue, implemented as some FIFO queues, that can contain any number (well, sort of ... ;-) CNCL compatible (restrictions see below) objects. A priority queue consists of a number of simple FIFO queues, one for each priority. The priority value (0 to ...) is taken from the public identifier `priority` of the `CNJob` object. Lower values mean higher priority in access. Only objects derived from `CNJob` are accepted.

Constructors:

```
CNPrioQueueFIFO();
CNPrioQueueFIFO(CNParam *param);
CNPrioQueueFIFO(int prios);
```

Initialize the priority-queue to an empty state. The `prios` value determines the number of different priority steps for. By default `prios` is set to two.

In addition to the member functions required by CNCL, `CNPrioQueueFIFO` provides:

```
virtual bool empty() const;
```

Returns `TRUE`, if all internal queues are empty.

```
virtual bool full() const;
```

Always returns `FALSE`.

```
virtual int length() const;
```

Returns the actual queue length, that means the sum of all internal queues.

```
virtual void put(CNObject *obj);
```

Puts an object into the queue. Only objects derived from `CNJob` may be put into a priority queue.

`virtual CNObject *get();`

Retrieves an object from the queue. If there are objects in more than one internal queue the object with the lowest value in `priority` is retrieved.

`virtual void delete_all();`

Deletes all objects in the queue.

5.12 CNJob — Job object

SYNOPSIS

```
#include <CNCL/Job.h>
```

TYPE

`CN_JOB`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNQueueSPT CNPrioQueueFIFO

DESCRIPTION

`CNJob` is a standard job object for the CNCL queues. FIFO, LIFO and Random Queus *can* contain `CNJob`, whereas for SPT and Priority Queues *must* contain it. Users may derive objects from `CNJob` if they need additional data. `CNJob` provides the following public member variables which can be set/read by any user:

```
CNSimTime in;
CNSimTime out;
CNSimTime start;
double orig_length;
double length;
int priority;
```

The three time variables are used to hold the time when a job enters a system, when a job leaves a system and when the serving begins. `length` defines the remaining service time of a job while `orig_length` defines its whole service time. They are equal in case of noninterrupting queueing strategies. `priority` identifies the priority of a job used in priority queues.

Constructors:

```
CNJob();
CNJob(CNParam *param);
CNJob(double len);
CNJob(int prio);
CNJob(int prio, double len);
```

Initialize the Job object, optionally setting length and/or priority.

`CNJob` provides no additional member functions besides the one required by CNCL.

5.13 CNSink — Kitchen Sink

SYNOPSIS

```
#include <CNCL/Sink.h.h>
```

TYPE

`CN_SINK`

BASE CLASSES

CNQueue

DERIVED CLASSES

None

RELATED CLASSES

CNQueueFIFO

DESCRIPTION

`CNSink` is a data sink, implemented as queue with only an input side. All objects which are put into this sink are destroyed there. There is no queueing strategy.

Constructors:

```
CNSink();
CNSink(CNParam *param);
```

Initialize the Sink.

In addition to the member functions required by CNCL, `CNSink` provides:

```
virtual bool empty() const;
```

Always returns `TRUE`.

```
virtual bool full() const;
```

Always returns `FALSE`.

```
virtual int length() const;
```

Returns the number of objects deleted in the queue.

```
virtual void put(CNObject *obj);
```

Puts an object into the queue. It will be deleted there.

```
virtual CNObject *get();
```

Produces an error. A sink is like a black hole.

```
virtual void delete_all();
```

Sets the current sink to its initial state, the number of deleted objects is set to zero.

5.14 CNStack — Stack

SYNOPSIS

```
#include <CNCL/Stack.h>
```

TYPE

`CN_STACK`

BASE CLASSES

CNQueue

DERIVED CLASSES

None

RELATED CLASSES

CNQueueLIFO

DESCRIPTION

`CNStack` is a stack, implemented as a LIFO queue which can hold any number of CNCL compatible objects. It acts very much like a LIFO queue but provides another kind of user interface.

Constructors:

```
CNStack();
CNStack(CNParam *param);
CNStack(long elem);
```

Initialize the Stack, optionally setting the size.

In addition to the member functions required by CNCL, `CNStack` provides:

```
bool empty() const;
```

Returns `TRUE` when stack is empty.

```
int depth() const;
```

Returns the number of objects on the stack.

```
void push(CNObject *obj);
```

Pushes an object onto the stack.

```
CNObject *pull();
```

Retrieves the last object from the stack.

```
void clear();
```

Deletes all objects on the stack.

```
long size();
void size(long num);
```

Gets/sets the maximum depth of the stack.

6 Event Driven Simulation

CNCL includes a set of classes for performing event driven simulation.

For programmers convinience a script `sim.h` has been added to CNCL, so that all necessary simulation tools are available by typing `#include <sim.h>`. By default the heap scheduler is used, but if the special abilities of `CNEventScheduler` are needed, your program only needs to define `#define NO_HEAP_SCHEDULER`.

6.1 CNEvent — Generic Event

SYNOPSIS

```
#include <CNCL/Event.h>
```

TYPE

`CN_EVENT`

BASE CLASSES

CNDLObject

DERIVED CLASSES

None

RELATED CLASSES

CNEventHandler, CNEventScheduler, CNSimTime

DESCRIPTION

`CNEvent` is a data type for events used in the simulation. Control is passed between different event handlers by sending events to each other. Events have a unique ID field, a priority, a user defined type, an issued and scheduled simulation time, an addressed event handler, a sending event handler, and a pointer to an arbitrary CNCL object.

`CNEvent`s can only be allocated on the heap via `new`.

Constructors:

```
CNEvent();
CNEvent(CNParam *param);
CNEvent(int new_type);
CNEvent(int new_type, const CNSimTime t, int prio=0);
CNEvent(int new_type, CNEventHandler *new_to, const CNSimTime t, CNObject *new_object=NIL,
int new_prio=0);
CNEvent(int new_type, CNEventHandler *new_from, CNEventHandler *new_to, const CNSimTime t,
CNObject *new_object=NIL, int new_prio=0);
CNEvent(int new_type, CNEventHandler *new_to, CNObject *new_object=NIL, int new_prio=0);
```

Initializes the event, optionally setting the event type, the addressed and sending event handler, the scheduled time, the referenced object, and the priority.

Destructors:

`~CNEvent();`

Deletes the event. The destructor is private and may only be called from `CNEvent` itself or the friend class `CNEventScheduler`. The destructor does NOT delete the referenced object.

In addition to the member functions required by CNCL, `CNEvent` provides:

```
static void set_max_events(unsigned long n);
static unsigned long get_max_events();
```

Gets/Sets maximum number of events. The default value for `max-event` is 100. If `set_max_events()` changes this value, it MUST be called before ANY event is created.

If more than `max-event` events are allocated, a `warning` message is printed and the simulation continues with `max-event` multiplied by 10.

```
int priority() const;
void priority(int prio);
int get_priority() const;
void set_priority(int prio);
```

Gets/sets the event's priority.

```
int type() const;
void type(int new_type);
int get_type() const;
void set_type(int new_type);
```

Gets/sets the event's type.

```
CNSimTime scheduled() const;
void scheduled(const CNSimTime new_scheduled);
CNSimTime get_scheduled() const;
void set_scheduled(const CNSimTime new_scheduled);
```

Gets/sets the event's scheduled simulation time.

```
CNSimTime issued() const;
CNSimTime get_issued() const;
```

Gets the event's issued simulation time.

```
CNEventHandler *to() const;
void to(CNEventHandler *new_to);
CNEventHandler *get_to() const;
void set_to(CNEventHandler *new_to);
```

Gets/sets the event's addressed event handler.

```
CNEventHandler *from() const;
void from(CNEventHandler *new_from);
CNEventHandler *get_from() const;
void set_from(CNEventHandler *new_from);
```

Gets/sets the event's sending event handler.

`CNObject *object() const;`

`void object(CNObject *obj);`

`CNObject *get_object() const;`

`void set_object(CNObject *obj);`

Gets/sets the event's referenced object.

`typedef unsigned long CNEventID;`

`CNEventID id() const;`

`CNEventID get_id() const;`

Returns the event's unique ID number.

`static void *operator new(size_t s);`

The `new` operator. It is allocating an event out of a pool

`static void operator delete(void *p);`

The `delete` operator.

`bool after(CNEvent e2);`

Returns `TRUE` if `*this` Event is scheduled after Event `e2`. If both Events are scheduled at the same time, the priority is checked; after that the Event ID. This method is used by the class `CNEventHeapSched`.

6.2 CNEventExploder — Send Events to multiple EventHandlers

SYNOPSIS

```
#include <CNCL/EventExploder.h>
```

TYPE

`CN_EVENTEXPLODER`

BASE CLASSES

CNEventHandler

DERIVED CLASSES

None

RELATED CLASSES

CNEvent, CNEventHandler, CNEventScheduler, CNSimTime

DESCRIPTION

`CNEventExploder` is a subclass derived from `CNEventHandler`.
It maintains a list of other arbitrary `CNEventHandlers` and forwards all received events to all the `CNEventHandlers` in its list. This can be used to implement "broadcast"-events.

Constructors:

`CNEventExploder();`

`CNEventExploder(CNParam *param);`

Initializes the event exploder.

In addition to the member functions required by CNCL, `CNEventExploder` provides the following member functions:

`virtual void event_handler(const CNEvent *ev);`

This function receives an event and re-sends it to other `CNEventHandlers`.

`virtual void add_handler(CNEventHandler *eh);`

This function adds a new `CNEventHandler` to the list. It will then receive all events that are send to the `CNEventExploder`.

If a `CNEventHandler` is added more than once, it will also receive all events as many times.

`virtual void rem_handler(CNEventHandler *eh);`

This function removes a previously added `CNEventHandler` from the list. It will no longer receive events managed by the `CNEventExploder`.

If a `CNEventHandler` has been added more than once, it also has to be removed as many times to be completely removed from the `CNEventExploder`.

6.3 CNEventHandler — Abstract Base Class for Event Handlers

SYNOPSIS

```
#include <CNCL/EventHandler.h>
```

TYPE

`CN_EVENTHANDLER`

BASE CLASSES

CNObject

DERIVED CLASSES

CNEventExploder

RELATED CLASSES

CNEvent, CNEventScheduler, CNSimTime

DESCRIPTION

`CNEventHandler` is a base class for creating simulation event handler. Deriving an event handler class from `CNEventHandler` is necessary for the simulation programm.

Each derived event handler must at least provide the function `event_handler()`, which is defined as pure virtual in class `CNEventHandler`.

Constructors:

`CNEventHandler();`

`CNEventHandler(CNParam *param);`

Initializes the event handler.

`CNEventHandler(CNStringR name);`

Initializes the event handler, setting the name to `name`.

In addition to the member functions required by CNCL, `CNEventHandler` provides the following *protected* member functions available to all derived classes:

`virtual void event_handler(const CNEvent *ev) = 0;`

This function *must* be defined in the derived classes. It should determine what shoulc happen to the transfered Events. Please see the example at the end of this chapter.

`CNEventScheduler *scheduler() const;`

Returns the scheduler, which called this event handler.

`int state() const;`

`int state(int new_state);`

`int get_state() const;`

`int set_state(int new_state);`

Gets/sets the event handler's state.

`CNSimTime now() const;`

Returns the current simulation time.

`CNEventID send_event(CNEvent *ev);`

`CNEventID send(CNEvent *ev);`

Send an event to the scheduler. Members left uninitialized in the event are set to default values (scheduled time = current time, addressed event handler = this event handler, sending event handler = this event handler). Returns the event's ID.

`CNEventID send_now(CNEvent *ev);`

Send an event to the scheduler. Same as `send_event()`, but scheduled time is set to the current time. Returns the event's ID.

`CNEventID send_delay(CNEvent *ev, double dt);`

Send an event to the scheduler. Same as `send_event()`, but scheduled time is set to the current time plus a time delay dt. Returns the event's ID.

`void delete_event(CNEventID id);`

Deletes event with ID `id` from the scheduler's event list.

`void delete_events(CNEventHandler *evh);`

Deletes all events from the list that are addressed to event handler `evh`. If `evh` is `NIL`, it deletes all events addressed to this event handler.

6.4 CNEventList — List of Events

SYNOPSIS

```
#include <CNCL/EventList.h>
```

TYPE

`CN_EVENTLIST`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNEvent, CNEventScheduler

DESCRIPTION

`CNEventList` is the list of events managed by `CNEventScheduler`. It is a wrapper around `CNDLList` for creating a list of sorted events. CNEvents are sorted by their scheduled time and their priority.

Constructors:

```
CNEventList();
CNEventList(CNParam *param);
```

Initializes the event list to empty.

In addition to the member functions required by CNCL, `CNEventList` provides:

```
void add_event(CNEvent *ev);
```

Adds an event to the list.

```
void delete_event(CNEventID id);
```

Deletes an event from the list, using the ID code.

```
void delete_events(CNEventHandler *evh);
```

Deletes all events from the list that are addressed to event handler `evh`.

```
void delete_all(CNEventID id);
```

Deletes all events from the list.

```
CNEvent *next_event();
```

Gets the next event (the one at the front) from the list.

```
CNEvent *peek_event();
```

`CNEvent *peek_event(CNEventID id);`

Returns a pointer to the next scheduled event or a pointer to the event with ID `id`, `NIL` if not available.

6.5 CNEventBaseSched — Abstract scheduler base class

SYNOPSIS

```
#include <CNCL/EventBaseSched.h>
```

TYPE

`CN_EVENTBASESCHED`

BASE CLASSES

CNObjedt

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

Constructors:

`CNEventBaseSched();`

`CNEventBaseSched(CNParam *);`

Initializes the `CNEventBaseSched`.

In addition to the member functions required by CNCL, `CNEventBaseSched` provides:

`CNSimTime::time();`

Returns the simulation time.

`CNStatistics *statistics() const;`

`void statistics(CNStatistics *st);`

Sets/Gets the (optional) scheduler statistic. By default no statistic is used.

The following functions are defined virtual and *must* be defined in derived classes:

`void add_event(CNEvent *ev);`

`void send_event(CNEvent *ev);`

Adds/sends an event to an event handler.

`void delete_event(CNEventID id);`

Deletes event with ID `id`.

`void delete_events(CNEventHandler *evh);`

Deletes all events from the list that are addressed to event handler `evh`.

`CNEvent *peek_event();`

`CNEvent *peek_event(CNEventID id);`

Peeks at next event or event with ID `id`. Returns pointer to event, or `NIL` if not available.

`void start();`

`void start(CNEvent *ev);`

Starts the scheduler with an optional initialization event.

`void stop();`

Stops the scheduler after processing the current event and deletes all pending events in the event list. May be used inside an event handler to stop the simulation.

6.6 CNEventScheduler — Event Scheduler

SYNOPSIS

```
#include <CNCL/EventScheduler.h>
```

TYPE

`CN_EVENTSCHEDULER`

BASE CLASSES

CNEventBaseSched

DERIVED CLASSES

None

RELATED CLASSES

CNEvent, CNEventList, CNEventHandler, CNEventHeapSched

DESCRIPTION

`CNEventScheduler` is the central simulation control. It manages the events and delivers them to the addressed event handlers.

Scheduled events are sorted with respect to their scheduled simulation time and their priority. If there are two or more events with exactly the same scheduled simulation time and the same priority, they are processed in FIFO order.

If this exact behaviour is not strictly required, if scheduled simulation time and priority are sufficient to determine the procession order of events, then CNEventHeapSched should be used instead of CNEventScheduler. CNEventHeapSched avoids some potential performace deficiencies CNEventScheduler might show.

Constructors:

```
CNEventScheduler();
CNEventScheduler(CNParam *param);
```

Initializes the event scheduler.

In addition to the member functions required by CNCL, `CNEventScheduler` provides:

```
void add_event(CNEvent *ev);
void send_event(CNEvent *ev);
```

Adds/sends an event to an event handler.

```
void delete_event(CNEventID id);
```

Deletes event with ID `id`.

`void delete_events(CNEventHandler *evh);`

Deletes all events from the list that are addressed to event handler `evh`.

`CNEvent *peek_event();`

`CNEvent *peek_event(CNEventID id);`

Peeks at next event or event with ID `id`. Returns pointer to event, or `NIL` if not available.

`void start();`

`void start(CNEvent *ev);`

Starts the scheduler with an optional initialization event.

`void stop();`

Stops the scheduler after processing the current event and deletes all pending events in the event list. May be used inside an event handler to stop the simulation.

`CNSimTime time();`

Returns the scheduler's current simulation time.

`CNEventIterator *create_iterator();`

Creates an iterator object to traverse the list of events.

6.7 CNEventHeapSched — Event Scheduler using a heap

SYNOPSIS

```
#include <CNCL/EventHeapSched.h>
```

TYPE

`CN_EVENTHEAPSCHED`

BASE CLASSES

CNEventBaseSched

DERIVED CLASSES

None

RELATED CLASSES

CNEvent, CNEventHandler, CNEventScheduler

DESCRIPTION

`CNEventHeapSched` is a replacement for `CNEventScheduler`. From the user's point of view, it is completely compatible, but it differs in the internally used datastructures and algorithms.

If a high number of events is used simultaneously, the eventlist used in `CNEventScheduler` can become very slow. `CNEventHeapSched` solves that problem by using a more efficient algorithm, a "heap".

However there is one drawback with `CNEventHeapSched`: in contrast to `CNEventScheduler` no FIFO order of processing can be guaranteed if events compare equal. If there are for example two or more events with exactly the same scheduled simulation time and the same priority, then they are processed in random order.

Constructors:

```
CNEventHeapSched();
CNEventHeapSched(CNParam *param);
```

Initializes the event scheduler.

In addition to the member functions required by CNCL, `CNEventHeapSched` provides:

```
void add_event(CNEvent *ev);
void send_event(CNEvent *ev);
```

Adds/sends an event to an event handler.

```
void delete_event(CNEventID id);
```

Deletes event with ID `id`.

`void delete_events(CNEventHandler *evh);`

Deletes all events from the list that are addressed to event handler `evh`.

`CNEvent *peek_event();`

`CNEvent *peek_event(CNEventID id);`

Peeks at next event or event with ID `id`. Returns pointer to event, or `NIL` if not available.

`void start();`

`void start(CNEvent *ev);`

Starts the scheduler with an optional initialization event.

`void stop();`

Stops the scheduler after processing the current event and deletes all pending events in the event list. May be used inside an event handler to stop the simulation.

`CNSimTime time();`

Returns the scheduler's current simulation time.

`CNEventIterator *create_iterator();`

Creates an iterator object to traverse the list of events.

6.8 CNEventIterator — iterate through event list

SYNOPSIS

```
#include <CNCL/EventLIterator.h> #include <CNCL/EventHIterator.h>
```

TYPE

`CN_EVENTLITERATOR` `CN_EVENTHITERATOR`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNEvent, CNEventScheduler, CNEventHeapScheduler

DESCRIPTION

The classes `CNEventLIterator` and `CNEventHIterator` are used to step through the list of events. Single events may be deleted. The classes can only be created on the heap by calling the member function `create_iterator` of the current scheduler.

In addition to the member functions required by CNCL, `CNEventLIterator` and `CNEventHIterator` provide:

`CNEvent *next_event();`

> Returns an event every time it is called. In case the `CNEventHeapSched` is used it can not be guaranteed that the same event is delivered only once. Furthermore the order in which the events are delivered needn't obey any rule. It is however guaranteed that every event will be returned at least once.

`void delete_current_event()`

> Deletes the event last returned by `next_event()`.

6.9 CNSimTime — Simulation Time

SYNOPSIS

```
#include <CNCL/SimTime.h>
```

TYPE

```
typedef double CNSimTime
```

DESCRIPTION

`CNSimTime` is a double variable which keeps the current simulation time.

6.10 Example of an Event Driven Simulation

The following example shows how to program an M/M/1 queuing system simulation with CNCL events:

```
// -*- C++ -*-

#include <iostream.h>

#include <CNCL/QueueFIFO.h>
#include <CNCL/EventScheduler.h>
#include <CNCL/FiboG.h>
#include <CNCL/NegExp.h>
#include <CNCL/Moments.h>
#include <CNCL/Job.h>

enum { NJOBS=100000 };

enum { EV_JOB, EV_TIMER_G, EV_TIMER_S }; // Event types for M/M/1 simulation

class Server : public CNEventHandler
{
private:
    CNJob *job;             // Served job
    CNQueueFIFO queue;    // CNQueue
    CNRandom &rnd_b;      // Distribution of service time b
    CNMoments t_w, t_b;   // Evaluation tau_w, tau_b
    enum { ST_WAITING, ST_SERVING };

public:
    virtual void event_handler(const CNEvent *ev);

    void print_results();
    void eval_job(CNJob *job);

    Server(CNRandom &rnd) : rnd_b(rnd), job(NIL), t_w("tau_w"), t_b("tau_b")
    {
        state(ST_WAITING);
    }
};

class Generator : public CNEventHandler
{
private:
    CNRandom &rnd_a;      // Distribution of arrival time a
    Server *server;       // Connected queue/server
    long n;

public:
    virtual void event_handler(const CNEvent *ev);

    Generator(CNRandom &rnd,Server *serv) : rnd_a(rnd),server(serv),n(0) {}
};

void Generator::event_handler(const CNEvent *ev)
{
    if(n == NJOBS)
```

```
            // Stop simulation
            return;

        // Incoming event -> generate new Job
        send_now(new CNEvent(EV_JOB, server, new CNJob));
        // Random delay
        send_delay(new CNEvent(EV_TIMER_G), rnd_a());
        n++;
    }



    void Server::event_handler(const CNEvent *ev)
    {
        switch(state())
        {
        case ST_SERVING:
            switch(ev->type())
            {
            case EV_JOB:
                // Incoming job, put into queue
                {
                    Job *job;
                    job = (CNJob *)ev->object();
                    job->in = now();
                    queue.put(job);
                }
                break;

            case EV_TIMER_S:
                // Timer event, service time run down
                job->out = now();
                // Evaluate job
                eval_job(job);
                delete job;
                job = NIL;
                // Get new job from queue
                if(!queue.empty())
                {
                    job = (CNJob *)queue.get();
                    job->start = now();
                    // Random service time
                    send_delay(new CNEvent(EV_TIMER_S), rnd_b());
                    state(ST_SERVING);
                }
                else
                    state(ST_WAITING);
                break;
            }
            break;

        case ST_WAITING:
            switch(ev->type())
            {
            case EV_JOB:
                // Incoming job
                job = (CNJob *)ev->object();
```

```
            job->in    = now();
            job->start = now();
            // CNRandom service time
            send_delay(new CNEvent(EV_TIMER_S), rnd_b());
            state(ST_SERVING);
            break;
        }
        break;
    }
}

void Server::eval_job(CNJob *job)
{
    t_w.put(job->start - job->in);
    t_b.put(job->out   - job->in);
}


void Server::print_results()
{
    cout << t_w << t_b;
}

main()
{
    CNRNG   *rng   = new CNFiboG;
    CNNegExp rnd_a(10, rng);
    CNNegExp rnd_b( 5, rng);

    Server           server(rnd_b);
    Generator        generator(rnd_a, &server);

    CNEventScheduler scheduler;
    scheduler.start(new CNEvent(EV_TIMER_G, &generator, 0));

    server.print_results();

}
```

7 Array Classes

The class `CNArray` and the derived classes provide arrays of different data types with array range checking. `CNArrayObject` manages pointers to `CNObject`. The other classes `CNArray<`*type*`>` manage arrays of standard data types.

The main purpose of these classes is to provide arrays with *range checking*, i.e. access to an array element outside the arrays bounds will terminate the program.

Range checking may be disabled by defining the preprocessor macro `NO_RANGE_CHECK`, e.g. by supplying `-DNO_RANGE_CHECK` on the compiler's command line.

7.1 CNArray — Abstract Array Base Class

SYNOPSIS

```
#include <CNCL/Array.h>
```

TYPE

`CN_ARRAY`

BASE CLASSES

CNObject

DERIVED CLASSES

CNArrayObject, CNArrayChar, CNArrayDouble, CNArrayFLoat, CNArrayInt, ...

RELATED CLASSES

None

DESCRIPTION

`CNArray` is the base class of the `CNArray<`*type*`>` classes. It defines the common interface.

Constructors:

```
CNArray();
CNArray(Param *param);
```

Initializes `CNArray`.

In addition to the member functions required by CNCL, `CNArray` provides:

```
unsigned get_size() const;
```

Returns the size of the array.

```
virtual void set_size(int sz = 0 ) = 0;
```

Sets the size of the array. This function must be implemented in the derived classes.

7.2 CNArrayObject — Array of Pointer to CNObject

SYNOPSIS

```
#include <CNCL/ArrayObject.h>
```

TYPE

`CN_ARRAYOBJECT`

BASE CLASSES

CNArray

DERIVED CLASSES

None

RELATED CLASSES

CNArrayChar, CNArrayDouble, CNArrayFLoat, CNArrayInt, CNArrayLong

DESCRIPTION

`CNArrayObject` manages arrays of pointers to `CNObject`.

Constructors:

```
CNArrayObject();
CNArrayObject(Param *param);
CNArrayObject(int sz);
```

Initializes the array and optionally sets array size to `sz`. All element pointers are initialized to `NIL`.

```
CNArrayObject(const CNArrayObject &a);
```

Copy constructor.

Destructors:

```
~CNArrayObject();
```

Deletes the array. The referenced objects are NOT deleted!

In addition to the member functions required by CNCL, `CNArrayObject` provides:

```
void set_size(int sz = 0 );
```

Sets the size of the array.

```
void put (int index, CNObject *value);
```

Puts value into array at indexed location.

```
CNObject *get (int index) const;
```

Returns value of array at indexed location.

`CNObject*& operator[] (int index);`

Access to array by `operator []`.

`CNArrayObject &operator= (const CNArrayObject &a);`

Defines the `operator =` for the array to allow copying of arrays.

7.3 CNArrayInt — Array of Integer

SYNOPSIS

```
#include <CNCL/ArrayInt.h>
```

TYPE

`CN_ARRAYINT`

BASE CLASSES

CNArray

DERIVED CLASSES

None

RELATED CLASSES

CNArrayObject, CNArrayChar, CNArrayDouble, CNArrayFLoat, CNArrayLong

DESCRIPTION

`CNArrayInt` manages arrays of integer (the builtin type `int`). `CNArrayInt` is presented here as an example for all `CNArray<`*type*`>` classes. The interface is the same for all classes only considering the different data types.

Constructors:

```
CNArrayInt();
```

```
CNArrayInt(Param *param);
```

```
CNArrayInt(int sz, int def=0);
```

Initializes the array and optionally sets array size to `sz`. All elements are set to the default value `def`.

```
CNArrayInt(const CNArrayInt &a);
```

Copy constructor.

Destructors:

```
~CNArrayInt();
```

Deletes the array.

In addition to the member functions required by CNCL, `CNArrayInt` provides:

```
void set_size(int sz = 0 );
```

Sets the size of the array.

```
void put (int index, int *value);
```

Puts value into array at indexed location.

`int get (int index) const;`

Returns value of array at indexed location.

`int& operator[] (int index);`

Access to array by `operator []`.

`CNArrayInt &operator= (const CNArrayInt &a);`

Defines the `operator =` for the array to allow copying of arrays.

7.4 CNArray<type> — Arrays of Other <Type>s

DESCRIPTION

CNCL currently provides array classes for the data types `char`, `double`, `float`, `int`, `long`, and `CNObject *` with the classes `CNArrayChar`, `CNArrayDouble`, `CNArrayFloat`, `CNArrayInt`, `CNArrayLong`, and `CNArrayObject` respectively.

All CNCL compatible objects can be stored in a `CNArrayObject`, thus that there is no need for specialized array types.

Nevertheless it is possible to generate arrays of other data types with the `CNarray` script.

Usage:

`CNarray` *name*

The required parameter is the name of the data type. `CNarray` generates two files *ArrayName.h* and *ArrayName.c* with the definition and implementation of the desired array class.

Please note that *name* must be a single word, pointers and references are not allowed, either. If you need an array of pointers or e.g. an array of `unsigned long`, you can use an appropiate typedef:

```
typedef unsigned long ulong;
typedef Data *DataP;
```

and then generate an array

```
CNarray ulong
CNarray DataP
```

yielding the classes `CNArrayUlong` and `CNArrayDataP`.

7.5 CNArray2 — Base class for 2-dimensional arrays

SYNOPSIS

```
#include <CNCL/Array2.h >
```

TYPE

`CN_ARRAY2`

BASE CLASSES

CNObject

DERIVED CLASSES

CNArray2Char, CNArray2Double, CNArray2Float, CNArray2Int, CNArray2Long, CNArray2Object

RELATED CLASSES

CNArray

DESCRIPTION

`CNArray2` is the base class of the `CNArray2<`*type*`>` classes. It defines the common interface.

Constructors:

```
CNArray2();
CNArray2(Param *param);
CNArray2(unsigned r, unsigned c);
```

Initializes `CNArray2`.The number of rows is r, the number of cols c.

In addition to the member functions required by CNCL, `CNArray2` provides:

```
unsigned get_rows() const;
unsigned rows() const;
unsigned get_cols() const;
unsigned cols() const;
```

Returns the number of rows resp. cols.

```
virtual void size(int r, int c);
void set_size(int r, int c);
```

Resizes the array.

7.6 CNArray2Char — char array class

SYNOPSIS

```
#include <CNCL/Array2Char.h>
```

TYPE

```
CN_ARRAY2CHAR
```

BASE CLASSES

CNArray2

DERIVED CLASSES

None

RELATED CLASSES

CNArray2Double, CNArray2Float, CNArray2Int, CNArray2Long, CNArray2Object

DESCRIPTION

Constructors:

```
CNArray2Char();
CNArray2Char(CNParam *param);
CNArray2Char(int r, int c, char deff = 0);
CNArray2Char(const CNArray2Char &a);
```

Initializes the `CNArray2Char` and optionally sets the arraysize to r rows and c cols. There is a copy constructor, too.

Destructor:

```
~CNArray2Char();
```

Deletes the array.

In addition to the member functions required by CNCL, `CNArray2Char` provides:

```
virtual void size(int r, int c);
```

Resizes the array to r rows and c cols.

```
void put(int r, int c, chr value);
```

Writes the character value to position (r, c).

```
char get(int r, int c) const;
```

Returns the character written on position (r, c).

7.7 CNArray2<type> — 2 dimensional Arrays of Other <Type>s

DESCRIPTION

CNCL currently provides 2 dimensional array classes for the data types `char`, `double`, `float`, `int`, `long`, and `CNObject *` with the classes `CNArray2Char`, `CNArray2Double`, `CNArray2Float`, `CNArray2Int`, `CNArray2Long`, and `CNArray2Object` respectively. The description of those classes is similar to `CNArray2Char`.

All CNCL compatible objects can be stored in a `CNArray2Object`, thus that there is no need for specialized array types.

Nevertheless it is possible to generate arrays of other data types with the `CNarray2` script.

Usage:

`CNarray2` *name*

The required parameter is the name of the data type. `CNarray2` generates two files *Array2Name.h* and *Array2Name.c* with the definition and implementation of the desired array class.

Please note that *name* must be a single word, pointers and references are not allowed, either. If you need an array of pointers or e.g. an array of `unsigned long`, you can use an appropiate typedef:

```
typedef unsigned long ulong;
typedef Data *DataP;
```

and then generate an array

```
CNarray2 ulong
CNarray2 DataP
```

yielding the classes `CNArray2Ulong` and `CNArray2DataP`.

8 Object Management

The classes described in this chapter provide facilities necessary for object management, as abstract keys, hash tables, and an object management fronend class.

8.1 CNKey — Abstract Base Class for Object Management via Keys

SYNOPSIS

```
#include <CNCL/Key.h>
```

TYPE

`CN_KEY`

BASE CLASSES

CNObject

DERIVED CLASSES

CNKeyString, CNKeyInt

RELATED CLASSES

CNHashTable, CNHashStatic, CNHashDynamic, CNHashIterator, CNManager

DESCRIPTION

`CNKey` is an abstract class for managing CNCL compatible objects via keys. Refer to the desription of the classes derived from `CNKey` for further information. Objects of this type can be stored in and retrieved from hash tables.

Constructors:

```
CNKey(CNObject *obj = NIL);
CNKey(CNParam *param);
```

Initializes `CNKey`.

In addition to the member functions required by CNCL, `CNKey` provides:

```
void set_object(CNObject *obj);
void set_object(CNObject &obj);
```

Stores a CNCL compatible object into the key. Normally, an object is stored in a key at creation time via the constructor.

```
CNObject *get_object() const;
```

Gets the object stored in the key. If an object is not available, `NIL` is returned.

The following virtual functions are to be defined by derived classes:

```
virtual unsigned long hash( unsigned long capacity, int par = 0) const = 0;
```

Function to evaluate the hash-table value.

```
virtual bool compare(CNKey *k) const = 0;
virtual bool compare(CNKey &k) const = 0;
```

Function to compare two `CNKeys`.

8.2 CNKeyString — Object Management via String Keys

SYNOPSIS

```
#include <CNCL/KeyString.h>
```

TYPE

`CN_KEYSTRING`

BASE CLASSES

CNKey

DERIVED CLASSES

None

RELATED CLASSES

CNKeyInt, CNHashTable, CNHashStatic, CNHashDynamic, CNHashIterator, CNManager

DESCRIPTION

`CNKeyString` is a class for managing CNCL compatible objects via string keys. Objects of this type can be stored in and retrieved from hash tables.

Constructors:

```
CNKeyString(char *key_string, CNObject *obj = NIL);
CNKeyString(CNParam *param);
```

Initializes `CNKeyString`. The supplied string key is used to calculate the hash table position. Therefore, the string key *must* be unique. Make sure, that the string key is valid during the whole lifetime of the respective key. Otherwise operations on this key are unpredictable.

In addition to the member functions required by CNCL, `CNKeyString` provides:

```
char *get_key() const;
```

Returns the string key. Unlike the object the string key cannot be changed.

```
virtual unsigned long hash( unsigned long capacity, int par = 0) const = 0;
```

Evaluates and returns the hash-table value. `par` is reserved for future use, any other value than zero will result in an fatal error.

```
virtual bool compare(CNKey *k) const = 0;
virtual bool compare(CNKey &k) const = 0;
```

Compares two `CNKeys`.

8.3 CNKeyInt — Object Management via Integer Keys

SYNOPSIS

```
#include <CNCL/KeyInt.h>
```

TYPE

`CN_KEYINT`

BASE CLASSES

CNKey

DERIVED CLASSES

None

RELATED CLASSES

CNKeyString, CNHashTable, CNHashStatic, CNHashDynamic, CNHashIterator, CNManager

DESCRIPTION

`CNKeyInt` is a class for managing CNCL compatible objects via integer keys. Objects of this type can be stored in and retrieved from hash tables.

Constructors:

```
CNKeyInt(unsigned long key_int, CNObject *obj = NIL);
CNKeyInt(CNParam *param);
```

Initializes `CNKeyInt`. The supplied integer key is used to calculate the hash table position. Therefore, the integer key *must* be unique.

In addition to the member functions required by CNCL, `CNKeyInt` provides:

```
unsigned long get_key() const;
```

Returns the integer key. Unlike the object the integer key cannot be changed.

```
virtual unsigned long hash( unsigned long capacity, int par = 0) const = 0;
```

Evaluates the hash-table value.

```
virtual bool compare(CNKey *k) const = 0;
virtual bool compare(CNKey &k) const = 0;
```

Compares two `CNKeys`.

8.4 CNHashTable — Abstract Base Class for Hash Tables

SYNOPSIS

```
#include <CNCL/HashTable.h>
```

TYPE

`CN_HASHTABLE`

BASE CLASSES

CNObject

DERIVED CLASSES

CNHashStatic, CNHashDynamic

RELATED CLASSES

CNKey, CNKeyString, CNKeyInt, CNHashIterator, CNManager

DESCRIPTION

`CNHashTable` is an abstract base class for storing and retrieving CNCL compatible objects.

Constructors:

```
CNHashTable();
CNHashTable(CNParam *param);
```

Initializes `CNHashTable`.

`CNHashTable` provides the following *virtual* functions which have to be defined in derived classes:

```
const unsigned long DEFAULT_HASH_TABLE_CAPACITY = 101;
```

The default capacity of a hash table. This value is used in derived classes.

```
virtual void store_key(CNKey *k) = 0;
virtual void store_key(CNKey &k);
```

Stores a key into the actual hash table (derived from class `HashTable`). The actual hash table is a homogenous table. Therefore, only keys of the *same* type may be stored into one table. If you nevertheless try to store keys of a different type into one table, it might be detected by the methods `get_key()` and `get_object()`. Refer to the description of the derived classes for further information.

```
virtual CNKey *get_key(CNKey *k) const = 0;
virtualCNKey *get_key(CNKey &k) const;
```

Returns the key which matches the supplied key. If no matching key was found, `NIL` is returned.

```
virtual CNObject *get_object(CNKey *k) const = 0;
```

`virtual CNObject *get_object(CNKey &k) const;`

Returns the object of the key which matches the supplied key. If no matching key was found or if no object has been stored in the key, `NIL` is returned.

`virtual bool reset() = 0;`

Deletes all entries of the actual hash table and resets it to its initial state. This method *does not* free the memory allocated for the keys stored in the hash table. If the hash table is already empty, `FALSE` is returned, otherwise `TRUE`.

`virtual bool reset_absolutely() = 0;`

Deletes all entries of the actual hash table and resets it to its initial state. This method *does* free the memory allocated for the keys stored in the hash table. If the hash table is already empty, `FALSE` is returned, otherwise `TRUE`.

`virtual bool delete_key(Key *k) = 0;`

`virtual bool delete_key(CNKey &k);`

Deletes the key `k` from the actual hash table. This method *does not* free the memory allocated for the key stored in the hash table. If the supplied key does not match any in the table, `FALSE` is returned, otherwise `TRUE`.

`virtual bool delete_key_absolutely(CNKey *k) = 0;`

`virtual bool delete_key_absolutely(CNKey &k);`

Deletes the key `k` from the actual hash table. This method *does* free the memory allocated for the keys stored in the hash table. If the supplied key does not match any in the table, `FALSE` is returned, otherwise `TRUE`.

`virtual bool is_full() const = 0;`

If the actual table's capacity is exhausted, `TRUE` is returned, otherwise `FALSE`.

`virtual bool is_empty() const = 0;`

If the actual table is empty, `TRUE` is returned, otherwise `FALSE`.

`virtual unsigned long get_num_entries() const = 0;`

Returns the number of entries of the actual table.

`virtual unsigned long get_capacity() const = 0;`

Returns the capacity of the current table.

8.5 CNHashStatic — Hash Tables with Static Capacity

SYNOPSIS

```
#include <CNCL/HashStatic.h>
```

TYPE

`CN_HASHSTATIC`

BASE CLASSES

CNHashTable

DERIVED CLASSES

None

RELATED CLASSES

CNHashDynamic, CNKey, CNKeyString, CNHashIterator, CNKeyInt

DESCRIPTION

`CNHashStatic` is a class which provides a hash table with static capacity for storing and retrieving CNCL compatible objects.

Constructors:

```
CNHashStatic(unsigned long cap = DEFAULT_HASH_TABLE_CAPACITY);
CNHashStatic(CNParam *param);
```

Initializes `CNHashStatic`. The hash table's capacity is set to the value passed to CNHashStatic. The capacity is static, therefore, you cannot change it during the lifetime of an instance of this class.

Destructors:

```
~CNHashStatic();
```

Frees all internally allocated resources.

`CNHashStatic` provides the member functions required by CNCL and `CNHashTable`. Some member functions defined in `CNHashTable` and implemented in `CNHashStatic` demand further explanation:

```
void store_key(CNKey *k);
```

Stores a key into the homogenou hash table. Only keys of the *same* type may be stored into the same table. The methods `get_key()` and `get_object()` should detect it. If you try to store a key into an already full table, an error message is displayed and the program is terminated.

`bool delete_key(CNKey *k);`

Deletes the key from the actual hash table which matches the given key. After having deleted a key from the hash table, the whole table is rehashed, i.e. the positions of all entries within the hash table are recalculated and all entries are stored in a new hash table. This might lead to small time delays when handling large hash tables. This method *does not* free the memory allocated for the keys stored in the hash table. If the supplied key does not match any in the table, `FALSE` is returned, otherwise `TRUE`.

`bool delete_key_absolutely(CNKey *k);`

Deletes the key from the actual hash table which matches the given key. After having deleted a key from the hash table, the whole table is rehashed. This method *does* free the memory allocated for the keys stored in the hash table. If the supplied key does not match any in the table, `FALSE` is returned, otherwise `TRUE`.

The following example shows how to use a `CNHashStatic` object in order to store and retrieve CNCL compatible objects.

```
CNHashStatic tab(200);
CNKeyString ks("Test", NIL);
CNObject *obj = &ks;

tab.store_key(new CNKeyString("Jabba", obj));
tab.store_key(new CNKeyString("Dabba"));
tab.store_key(new CNKeyString("Dooo"));

if (tab.get_object(CNKeyString("Jabba")) != obj)
   cout << "strange behaviour\n";
else
   cout << "found obj\n";

tab.store_key(new CNKeyInt(10, obj)); // error, key of different type

tab.reset_absolutely();

tab.store_key(new CNKeyInt(10, obj)); // okay
```

8.6 CNHashDynamic — Hash Tables with Dynamic Capacity

SYNOPSIS

```
#include <CNCL/HashDynamic.h>
```

TYPE

`CN_HASHDYNAMIC`

BASE CLASSES

CNHashTable

DERIVED CLASSES

None

RELATED CLASSES

CNHashStatic, CNHashIterator, CNKey, CNKeyString, CNKeyInt, CNManager

DESCRIPTION

`CNHashDynamic` is a class which provides a hash table with dynamic capacity for storing and retrieving CNCL compatible objects.

Constructors:

```
CNHashDynamic(unsigned long cap = DEFAULT_HASH_TABLE_CAPACITY);
CNHashDynamic(CNParam *param);
```

Initializes `CNHashDynamic`. The hash table's capacity is set to the value passed to HashDynamic. The capacity is dynamic, i.e. if the number of entries exceeds 3/4 of the hash table's capacity, it is enlarged to a proper value.

Destructors:

```
~CNHashDynamic();
```

Frees all internally allocated resources.

`CNHashDynamic` provides the member functions required by CNCL and `CNHashTable`. Some member functions defined in `CNHashTable` and implemented in `CNHashDynamic` demand further explanation:

```
void store_key(CNKey *k);
```

Stores a key into the homogenous hash table. Therefore, only keys of the *same* type may be stored into the same table. If you nevertheless try to store keys of different types into one table, it might be detected by the methods `get_key()` and `get_object()`. The capacity is dynamic, i.e. if the number of entries exceeds 3/4 of the hash table's capacity, it is enlarged to a proper value.

`bool delete_key(CNKey *k);`

Deletes the key from the actual hash table which matches the given key. After having deleted a key from the hash table, the whole table is rehashed, i.e. the positions of all entries within the hash table are recalculated and all entries are stored in a new hash table. This might lead to small time delays when handling large hash tables. This method *does not* free the memory allocated for the keys stored in the hash table. If the supplied key does not match any in the table, `FALSE` is returned, otherwise `TRUE`.

`bool delete_key_absolutely(CNKey *k);`

Deletes the key from the actual hash table, which matches the given key. After having deleted a key from the hash table, the whole table is rehashed. This method *does* free the memory allocated for the keys stored in the hash table. If the supplied key does not match any in the table, `FALSE` is returned, otherwise `TRUE`.

Refer to `CNHashStatic` for an example as to how to use a `CNHashDynamic` object in order to store and retrieve CNCL compatible objects.

8.7 CNHashIterator — Sequential Iterator for Hash Tables

SYNOPSIS

```
#include <CNCL/HashIterator.h>
```

TYPE

`CN_HASHITERATOR`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNHashTable, CNHashDynamic, CNHashStatic, CNKey, CNKeyString, CNKeyInt, CNManager

DESCRIPTION

`CNHashIterator` is an iterator to sequentially traverse a `CNHashTable` hash table.

Constructors:

```
CNHashIterator();
CNHashIterator(CNParam *param);
```

Initializes `CNHashIterator`.

```
CNHashIterator(const CNHashTable *new_hash_table);
CNHashIterator(const CNHashTable &new_hash_table);
```

Initializes `CNHashIterator` with hash table `new_hash_table`. The iterator is reset to the first element of the hash table.

In addition to the member functions required by CNCL, `CNHashIterator` provides:

```
void reset(const CNHashTable *new_hash_table);
void reset(const CNHashTable &new_hash_table);
void reset();
```

Resets the iterator to a new hash table `new_hash_table` and/or sets the iterator to the first element of the hash table.

```
CNKey *key()
CNKey *get_key()
```

Gets the referenced key from the current iterator position. It returns the key or `NIL`, if none is available.

`CNKey *first_key();`

`CNKey *first();`

Sets the iterator to the first element of the hash table. It returns the referenced key or `NIL`, if none is available.

`CNKey *last_key();`

`CNKey *last();`

Sets the iterator to the last element of the hash table. It returns the referenced key or `NIL`, if none is available.

`CNKey *next_key();`

`CNKey *next();`

`CNKey *operator ++();`

`CNKey *operator ++(int);`

Moves the iterator to the next element of the hash table. It returns the current referenced key (the one BEFORE moving the iterator) or `NIL`, if none is available.

`CNKey *prev_key();`

`CNKey *prev();`

`CNKey *operator --();`

`CNKey *operator --(int);`

Moves the iterator to the previous element of the hash table. It returns the current referenced key (the one BEFORE moving the iterator) or `NIL`, if none is available.

`CNObject *object()`

`CNObject *get_object()`

Gets the object of the referenced key from the current iterator position. It returns the object or `NIL`, if none is available.

`CNObject *first_object();`

Sets the iterator to the first element of the hash table. It returns the object of the referenced key or `NIL`, if none is available.

`CNObject *last_object();`

Sets the iterator to the last element of the hash table. It returns the object of the referenced key or `NIL`, if none is available.

`CNObject *next_object();`

Moves the iterator to the next element of the hash table. It returns the object of the current referenced key (the one BEFORE moving the iterator) or `NIL`, if none is available.

`CNObject *prev_object();`

Moves the iterator to the previous element of the hash table. It returns the current referenced key (the one BEFORE moving the iterator) or `NIL`, if none is available.

The following examples show how to use a `CNHashIterator` object to traverse an hash table:

Forward:

```
CNHashDynamic hash_table;

...
```

```
CNHashIterator trav(hash_table);
CNKey *key;
CNObject *obj;

while(key = trav++)
{
    // Do something with key ...
    obj = key->get_object();
}
```

Alternate forward:

```
for(trav.reset(hash_table); key = trav.key(); trav.next())
{
    // ...
}
```

Forward with objects:

```
for(trav.first(); obj = trav.object(); trav.next_object())
{
    // ...
}
```

Backward:

```
for(trav.last(); key = trav.key(); trav--)
{
    // ...
}
```

The only way to delete all entries of an hash table, that is guaranteed to work with `CNHashIterator`:

```
for(trav.reset(hash_table); key = trav.key(); trav.reset())
{
    // delete object associated with key
    delete key->get_object();
    // delete hash table entry and key
    hash_table->delete_key_absolutely(key);
}
```

8.8 CNManager – Object Management Frontend

SYNOPSIS

```
#include <CNCL/Manager.h>
```

TYPE

`CN_MANAGER`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNHashTable, CNHashDynamic, CNHashIterator, CNKey, CNKeyString, CNKeyInt

DESCRIPTION

`CNManager` is a class which provides facilities for storing and retrieving CNCL compatible objects in a filesystem-like manner. Objects of type `CN_MANAGER` can be compared with directories, whereas all other CNCL compatible objects play the role of the files. `CNManager` internally uses a dynamic hash table for managing the objects.

Constructors:

```
CNManager(char *object_name = NIL);
CNManager(CNParam *param);
```

Initializes `CNManager`.

Destructor:

```
~CNManager();
```

Frees all internally allocated resources.

In addition to the member functions required by CNCL, `CNManager` provides:

```
CNObject *new_object(char *object_name, CNClassDesc desc, CNParam *param = NIL);
```

`new_object()` is used to create a tree of objects similar to a filesystem's tree of directories and files. The class descriptor `desc` determines whether a directory (`CN_MANAGER`) or a file (any other type) is to be created. The `object_name` may consist of a pathname and/or a basename. All directory-names and the basename must be separated by slashes ('/'). The pointer to the new created object is returned, if no error occurs, otherwise, if e. g. the specified pathname was incorrect, `NIL` is returned.

`bool delete_object(char *object_name);`

Removes the specified object from the internal hashtable. The memory allocated for the object via `new_object()` is deleted, too. As described under `new_object()` you may pass a filesystem-like path to `delete_object()`. If the object is deleted succesfully, `TRUE` is returned, `FALSE` otherwise. If the object to be deleted is a 'directory' and if the 'directory' still contains valid 'files', `delete_object()` fails. You must first delete all subentries before deleting the entry itself.

`CNObject *get_object(char *object_name) const;`

Returns the object associated to the given pathname. If the specified object does not exist, `NIL` is returned.

`char *get_name();`

Returns the basename of the respective object.

`bool is_empty() const;`

Returns TRUE if no object is stored in the storing facilities of this class, FALSE otherwise.

The following example shows how to use `CNManager` objects in order to store and retrieve CNCL compatible objects.

```
// this is the root of the example's object hierarchy
CNManager root;
// some 'directory' pointers
CNManager *mobile, *base, *subbase;
// some 'file' pointers
CNHashDynamic *table1, *table2;

// create a 'directory' in the root 'directory'
mobile = (CNManager *)root.new_object("mobile", CN_MANAGER);
if (mobile == NIL)
    ...
// create another 'directory'
base = (CNManager *)root.new_object("base", CN_MANAGER);
if (base == NIL)
    ...

// now create a 'file' (CNCL compatible object) in the
// 'mobile' 'directory'
table1 = (CNHashDynamic *)mobile->new_object("table1", CN_HASHDYNAMIC);
if (table1 == NIL)
    ...

// now create a 'subdirectory' of 'base' using an absolute path
subbase = (CNManager *)root.new_object("/base/subbase", CN_MANAGER);
if (subbase == NIL)
    ...

// create a 'file' using a path relative to 'base'
table2 = (CNHashDynamic *)base->new_object("subbase/table2", CN_HASHDYNAMIC);
if (table2 == NIL);
    ...

// now try to get an object
if (base->get_object("subbase/table2") == table2)
    ...
```

```
// try to delete 'subbase'
if (!root.delete_object("/base/subbase")) // error, dir not empty
    ...

// first delete all subentries
if (!root.delete_object("/base/subbase/table2"))
    ...

// now delete subbase
if (!base->delete_object("subbase")) // okay
    ...

...
```

9 Miscellaneous Classes

The classes described in this chapter are provided by the CNCL class library for miscellaneous purposes such as:

common coordinates for the graphical interface to EZD

common string handling (as a *CNObject*)

common integer and double handling (as a *CNObject*)

support of named object management

integer2string and double2string conversion

9.1 CNCoord — 2-Dimensional Coordinates

SYNOPSIS

```
#include <CNCL/Coord.h>
```

TYPE

`CN_COORD`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNICoord

DESCRIPTION

`CNCoord` is a data type for managing 2-dimensional coordinates. It is typically used together with `CNICoord` for world coordinates and pixel coordinates respectively.
A `CNCoord` has `double` x and y members which are public accessible.
`CNCoord`s can be automatically converted to `CNICoord`s and vice versa. This is done by applying the conversion factor `CNCoord::scale`.

Constructors:

```
CNCoord();
CNCoord(CNParam *param);
CNCoord(double vx, double vy);
CNCoord(const CNICoord &v);
CNCoord(const CNCoord &v);
```

Initializes the coordinates object and optionally sets x and y components.

Public accessible members:

`double x;`

`double y;` The x and y components of `CNCoord`.

In addition to the member functions required by CNCL, `CNCoord` provides:

```
CNCoord &operator += (const CNCoord &v);
CNCoord &operator -= (const CNCoord &v);
```

Defines the operators `+=` and `-=` for coordinates by applying the standard C/C++ operators to the x and y components.

The following static member functions are provided to manipulate the conversion scale setting:

```
static double CNCoord::get_scale();
static double CNCoord::set_scale(double new_scale);
```

Gets/sets the scale setting.

Global operators:

```
CNCoord operator + (const CNCoord &a, const CNCoord &b);
CNCoord operator - (const CNCoord &a, const CNCoord &b);
```

Adds/substracts coordinates by adding/subtracting the x and y components.

9.2 CNICoord — 2-Dimensional Integer Coordinates

SYNOPSIS

```
#include <CNCL/ICoord.h>
```

TYPE

`CN_ICOORD`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNCoord

DESCRIPTION

`CNICoord` is a data type for managing 2-dimensional integer coordinates. It is typically used with `CNCoord` for pixel coordinates and world coordinates respectively.
A `CNICoord` has `int` x and y members which are public accessible.
`CNICoord`s can be automatically converted to `CNCoord`s and vice versa. This is done by applying the conversion factor `CNCoord::scale`.

Constructors:

```
CNICoord();
CNICoord(CNParam *param);
CNICoord(int vx, int vy);
CNICoord(const CNCoord &v);
CNICoord(const CNICoord &v);
```

Initializes the integer coordinates object and optionally sets x and y components.

Public accessible members:

`int x;`

`int y;` The x and y components of `CNICoord`.

In addition to the member functions required by CNCL, `CNICoord` provides:

```
CNICoord &operator += (const CNICoord &v);
CNICoord &operator -= (const CNICoord &v);
```

Defines the operators `+=` and `-=` for integer coordinates, applying the standard C/C++ operators to the x and y components.

The following static member functions are provided to manipulate the conversion scale setting:

```
static double CNICoord::get_scale();
static double CNICoord::set_scale(double new_scale);
```

Gets/sets the scale setting.

Global operators:

```
CNICoord operator + (const CNICoord &a, const CNICoord &b);
CNICoord operator - (const CNICoord &a, const CNICoord &b);
```

Adds/substracts integer coordinates by adding/subtracting the x and y components.

9.3 CNString — Character String

SYNOPSIS

```
#include <CNCL/String.h>
```

TYPE

`CN_STRING`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNNamed

DESCRIPTION

`CNString` is a dynamic string manipulating class. Each `CNString` manages the string itself (stored as a normal C character string), the length of the string, and the allocated space. The characters are indexed from 0 to *length*-1.

Constructors:

```
CNString();
CNString(int extra);
CNString(const char* cs);
CNString(const char* cs, int extra);
CNString(const CNString& s);
CNString(const CNString& s, int extra);
CNString(CNParam *param);
```

Initializes the CNString. The `extra` parameter gives a hint as to how many extra characters one expects the string to grow. Its default value is 10.

Destructors:

```
~CNString();
```

Deletes the CNString.

In addition to the member functions required by CNCL, `CNString` provides:

```
default_extra = 10;
```

Default amount of additional storage allocated. Actually set to 10.

`void resize(unsigned i);`

Changes the capacity of the string either to its length plus default_extra or to i, whichever is longer.

`void to_lower();`

Converts all characters in this string to lowercase.

`void to_upper();`

Converts all characters in this string to uppercase.

`void capitalize();`

Converts each first character of a word to uppercase, all other characters to lowercase. A new word is either at the beginning of the string or it starts with a space (e.g. "a.R fd" will be capitalized to "A.r Fd").

`void to_ascii();`

Converts the characters to ascii code.

`unsigned capacity() const;`

Returns the amount of space allocated to this string.

`unsigned length() const;`

Returns the length of this string.

`CNString after(unsigned pos, unsigned l = 0) const;`

Returns the l characters of this string after position pos (zero-based) as a string. The character at pos will be the first character of the new string. The default value l = 0 will return all characters after pos.

`CNString before(unsigned pos, unsigned l = 0)const;`

Returns the l characters of this string before position pos (zero based) as a string. The character at pos will be the last character of the new string. The default value l = 0 will return all characters before pos.

`CNString& add(char c);`

`CNString& add(char* cs);`

Adds the character c or the character string cs to the end of this string. If this string does not have enough space allocated it will be changed automatically.

`CNString& del(unsigned pos = 0, unsigned l = 1);`

Deletes l characters beginning at position pos (zero based). Default values are pos = 0 and l = 1 (e.g. a.del() will delete the first character of string a). If l is set to 0 all characters beginning at pos will be deleted (e.g. a.del(0,0) will delete all characters in string a).

`CNString& del(char c, unsigned pos = 0);`

`CNString& del(char* cs, unsigned pos = 0);`

`CNString& del(CNString& s, unsigned pos = 0);`

Deletes character c, character string cs or string s at its first occurence in this string after position pos (zero based, default value pos = 0). It deletes nothing if c, cs or s is not in.

`CNString& insert(char c, unsigned pos = 0);`

`CNString& insert(char* cs, unsigned pos =0);`

`CNString& insert(CNString& s, unsigned pos = 0);`

Inserts character c, character string cs or CNString s at position pos (zero based, default value pos = 0).

`CNString& replace(char c, unsigned pos = 0);`

Replaces the character at position pos (zero based, default value pos = 0) with the character c.

`CNString& replace(CNString& s, unsigned pos = 0, unsigned l = 0);`

Replaces the l characters starting at position pos (zero based, default value pos = 0) with the characters of string s. Default value l = 0 will replace as many characters as s.length(). The capacity is changed to the needed space plus the default extra value.

`CNString& replace(char oldc, char newc, unsigned pos = 0);`

Replaces character oldc at its first occurence after position pos (zero based, default value pos = 0) with newc. If oldc is not in this string after pos nothing will be changed.

`CNString& replace(CNString& olds, CNString& news, unsigned pos = 0);`

Replaces string olds at its first occurence after position pos (zero based, default value pos = 0) with string news. If olds is not in this string nothing will be changed.

`unsigned downsearch(char c, unsigned pos = 0);`

`unsigned downsearch(CNString& s, unsigned pos = 0);`

Returns the last occurence of character c or string s in this string before position pos (zero based, default value pos = 0) or the end of this string if pos = 0 or pos >= len. If c or s is not in this string before pos len is returned.

`unsigned upsearch(char c, unsigned pos = 0);`

`unsigned upsearch(CNString&s, unsigned pos = 0);`

Returns the first occurence of character c or string s in this string after position pos (zero based, default value pos = 0). Returns len if c or s is not in this string after pos.

`bool matches(char* cs, unsigned pos = 0);`

`bool matches(CNString& s, unsigned pos = 0);`

Returns TRUE if the character string cs or the string s is equal to the part of this string starting at position pos (zero based, default value pos = 0).

`operator char * () const;`

`operator const char * () const;`

Returns the the C string component of `CNString` to allow the use of `CNString`s in a `char *` context.

`char &operator ()(int i);`

`char &operator [](int i);`

Returns the i-th character (zero based) of this string.

`void *operator = (const CNString& s);`

`void *operator = (const char* cs);`

`void *operator = (const char c);`

Replaces this string with a copy of string s, character string cs or character c.

`friend CNString operator + (const CNString& a, const CNString& b);`

Returns a string that is the concatenation of the strings `a` and `b`.

`CNString& *operator +=(const CNString &s);`

```
CNString& *operator +=(const char* cs);
```

Appends the C string `s` or the `CNString` `cs` to the end of this string.

```
friend bool operator < (const CNString& a, const CNString& b);
friend bool operator > (const CNString& a, const CNString& b);
friend bool operator >= (const CNString& a, const CNString& b);
friend bool operator <= (const CNString& a, const CNString& b);
friend bool operator == (const CNString& a, const CNString& b);
friend bool operator != (const CNString& a, const CNString& b);
```

Returns `TRUE` if the relation holds between the strings.

```
void icopy(istream& strm = cin);
istream &operator >> (istream& strm, CNString& s);
```

Reads all characters up to (not including) the next newline from input stream strm into the string.

For programming convenience, a type for a `const` reference to a string is provided:

```
typedef const CNString & CNStringR;
```

9.4 CNNamed — CNObject with Name

SYNOPSIS

```
#include <CNCL/Named.h>
```

TYPE

`CN_NAMED`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNString

DESCRIPTION

`CNNamed` is a data type for managing CNObject's names in CNString format.

Constructors:

```
CNNamed();
CNNamed(CNStringR name);
CNNamed(CNParam *param);
```

Initializes the string name object and optionally sets the name.

In addition to the member functions required by CNCL, `CNNamed` provides:

```
CNStringR name() const;
CNStringR get_name() const;
```

Returns the object's name.

```
void name(CNStringR name) const;
void set_name(CNStringR name) const;
```

Sets the object's name.

9.5 CNFormInt — Integers as CNStrings

SYNOPSIS

```
#include <CNCL/FormInt.h>
```

TYPE

`CN_FORMINT`

BASE CLASSES

CNString

DERIVED CLASSES

RELATED CLASSES

DESCRIPTION

This class converts Integers to Strings using different kinds of format styles provided by the stream-2.0-implementation. The current formats implemented in this class are 'left' and 'right'. Note: If the Integer needs more characters than the width `w` indicates, the most right ones are ignored.

Constructors:

```
CNFormInt();
CNFormInt(CNParam *param);
CNFormInt(int a, int w);
CNFormInt(int a, int w, char fill);
CNFormInt(int a, int w, char fill, int f);
```

Initializes `CNFormInt`, setting the value to the integer value a (default = 0), the string's width, the fill character fill (default = ' ') and the format f (default = CNFormInt::right)

The different formats implemented in `CNFormInt` are:

```
int right = 1
int left = 2
```

In addition to the member functions required by CNCL, `CNFormInt` provides:

```
int get_value();
int value();
```

Returns the value as an integer.

```
void set_value(int a);
void value(int a);
```

Changes the old String and the old value to a.

```
char get_fill();
char fill();
```
Returns the current fill character.
```
void set_fill(char f);
void fill(char f);
```
Changes the String to the fill character f.
```
int get_format();
int format();
```
Returns the current format as an integer. '1' descibes 'right', '2' describes left.
```
void set_format(int f);
void format(int f);
```
Changes the String to the new CNFormInt::formats f.
```
int get_width();
int width();
```
Returns the String's width as an integer.
```
void set_width(int w);
void width(int w);
```
Changes the old String's width to w.

9.6 CNFormFloat — Doubles as CNStrings

SYNOPSIS

```
#include <CNCL/FormFloat.h>
```

TYPE

`CN_FORMFLOAT`

BASE CLASSES

CNString

DERIVED CLASSES

None

RELATED CLASSES

CNFormInt

DESCRIPTION

This class converts doubles to CNStrings using different kinds of format styles provided by the stream-2.0-implementation. The current formats implemented in this class are 'left' and 'right' in connection with 'scientific', 'showpoint' or 'fixed'. Note: If the number is too long, the right part of it will be cut, no matter wich format style is chosen. If the number is cut (because of the chosen precision) it will be round. If the precision is not in the interval [0,16] problems with the accuracy might occur.

Constructors:

```
CNFormFloat();
CNFormFloat(CNParam *param);
CNFormFloat(double x, int w);
CNFormFloat(double x, int w, char fill);
CNFormFloat(double x, int w, char fill, int format);
CNFormFloat(double x, int w, char fill, int format, int pr);
```

Initializes the `FormFloat` with the value x (default = 0.0), the width w (= 3), the fill character fill (= ' '), the format (= right) and the precision pr (= 6).

The different formats implemented in `CNFormFloat` are:

```
int right = 1
int left = 2
int scientific = 4
int showpoint = 8
```

`int fixed = 16`

These formats can be devided into two groups: right and left on one hand and scientific, showpoint and fixed on the other. Those two groups can be combiened with each other, e.g. left and scientific (= 6) is as far possible as right and fixed (= 17).

In addition to the member functions required by CNCL, `CNFormFloat` provides:

```
double get_value();
double value();
char get_fill();
char fill();
formats get_format();
formats format();
int get_width();
int width();
int get_precision();
int precision();
```

Returns value, fill character, format style, width or precision of the current CNFormFloat.

```
void set_value(double x);
void value(double x);
void set_fill(char f);
void fill(char f);
void set_format(formats f);
format(formats f);
void set_width(int w);
void width(int w);
void set_precision(int pr);
void precision(int pr);
```

Changes the current CNFormFloat by its value, fill character, format style, width or precision.

9.7 CNInt — Integers derived from CNObject

SYNOPSIS

```
#include <CNCL/Int.h>
```

TYPE

`CN_INT`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNDouble

DESCRIPTION

This class provides a long int value derived from CNObject. Such it combines the behavior of the builtin type long int with the possibility to use this type with generic containers (like CNDLList) or as parameters to CNEvents and SDLSignals.
Note: CNInt consumes much more memory than an ordinary long int. Therefore it isn't a good idea to create large arrays of CNInts when one doesn't need its special capabilities.
Constructors:

`CNInt(long n=0);`

`CNInt(CNParam *param);`

Initializes the `Int` with the value n (default = 0).

In addition to the member functions required by CNCL, `CNInt` provides:

`operator long()`

Conversion of CNInt to ordinary long. This allows the use of `CNInt` in calculations. Note: to explicitly convert a CNInt identifier to a double (or else) write double(long(identifier)). Implicit conversion however works well without casting to long first.

`long operator ++()`

`long operator ++(int)`

Prefix and postfix version of increment.

`long operator --()`

`long operator --(int)`

Prefix and postfix version of decrement.

`long operator -()`

`long operator +()`

Unary minus and plus operator.

`long operator +=(long n)`

`long operator -=(long n)`

`long operator *=(long n)`

`long operator /=(long n)`

`long operator %=(long n)`

Arithmetic operators where a `CNInt` is on the left and on the right side of an equation.

`long operator ^=(long n)`

`long operator |=(long n)`

`long operator &=(long n)`

`long operator !=(long n)`

Logical operators where a `CNInt` is on the left and on the right side of an equation.

`long operator <<=(long n)`

`long operator >>=(long n)`

Left and right shifts of a `CNInt`.

Note: unary * and & operators aren't overloaded.

9.8 CNDouble — Doubles derived from CNObject

SYNOPSIS

```
#include <CNCL/Double.h>
```

TYPE

`CN_DOUBLE`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNInt

DESCRIPTION

This class provides a double value derived from CNObject. Such it combines the behavior of the builtin type double with the possibility to use this type with generic containers (like CNDLList) or as parameters to CNEvents and SDLSignals.
Note: CNDouble consumes much more memory than an ordinary double. Therfore it isn't a good idea to create large arrays of CNDoubles when one doesn't need its special capabilities.
Constructors:

`CNDouble(double n=0.0);`

`CNDouble(CNParam *param);`

Initializes the `Double` with the value n (default = 0.0).

In addition to the member functions required by CNCL, `CNDouble` provides:

`operator double()`

Conversion of CNDouble to ordinary double. This allows the use of `CNDouble` in calculations.
Note: to explicitly convert a CNDouble identifier to an ordinary long (or else) write long(double(identifier)). Implicit conversion however works well without casting to double first.

`double operator -()`

`double operator +()`

Unary minus and plus operator.

`double operator +=(double n)`

`double operator -=(double n)`

`double operator *=(double n)`

`double operator /=(double n)`

Arithmetic operators where a `CNDouble` is on the left and on the right side of an equation.

Note: unary * and & operators aren't overloaded.

10 Unix Classes

The classes described here provide an interface to the UNIX operating system. Thus they will only work on machines where UNIX or compareable systems like LINUX are installed. E.g., You won't be able to use them in a DOS environment.

Additional information about the different Unix systam calls see the man pages and your manual.

10.1 CNPipe — Unix Pipe

SYNOPSIS

```
#include <CNCL/Pipe.h>
```

TYPE

`CN_PIPE`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

`CNPipe` creates and manages Unix I/O pipes between two programs, thus that two cooperating processes can transfer data.

Constructors:

```
CNPipe();
CNPipe(CNParam *param);
CNPipe(const CNString& prog);
```

Initializing the pipe. If the program name `prog` is set, the pipe will be opened immediately. Otherwise the open() command has to be used when needed. The destructor closes the pipe if it was not closed before.

In addition to the member functions required by CNCL, `CNPipe` provides:

```
int open(const CNString& prog);
```

Opens the pipe to the program. If the pipe already exists, the old pipe will be closed before opening the new one. The returned integer value indicates the success of the opening process. If successful 0 is returned, otherwise an error message is shown and -1 is returned.

```
int close();
```

Closes an existing pipe. If an error occurs, -1 is returned, otherwise a 0.

```
int fd_in();
int fd_out();
```

Returns the I/O pipe file descriptors.

10.2 CNSelect — Class Interface to Select(2) System Call

SYNOPSIS

```
#include <CNCL/Select.h>
```

TYPE

`CN_SELECT`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

`CNSelect` is the interface to Unix's "select(2)" system call for synchronous I/O multiplexing. Select(2) (refer to Unix man pages, too) examines the descriptor sets and detects if they are ready for reading, writing, or having an exceptional status.

Constructors:

```
CNSelect();
CNSelect(CNParam *param);
CNSelect(int fd);
```

Initializes the three descriptor sets (one for the reading, writing and exceptional descriptor set) to null sets. Optionally the particular descriptor `fd` in the read set.

In addition to the member functions required by CNCL, `CNSelect` provides:

```
void add_read(int fd);
void del_read(int fd);
void test_read(int fd);
```

Adds/deletes/tests file descriptor `fd` to the read set.

```
void add_write(int fd);
void del_write(int fd);
void test_write(int fd);
```

Adds/deletes/tests file descriptor `fd` to the write set.

```
void add_except(int fd);
```

`void del_except(int fd);`

`void test_except(int fd);`

Adds/deletes/tests file descriptor `fd` to the except set.

`bool select();`

`bool select(long sec, long usec);`

Returns true if there are any ready descriptors in the three descriptor sets. If the time values `sec` and `usec` for the `timeout` are set, false is returned if the timer expires.

11 EZD Interface Classes

The following classes provide an interface to DEC's Easy Draw application. Easy Draw is an application written in scheme which alows easy drawing on the X 11 Window system. The CNCL simulation programs communicate via UNIX pipes with Easy Draw.
These classes use Xcolors and Xfonts. A list of the colors you can find at `/usr/global/lib/X11` in the file rgb.txt, for a list of fonts use the command `xlsfonts`. Only the color 'clear' is added to allow transparent objects in the drawing.
Additionally to the colors some stipple patterns are defined at EZD. They are called *s0*, *s1*, ... , *s16*, where the number is representing the amount of drawn pixels (in the defined color) in a square of 4-by-4. However, this option is ignored when postscript files are written.

You can find additional information about ezd at the man pages.

11.1 EZD — Base Class for EZD Graphic Objects

SYNOPSIS

```
#include <CNCL/EZD.h>
```

TYPE

`CN_EZD`

BASE CLASSES

CNNamed

DERIVED CLASSES

EZDDrawing, EZDObject, EZDPushButton, ...

RELATED CLASSES

None

DESCRIPTION

`EZD` is the base class for EZD graphic objects. It manages the access to EZD's I/O streams and provides the EZD drawing primitives. For additional information about EZD, refer to the manual pages of EZD.
Constructors:

```
EZD();
EZD(CNParam *param);
EZD(const CNStringR name);
```

Initializes and opens the pipe to the EZD process.

In addition to the member functions required by CNCL, `EZD` provides:

```
static ostream & out();
```

Connects the stream from the program to the EZD process.

```
static istream & in();
```

Connects the stream from the EZD process to the program.

```
static void draw_point(int x, int y, const CNStringR col)
```

Draws a point at position (x,y) in color `col`.

```
static void draw_line(int x1, int y1, int x2, int y2, const CNStringR col, int width = -1);
static void draw_dash_line(int x1, int y1, int x2, int y2, const CNStringR col, int width = -1);
```

Draws a (dashed) line from point `(x1,y1)` to point `(x2,y2)` in color `col` and `width` pixels wide. If `width` is <= 0, the minimum of one pixel is used for it.

`static void draw_arc(int x, int y, int w, int h, int a1, int a2, const CNStringR col, int width = -1);`

Draws an unfilled arc. `(x,y)` are the minimum coordinates of the rectangle defining the arc. The arc's rectangle size is width `w` by height `h`. The arc starts at `a1` degrees and spans `a2` degrees - all angles are measured in the positive mathematical sense, the positive x-axis is the reference axis. The drawing color is `col` and the line width is `width`. If width <= 0 EZD's default value (1 pixel) is used.

`static void draw_fill_arc(int x, int y, int w, int h, int a1, int a2, const CNStringR col);`

Draws a filled arc (end-point to end-point of the arc). `(x,y)` are the minimum coordinates of the rectangle defining the arc. The arcs rectangle size is width w by heigth h. The arc starts at a1 degrees and spans a2 degrees - all angles measured in positive mathematical sense, the positive x-axis is the reference axis. The drawing color is `col`.

`static void draw_pie_arc(int x, int y, int w, int h, int a1, int a2, const CNStringR col);`

Draws an arc (the arc is closed by drawing a line from each arc end point to the center of the rectangle defining the arc). `(x,y)` are the minimum coordinates of the rectangle defining the arc. The arcs rectangle size is width w by heigth h. The arc starts at a1 degrees and spans a2 degrees - all angles measured in positive mathematical sense, the positive x-axis is the reference axis. The drawing color is `col`.

`static void draw_rectangle(int x, int y, int w, int h, const CNStringR col, int width = -1);`

Draws an unfilled rectangle. `(x,y)` are the minimum coordinates. The size is defined by its width w and its heigth h. The drawing color is `col` and the line width is `width`. If width is <= 0 EZD's default value (1 pixel) is used.

`static void draw_fill_rectangle(int x, int y, int w, int h, const`

CNStringR col);

`static void draw_fill_rectangle(int x, int y, int w, int h, const CNStringR col, CNStringR stipple);`

Draws a filled rectangle. `(x,y)` are the minimum coordinates. The size is defined by its width w and its heigth h. The drawing color is `col`.

`static void draw_text(int x, int y, const CNStringR text, const CNStringR col, const CNStringR font);`

`static void draw_text(int x, int y, int w, int h, const CNString align, const CNStringR text, const CNStringR col, const CNStringR font);`

Writes `text` in the current drawing. `(x,y)` are the minimum coordinates of the rectangle containing the text. `col` and `font` describe color and font of the text. The optional width w, heigth h and alignment information describe the size of this rectangle and how the text has to be positioned in it. If the text does not fit in, no text is displayed at all.

`static void draw_bitmap(int x, int y, CNStringR filename, CNStringR color1, CNStringR color2);`

`static void draw_bitmap(int x, int y, int w, int h, CNStringR filename, CNStringR color1, CNStringR color2);`

Draws the bitmap for the file `filename` at position `x,y` in width `w` and height `h`. The bits that are on are drawn in `color1`, those which are off in `color2`.
NOTE: The file must contain an X11 bitmap, a monochrome portable bitmap (PBM, type P1), a gray scale portable bitmap (PGM, type P2), or a color portable bitmap (PPM, type P3).

`static void draw_now();`

Causes any buffered changes to be drawn immideately.

`static void draw_clear();`

Clears the current drawing.

`static void pause(int msec);`

Forces out any buffered changes and stops reading commands from stdin until either `msec` milliseconds pass or some event action issues a quit command.

`static CNString event();`

Reads event from pipe in EZD.

`static bool test_event();`

Returns true if there are ready descriptors in the I/O descriptor sets (see also: CNSelect or man pages to "select(2)" system call)

`static void set_scale(const float xscale_x, const float xscale_y, const int xorigin_x, const int xorigin_y);`

`static int x2pix(const float x);`

`static int y2pix(const float y);`

In some applications it is necessary to map a drawing with a cartesian coordinate system onto a window which uses pixels. `set_scale` specifies the scale factor (default: scale (x,y) = (1.0,1.0), origin (x,y) = (0,0)). `x2pix` and `y2pix` evaluate the coordinate transformation for the given values `(x,y)` by the formula `(value * scale + origin)`, rounded to an integer.

`static void print_window(CNStringR winname, CNStringR dateiname);`

Prints the window named `winname` to disk. The file `dateiname` is in postscript format.

`static void save_drawing();`

`static void restore_drawing();`

Saves/restores the current drawing.

11.2 EZDObject — Interface to EZD Object

SYNOPSIS

```
#include <CNCL/EZDObject>
```

TYPE

`CN_EZDOBJECT`

BASE CLASSES

EZD

DERIVED CLASSES

EZDQueue, EZDServer, EZDText, EZDTimer

RELATED CLASSES

EZDDrawing, EZDPushButton, EZDWindow

DESCRIPTION

`EZDObject` provides an interface to the objects of Easy Draw. Thus the drawings in all derived classes are handled as objects.
Constructors:

```
EZDObject();
EZDObject(CNParam *param);
EZDObject(int x, int y);
EZDObject(const CNString &name, int x, int y);
```

Initializes the EZD-object with either name "obj" or `&name`. `(x,y)` are the object's minimum coordinates.

Public accessible members:

`int x();` Returns the `EZDObject`'s x coordinate.

```
int x(int vx);
```

Returns the `EZDObject`'s old x coordinate and changes it to `vx`.

`int y();` Returns the `EZDObject`'s y coordinate.

```
int y(int vy);
```

Returns the `EZDObject`'s old y coordinate and changes it to `vy`.

In addition to the member functions required by CNCL, `EZDObject` provides:

```
void start();
void start(const CNString &object_name);
void end();
```

"Start/End object drawing" command. All drawing commands between `start()` and `end()` draw the named object in the current drawing. If the named object already exists in the current drawing, it is replaced by the new one. If no drawing commands are specified between `start()` and `end()`, the drawing still exists, but with no graphical representation.

```
void delete_obj(const CNString &draw_name = "draw");
```

Deletes the named drawing in the current object.

```
int get_lastxb(void);
int get_lastyb(void);
int get_lastxe(void);
int get_lastye(void);
```

Returns the object's last positions.

```
void point(int x, int y, const CNString &col);
void line(int x1, int y1, int x2, int y2, const CNString &col, int width = -1);
void arc(int x, int y, int w, int h, int a1, int a2, const CNString &col, int width = -1);
void fill_arc(int x, int y, int w, int h, int a1, int a2, const CNString &col);
void pie_arc(int x, int y, int w, int h, int a1, int a2, const CNString &col);
void rectangle(int x, int y, int w, int h, const CNString &col, int width = -1);
void fill_rectangle(int x, int y, int w, int h, const CNString &col, int width = -1);
void text(int x, int y, const CNString &text, const CNString &col, const CNString &font);
void text(int x, int y, int w, int h, const CNString align, const CNString &text,
const CNString &col, const CNString &font);
void bitmap(int x, int y, int w, int h, const CNString &filename, const CNString
&color1 = "black", const CNString &color2 = "");
void bitmap(int x, int y, const CNString &filename, const CNString &color1 = "black",
const CNString &color2 = "");
```

Basic drawing commands (exact description see section `EZD` or EZD's man pages). The command's `(x,y)` coordinates determine the position in the object. For the position in the current drawing the object's coordinates `(ox,oy)` are added automatically.

```
virtual void redraw();
```

Virtual redraw function (has to be implemented by derived objects).

11.3 EZDDrawing — Interface to EZD Drawings

SYNOPSIS

```
#include <CNCL/EZDDrawing.h>
```

TYPE

`CN_EZDDRAWING`

BASE CLASSES

EZD

DERIVED CLASSES

None

RELATED CLASSES

EZDObject, EZDPushButton, EZDWindow

DESCRIPTION

`EZDDrawing`
Constructors:

```
EZDDrawing();
EZDDrawing(CNParam *param);
EZDDrawing(const CNString &name);
EZDDrawing(const CNString &name, const int x, const int y, const int w, const int h);
```

Sets this drawing to the current drawing. Its name is either `name` or (by default) "draw".

Additional functions :

```
void set();
```

sets this drawing to the current drawing.

11.4 EZDPushButton — Interface to EZD Push-Button

SYNOPSIS

```
#include <CNCL/EZDPushButton>
```

TYPE

`CN_EZDPUSHBUTTON`

BASE CLASSES

EZD

DERIVED CLASSES

None

RELATED CLASSES

EZDObject, EZDDrawing, EZDWindow

DESCRIPTION

`EZDPushButton` defines a push-button in the current drawing. The button is defined by it's minimum `(x,y)` coordinates, it's width `w` by heigth `h` rectangle size and it's `CNNamed` object name. The default values are x = y = 0, w = 50, h = 20, name = "button".The button is drawn as a filled white rectangle with a black border. The button's `text` is written in black characters in the center of the button.
When the mouse enters a button with an action, the border is thickened. When the mouse button 1 is pressed, the button colors are reversed. Releasing mouse button 1 the push-button's `action` is taken and it is drawn as before.
NOTE: If you leave the button's area with your mouse during pressing/releasing the mouse button no `action` will be taken.
Constructors:

```
EZDPushButton();
```

```
EZDPushButton(CNParam *param);
```

```
EZDPushButton(const CNString &name, int vx, int vy, int vw, int vh, const CNString &vtext);
```

```
EZDPushButton(const CNString &name, int vx, int vy, int vw, int vh, const CNString &vtext,
CNString &vaction);
```

> Initializes the push-button in the current drawing. `name` is the object's name (default: "button"), `(vx, vy)` the minimum (x,y) coordinates (default: (0,0)), `(vw, vh)` the button's width and heigth (50,20), `vtext` the describing text in the current drawing ("button") and `vaction` the action taken when the button is pressed ("log-event").

In addition to the member functions required by CNCL, `EZDPushButton` provides:

`void set_text(const CNString &t);`

Sets the push-button's text to t.

11.5 EZDWindow — Interface to EZD Window

SYNOPSIS

```
#include<CNCL/EZDWindow>
```

TYPE

`CN_EZDWINDOW`

BASE CLASSES

EZD

DERIVED CLASSES

None

RELATED CLASSES

EZDDrawing, EZDObject, EZDPushButton

DESCRIPTION

With EZD a drawing can only be displayed when it is mapped into a window. For this, `EZDWindow` creates named windows and maps drawings into it. The window is always displayed in the upper left corner of the screen and is `w_width` pixels wide and `w_heigth` pixels high. The background color is white and the foreground color is black.
Note: If the named window already exists, the old window is deleted. Windows are only visible if drawings are displayed in them.
Constructors:

```
EZDWindow();
EZDWindow(CNParam *param);
EZDWindow(const CNString &name, int w, int h);
EZDWindow(int w, int h);
```

Creates the window named `name` (default: "win"), `w` pixels wide (default: 200) and `h` pixels high (default: 200).

Destructor:

```
~EZDWindow();
```

Deinitilizes (deletes) the current window.

In addition to the member functions required by CNCL, `EZDWindow` provides:

```
int heigth() const;
int width() const;
int wherex() const;
int wherey() const;
```

Returns the width, heigth or (x,y) position of the current window.

```
void overlay(EZDDrawing *d);
void underlay(EZDDrawing *);
```

Overlays/undelays the drawing `d` over the existing drawings in the current window.

```
void scale_drawing(EZDDrawing *draw, const int origin_x, const int origin_y,
const float d_scale_x, const float d_scale_y, const int scale_linewidth);
```

Maps the drawing`draw` with a cartesian coordinate system onto the current window. The `origin_(x,y)` parameters specifie the size of the origin coordinate system, `d_scale_(x,y)` and `scale_linewidth` are the scaling factors.

```
void set_zoom(EZDDrawing *, const float factor);
```

Zooms the drawing by `factor`.

```
void set_norm(EZDDrawing *);
```

Sets the drawing back to normalsize.

```
void set_auto_resize(const int nargs, EZDDrawing * ...);
```

Resizes all drawings of the argument. `nargs` is the number of arguments.

```
void delete_drawing(EZDDrawing *d);
```

Deltes the drawing `d` in the current window.

```
void print_window(const CNString &filename);
```

Prints the current window as a postscript file on disk.

11.6 EZDDiagWin — Extra window with x-y diagram

SYNOPSIS

```
#include <CNCL/EZDDiagWin.h>
```

TYPE

`CN_EZDDIAGWIN`

BASE CLASSES

EZD

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

`EZDDiagWin` craetes a named window including a diagram. Actually there are three different styles (as an enum) implemeted in this class:

- DOT : draws the diagram in form of dots
- LINE : draws a line
- HISTO: draws a histogram

Constructors:

```
EZDDiagWin(CNStringR n, CNStringR title, int w, int h, int st, CNStringR cb, Style s=LINE);
EZDDiagWin();
EZDDiagWin(CNParam *param);
EZDDiagWin(int w, int h, Style s=LINE);
EZDDiagWin(CNStringR n, int w, int h, Style s=LINE);
EZDDiagWin(CNStringR n, CNStringR title, int w, int h, Style s=LINE);
EZDDiagWin(CNStringR n, CNStringR title, int w, int h, int st, CNStringR cd, CNStringR cb,
Style s=LINE);
```

Initializes a named `EZDDiagwin`. `title` is the title of the window, `w` and `h` the window's width and height. `st` is the size of a step between two values (default: 1), `cd` is the color of the diagram, `cb` the color of the bar (out-of-range display), `s` the style of the diagram.

In addition to the member functions required by CNCL, `CNEZDDiagWin` provides:

`void style(Style s);`

Changes the style to `s`.

`void add(int v);`

Adds a new value `v` at the current x-position to the diagram.

`void clear();`

Clears the diagram window.

`void set();`

Sets current drawing to diagram window drawing.

11.7 EZDTextWin — ezd window for easy text display

SYNOPSIS

```
#include <CNCL/EZDTextWin.h>
```

TYPE

`CN_EZDTEXTWIN`

BASE CLASSES

EZD

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

`EZDTextWin` provides an easy possibility for a text display in an EZD window.
Constructors:

```
EZDTextWin(CNParam *param);
EZDTextWin(int r=ROWS, int c=COLS, int ix=INCX, int iy=INCY);
EZDTextWin(int r, int c, int ix, int iy, CNStringR f);
EZDTextWin(CNStringR n, int r=ROWS, int c=COLS, int ix=INCX, int iy=INCY);
EZDTextWin(CNStringR n, int r, int c, int ix, int iy, CNStringR f);
EZDTextWin(CNStringR n, CNStringR t, int r=ROWS, int c=COLS, int ix=INCX, int iy=INCY);
EZDTextWin(CNStringR n, CNStringR t, int r, int c, int ix, int iy, CNStringR f);
```

Initializes `EZDTextWin`. `r` is the number of rows, `c` the number of columns. `ix` and `iy` are the x-size of one column / y-size of one row. Thus the total size of the window is (`c` • `ix`) by (`r` • `iy`). The strings are the window's name (`n`), it's title (`t`) and the font (`f`). No text is displayed in the window yet.The constant default values are:
ROWS : 24, COLS : 80, INCX : 9, INCY : 15

In addition to the member functions required by CNCL, `EZDTextWin` provides:

```
int width() const;
int height() const;
```

Returns the window's width or height.

`int cols() const;`

`int rows() const;`

Returns the number of rows or cols in the window.

`int incx() const;`

`int incy() const;`

Returns the x-size of a column or the y-size of a row.

`int row_to_y(int r) const;`

`int col_to_x(int c) const;`

Returns the y-/x-coordinate of the r-th row/ c-th column.

`void clear();`

Clears the whole text window, painting everything white.

`void clear(int r, int c, int l);`

Clears a window area. `r` is the row and `c` the column wich is cleared, (`l` • `incx`) the horizontal length in this area wich is painted white.

`void set();`

Sets the current drawing to text window drawing.

`void hline(int r, int c, int l);`

`void vline(int r, int c, int l);`

`void add(int r, int c, CNStringR s);`

`void add(int r, int c, CNStringR s, CNStringR f);`

Draws the text `s` in row `r`, column `c`. Possible old text at this position (length of string s) is wiped out. `f` is the font.

11.8 EZDDiag — x-y diagram as an EZDObject

SYNOPSIS

```
#include <CNCL/EZDDiag.h>
```

TYPE

`CN_EZDDIAG`

BASE CLASSES

EZDObject

DERIVED CLASSES

None

RELATED CLASSES

EZDDiagWin

DESCRIPTION

`EZDDiag` creates a diagram as an ezd object. Actually there are three different styles (as an enum) implemeted in this class:

- DOT : draws the diagram in form of dots
- LINE : draws a line
- HISTO: draws a histogram

Constructors:

```
EZDDiag();
EZDDiag(CNParam *param)
EZDDiag(int w, int h, Style s=LINE);
EZDDiag(int w, int h, int x, int y, Style s=LINE);
EZDDiag(const CNString &name, int w, int h, int x, int y, Style s=LINE);
EZDDiag(int w, int h, int x, int y, Style s, CNString cd, CNString cb);
EZDDiag(const CNString &name, int w, int h, int x, int y, Style s, CNString cd, CNString cb);
```

Initializes `EZDDiag` at the position `(x,y)`, default (0,0), of the current drawing as a named ezd object. `w` (default: 200) and `h` (100) are the object's width and heigth. `cd` ("black") and `cb` ("red") are color-settings for the diagram and the bar which is drawn if the value is out of the given range - e.g. negative values or values > 100 at the default setting. The possible styles `s` are described above. The object is drawn frameless, but a frame can be added, see the functions below.

In addition to the member functions required by CNCL, `EZDDiag` provides:

`void style(Style s);`

Sets the style of the diagram to `s`.

`void set_color_draw(CNString c);`

Sets the drawing color to `c`.

`void set_color_bar(CNString c);`

Sets the color of the bar (out-of-range display) to `c`.

`void add(int v);`

Adds the value `v` to the diagram.

`void clear();`

Clears the diagram.

`virtual void redraw();`

Redraws the diagram.

`void set_frame();`

Sets a frame to the diagram.

11.9 EZDBlock — Block with small rectangles for bit display

SYNOPSIS

```
#include <CNCL/EZDBlock.h>
```

TYPE

`CN_EZDBLOCK`

BASE CLASSES

EZDObject

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

`EZDBlock` draws a block of several square rectangles. Each rectangle's side is `bsize` long, so that the total size of the block is determined by the number of rows and cols. Each rectangle's color is representing the state of a bit, `black` if the bit is turned on and `white` if turned off. The frame of all rectangles is 1 pixel by default.
Constructors:

```
EZDBlock();
EZDBlock(CNParam *param);
EZDBlock(const CNString &name, int x, int y, int r, int c, int b);
```

Initializes the `EZDBlock` (does not draw it yet). `&name` is the `EZDObjects` name, `(x,y)` its minimum coordinates (default (0,0)). The block is made out of r by c square rectangles, each side b long. At the initialzation all bits are assumed to be set off (drawn white).

In addition to the member functions required by CNCL, `EZDBlock` provides:

```
virtual void redraw();
```

Draws / redraws the whole block.

```
void on(int b);
void off(int b);
```

Turns bit on(black)/off(white) and sets the color for it; redraws the (whole) block.

11.10 EZDPopUp — Interface to EZD popup menu

SYNOPSIS

```
#include <CNCL/EZDPopUp.h>
```

TYPE

`CN_EZDPOPUP`

BASE CLASSES

EZDObject

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

`EZDPopUp` is the interface to ezd's popup menus. It defines a named scheme that is called as the action on a button down event to display a popup menu. The scheme's functions, displayed names and all other parameters are stored in the struct `MENU`. When button 1 is pressed down, the menu comes up. Releasing the button executes the action associated with the current menu entry. Moving the mouse outside the menu without releasing the button will cause the menu to disappear, no action is executed. For additional information see the ezd manual.
Constructors:

```
EZDPopUp();
EZDPopUp(CNParam *param);
EZDPopUp(const int nmenu, const MENU *menu ...);
```

Initializes the conection to ezd's popup with a variable size of parameters. `nmenu` is the total number of menu items, `MENU` a struct descibed below.

In addition to the member functions required by CNCL, `EZDPopUp` provides:

```
typedef struct menue MENU;
```

`MENU` is a struct of :

`int posx, posy`

minimum (x,y) coordinates of the menu

`int height`

height of the menu

`CNString title`
: menu title, used as name of the scheme

`CNString t_color`
: title color

`CNString b_color, f_color`
: color of the background and the frame

`CNString* inhalt`
: menu entries

`CNString* scheme_func`
: scheme functions

`int anz`
: total number of menu entries

  Take note that the [i]th entrie and the [i]th function are associated with each other. If `anz` is larger than the correct number the program will terminate, if it is smaller, all entries after that number are set inactive.

11.11 EZDQueue — Graphical Representation of a Queue

SYNOPSIS

```
#include <CNCL/EZDQueue.h>
```

TYPE

`CN_EZDQUEUE`

BASE CLASSES

EZDObject

DERIVED CLASSES

None

RELATED CLASSES

EZDServer

DESCRIPTION:

`EZDQueue` is the graphical representation of a queue. This drawing is handled as an `EZDObject`, in which x is the most left and y the middle axis' coordinate. The bar, itself an `EZDObject`, is representing the amount of tasks waiting inside the queue.
Constructors:

```
EZDQueue();
EZDQueue(Param *param);
EZDQueue(int x, int y, int width = WIDTH, int heigth = HEIGTH, int tail = TAIL);
EZDQueue(const CNString &name, int x, int y, int width = WIDTH, int heigth = HEIGTH,
int tail = TAIL);
```

Initializes an `EZDQueue` as an `EZDObject` (default object name : "queue") inside the current drawing at the coordinates `(x,y)` (default value (0,0)). Width (100 pixels) and heigth (20 pixels) are the queue body's rectangle size measured in pixels, tail (20 pixels) the tail's length.

In addition to the member functions required by CNCL, `EZDQueue` provides:

```
int left() const;
int rigth() const;
int upper() const;
```

```
int lower() const;
```

Returns the left, right, upper or lower coordinate of the object queue inside the current drawing.

```
virtual void redraw();
```

Redraws this queue in the current drawing.

```
int length() const;
int get_length() const;
void length(int l);
void set_length(int l);
```

Get/set length of the queue-bar.

```
void color(const CNString &c);
```

Sets the color of the queue-bar to `c` and redraws the queue.

11.12 EZDServer — Graphical Representation of a Server

SYNOPSIS

```
#include <CNCL/EZDServer.h>
```

TYPE

`CN_EZDSERVER`

BASE CLASSES

EZDObject

DERIVED CLASSES

None

RELATED CLASSES

EZDQueue

DESCRIPTION:

`EZDServer` is the graphical representation of a server. This drawing is handled as an `EZDObject`, in which x is the most left and y the middle axis' coordinate.
Constructors:

```
EZDServer();
EZDServer(CNParan *param);
EZDServer(int x, int y , int radius = RADIUS);
EZDServer(const CNString &name, int x, int y, int radius=RADIUS);
```

Initializes an `EZDServer` as an `EZDObject` (default object name : "server") inside the current drawing at the coordinates `(x,y)` (default value (0,0)) with `radius` (default 20 pixels).

In addition to the member functions required by CNCL, `EZDServer` provides:

```
virtual void redraw();
```

Redraws the server in the current drawing.

```
void color(const CNString &c);
```

Sets the color of the server to `c` and redraws the server in the current drawing.

11.13 EZDText — EZD Object with Text

SYNOPSIS

```
#include <CNCL/EZDText.h>
```

TYPE

`CN_EZDTEXT`

BASE CLASSES

EZDObject

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION:

`EZDText` writes text into a clear rectangle. The rectangle is handled as an `EZDObject`, its size is automatically as large as necessary to hold the text in the given `font` and `(x,y)` are its minimum coordinates in the current drawing. Note that the `font` name is specified as a string in X font name terminology.
Constructors:

```
EZDText();
EZDText(CNParam *param);
EZDText(int x, int y, const CNString &text);
EZDText(const CNString &name, int x, int y, const CNString &text);
EZDText(int x, int y, const CNString &text, const CNString &font);
EZDText(const CNString &name, int x, int y, const CNString &text, const CNString &font);
```

Initializes the `EZDObject` and displays the `text` in the current drawing at the `(x,y)` coordinates. If no `EZDObject` name is chosen it is called "text".

In addition to the member functions required by CNCL, `EZDText` provides:

```
virtual void redraw();
```

Redraws the `text` in the current drawing.

```
void set_text(const CNString &t);
void set_text_val(int x);
```

```
void set_text_val(const CNString &t, int x);
void set_text_val(double x);
void set_text_val(const CNString &t, double x);
```

Creates `text` out of text, integer or double and displays it in the current drawing.

11.14 EZDTimer — Graphical Representation of a Timer

SYNOPSIS

```
#include <CNCL/EZDTimer.h>
```

TYPE

`CN_EZDTIMER`

BASE CLASSES

EZDObject

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

`EZDTimer` draws an `EZDObject` representing a timer. It is realized as a filled pie arc inside of an oval. The `(x,y)` coordinates are the center coordinates of the object in the current drawing. The timer starts (by default) at 270 degrees (12 o'clock position) as an offset value. The pie dimension angle (in degrees) is the pie's size. The default colors are black on white backround. Please note that the timer doesn't count automatically, each change of the shown value has to be programmed.
Constructors:

```
EZDTimer();
EZDTimer(CNParam *param);
EZDTimer(int x, int y, int radiusx=RADIUSX, int radiusy=RADIUSY);
EZDTimer(const CNString &name, int x, int y, int radiusx=RADIUSX, int radiusy=RADIUSY);
```

Initializes the timer as an `EZDObject`. The default object's name is "Timer". Radiusx and radiusy are the oval's radius in (x,y) direction, measured in pixels (default:(5,5))

In addition to the member functions required by CNCL, `EZDTimer` provides:

```
int left();
int right();
int upper();
int lower();
```

Returns the oval's most left/right/upper/lower layout coordinates in pixels.

`virtual void redraw();`

Redraws the timer in the current drawing.

`int offset_angle()`

`int get_offset_angle()`

Returns the timer's current offset angle.

`void offset_angle(int a);`

`void set_offset_angle(int a);`

Sets the timer's offset angle to `a` degrees and redraws the pie.

`int angle();`

`int get_angle();`

Returns the timer's pie dimension angle.

`void angle(int a);`

`void set_angle(int a);`

Sets the timer's pie dimension angle to `a` degrees and redraws the pie.

`void color(const CNString &c);`

Sets the timer's color to `c` and redraws the timer.

`void activate();`

`void deactivate();`

Activates/deactivates an additional oval in 4 pixels distance to the timer.

`void active_color(const CNString &c);`

Sets the additional oval's color to `c` but does not redraw it.

12 Fuzzy classes

CNCL provides a set of classes for building fuzzy inference engines. Available are fuzzy sets, fuzzy variables, and a inference engine based on fuzzy rules. The membership values are normalized (by default), but can be changed to max and min values in different classes. All fuzzy sets and functions are realized with crisply defined membership values (type 1 fuzzy sets and functions). Currently prod-min inference is used and a center-of-gravity output defuzzification. This will be extended in a future release, providing different operators for aggregation, inference, and accumulation. Up to now two different fuzzy set representations are implemented: LR-representation and arrays.

For more information about fuzzy logic see:
Zimmermann,. H.-J. [1991]. Fuzzy Set Theory And Its Applications. Kluwer Accademic Publishers.
Additionally, the FAQ of the newsgroup "comp.ai.fuzzy" should be recommended in this context.

12.1 CNFClause — Clause of a fuzzy rule

SYNOPSIS

```
#include <CNCL/FClause.h>
```

TYPE

`CN_FCLAUSE`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

DESCRIPTION

A `CNFClause` is basically a structure containing a fuzzy variable / fuzzy value pair. The members

```
CNFVar *var; // Linguistic variable
CNFSet *value; // Linguistic value
```

are public accessible.

Constructors:

`CNFClause();`

`CNFClause(CNParam *param);`

`CNFClause(CNFVar *xvar, CNFSet *xset);`

`CNFClause(CNFVar &xvar, CNFSet &xset);`

Initializes a `CNFVar` and a `CNFSet` to a `CNFClause`.

In addition to the member functions required by CNCL, `CNFClause` provides:

`void print_clause(ostream &strm, bool lhs) const;`

Prints the public accessible variables on the output stream `&strm`.

12.2 CNFVar — Fuzzy variable

SYNOPSIS

```
#include <CNCL/FVar.h>
```

TYPE

`CN_FVAR`

BASE CLASSES

CNNamed

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

This class creates a fuzzy variable and its corrosponding linguistic variable. For example, the linguistic variable "temperature" migth be programmed this way:
CNFVar temp("Temperature", 0, 100);
enum { COLD, MEDIUM, WARM, HOT };
temp.add_value_set(COLD, new CNFSetTrapez("cold", 0, 10, 0, 10));
temp.add_value_set(MEDIUM, new CNFSetTrapez("medium", 20, 20, 10, 10));
temp.add_value_set(WARM, new CNFSetTrapez("warm", 30, 40, 10, 5));
temp.add_value_set(HOT, new CNFSetTrapez("hot", 40,100, 10, 0));
Here trapezium functions have been assumed as membership function, but all other kinds of (implemented) functions are possible.

Constructors:

```
CNFVar(CNParam *param);
CNFVar(double min = 0, double max = 1);
CNFVar(const CNString &xname, double min = 0, double max = 1);
```

Initializes the `CNFVar`.

In addition to the member functions required by CNCL, `CNFVar` provides:

```
double value( double x );
double set_value( double x );
```

Sets the (non fuzzy) variable value to `x` and returns the old value.

```
double value() const
```

`double get_value()const;`

Returns the (non-fuzzy) value.

`CNFSet* fuzzy_value( CNFSet* x );`

`CNFSet* set_fuzzy_value( CNFSet* x );`

Sets the fuzzy variable value (fuzzy set) to `x` and returns the old value.

`CNFSet* fuzzy_value() const;`

`CNFSet* get_fuzzy_value() const;`

Returns the fuzzy value (fuzzy set).

`double xmin() const;`

`double get_xmin() const;`

`double xmax() const;`

`double get_xmax() const;`

Returns the minimum and maximum values of the variable's range.

`void add_value_set(CNFSet &fset);`

`void add_value_set(CNFSet *fset);`

`void add_value_set(int i, CNFSet &fset);`

`void add_value_set(int i, CNFSet *fset);`

Adds the fuzzy set `fset` to the array of affiliated sets, either at position `i` (overwritting) or at the end (extending).

`CNFSet *get_value_set(int i);`

Returns the fuzzy set at array position `i`.

`double get_membership( int i);`

Gets the membership value of the fuzzy set at array position `i`.

`void print_membership();`

Computes the membership value for all fuzzy sets using the current value and prints the result.

12.3 CNFRule — Fuzzy Rule

SYNOPSIS

```
#include <CNCL/FRule.h>
```

TYPE

`CN_FRULE`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

Constructors:

```
CNFRule();
CNFRule(CNParam *param);
```

Initializes a `CNFRule` variable.

In addition to the member functions required by CNCL, `CNFRule` provides:

```
void add_lhs(CNFClause *clause);
void add_lhs(CNFClause &clause);
void add_rhs(CNFClause *clause);
void add_rhs(CNFClause &clause);
```

Adds a `CNFClause` to the LHS or the RHS rules.

```
int get_n_lhs() const;
int get_n_rhs() const;
```

Gets the total number of LHS/RHS clauses.

```
CNFClause* get_lhs(int i) const;
CNFClause* get_rhs(int i) const;
```

Returns the LHS/RHS clause from array-position `i`.

```
double certainty(double x);
double set_certainty(double x);
```

Sets the certainty to `x` and returns the old certainty.

`double certainty()const;`

`double get_certainty()const;`

Returns the certainty.

`double aggregate_value() const;`

Returns the current value of the aggregation.

`double aggregate()`

Aggregates the LHS rule inputs and returns the result.

12.4 CNFRuleBase — Rule base and Fuzzy inference engine

SYNOPSIS

```
#include <CNCL/FRuleBase.h>
```

TYPE

`CN_FRULEBASE`

BASE CLASSES

CNNamed

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

Constructors:

```
CNFRuleBase();
CNFRuleBase(CNParam *param);
CNFRuleBase(const CNString &name);
```

Initializes the `CNFRuleBase` with `name` as the object's name.

In addition to the member functions required by CNCL, `CNFRuleBase` provides:

```
void add_rule(CNFRule *rule);
void add_rule(CNFRule &rule);
```

Adds rule to base.

```
void add_in_var(CNFVar *rule);
void add_in_var(CNFVar &rule);
void add_out_var(CNFVar *rule);
void add_out_var(CNFVar &rule);
```

Adds input/output variables to the base.

```
int get_n_rules()const;
int get_n_in_vars()const;
int get_n_out_vars()const;
```

Returns the number of rules/input variables/output variables in this base.

`int resolution()const;`

Returns the output fuzzy set resolution.

`void inference(CNFVar *var, CNFSet *set , double match, CNFSetArray &res);`

Computes the inference set for variable `var` and value `set`, using value `match` of LHS aggregation. The result is stored in the array fuzzy set `res`.

`void evaluate(CNFVar *var, CNFSetArray &res);`

Computes output set for variable `var`, combining the `inference()` results of all rules. The result is stored in the array fuzzy set `res`.

`void aggregate_all();`

Aggregates all rules.

`void evaluate_all();`

Evaluates all variables.

`void defuzzy_all();`

Defuzzifies all output variables.

`void debug_rules((ostream &strm=cout, int lvl=0);`

Debugs the output on `&strm` for fuzzy rules.

12.5 CNFSet — Fuzzy set abstract base class

SYNOPSIS

```
#include <CNCL/FSet.h>
```

TYPE

`CN_FSET`

BASE CLASSES

CNNamed

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

`CNFSet` is the abstract base class for fuzzy set realizations.

Constructors:

```
CNFSet(CNParam *param);
CNFSet(double min = 0, double max = 1);
CNFSet(const CNStringR xname, double min = 0, double max = 1);
```

Initializes a `CNFSet` with `xname` as the object's name. `min` and `max` determine the range of the membership values.

In addition to the member functions required by CNCL, `CNFSet` provides:

```
virtual double get_membership(double x) const = 0;
```

Gets the membership values for `x`.

```
virtual double center_of_gravity(double min, double max) const;
```

Computes the center of gravity for the defuzzyfication. (not implemented yet)

12.6 CNFSetArray — Fuzzy set based on array with membership values

SYNOPSIS

```
#include <CNCL/FSetArray.h>
```

TYPE

`CN_FSETARRAY`

BASE CLASSES

CNFSet

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

This class realizes a fuzzy set as an array. The array is containing the membership values and the index is its corrosponding x-value. Thus membership values of non-integers are interpolated.

Constructors:

```
CNFSetArray();
CNFSetArray(CNParam *param);
CNFSetArray(int sz, double min, double max);
```

Initializes `CNFSetArray`.

In addition to the member functions required by CNCL, `CNFSetArray` provides:

```
virtual double get_membership(double x)const;
```

Gets membership value for `x`.

```
virtual double center_of_gravity(double min, double max)const;
```

Computes the center of gravity.

```
double get(int i) const;
```

Gets the value from the array.

```
void put(int i, double x);
```

Puts value `x` into array at position `i`.

```
double &operator [] (int i);
```

Provides the direct access to the internal array.

```
int get_n() const;
```

Returns the size of the array.

12.7 CNFSetLR — Fuzzy set with L and R functions

SYNOPSIS

```
#include <CNCL/FSetLR.h>
```

TYPE

`CN_FSETLR`

BASE CLASSES

CNFSet

DERIVED CLASSES

CNFSetTrapez

RELATED CLASSES

None

DESCRIPTION

`CNFSetLR` provides the LR-representation of fuzzy sets. At this representation, the interval [xm1, xm2] is considered to have the maximum membership value (1 if normalized), the left (L) and right (R) approach is described by a shape function. Additionally, a left (alpha) and a right (beta) slope is defined, thus the whole membership function is:
L ((xm1 - x) / alpha) for x <= xm1
max (1 if normalized) for xm1 <= x <= xm2
R ((x - xm2) / beta) for x >= xm2

Constructors:

```
CNFSetLR();
```

```
CNFSetLR(CNParam *param);
```

```
CNFSetLR(double min, double max, double xm1, double xm2, double xalpha, double xbeta,
CNFuncType fL, CNFuncType fR);
```

```
CNFSetLR(double xm1, double xm2, double xalpha, double xbeta, CNFuncType fL, CNFuncType fR);
```

```
CNFSetLR(const CNStringR xname, double min, double max, double xm1, double xm2,
double xalpha, double xbeta, CNFuncType fL, CNFuncType fR);
```

```
CNFSetLR(const CNStringR xname, double xm1, double xm2, double xalpha, double xbeta,
CNFuncType fL, CNFuncType fR);
```

Initializes `CNFSetLR`. The possible variables and their default values are:
left/right maximum `(xm1,xm2)` (0,0), the left/right slope `(alpha,beta)` (0,0), the left/right function `(fL,fR)` (CNFuncLin,CNFuncLin) and the values for the named `CNFSet xname,min,max` with the according default settings of that class.

In addition to the member functions required by CNCL, `CNFSetLR` provides:

`typedef double (*CNFuncType)(double x);`

Function pointer to the functions for L/R use. Implemented functions are:

- double CNFuncExp (double x);
  exp(-x)
- double CNFuncExp2(double x);
  exp(-x^2)
- double CNFuncLin (double x);
  1-x
- double CNFuncSqr (double x);
  1-x^2
- double CNFuncHyp (double x);
  1/(1+x)
- double CNFuncHyp2(double x);
  1/(1+x^2)

`virtual double get_membership(double x) const;`

Gets the membership values for `x`.

`double get_m1() const;`

Returns the value `m1` (left maximum).

`double get_m2() const;`

Returns the value `m2` (right maximum).

`double get_alpha() const;`

Returns the value `alpha` (left slope).

`double get_beta() const;`

Returns the value `beta` (right slope).

12.8 CNFSetTrapez — Fuzzy set with trapezium function

SYNOPSIS

```
#include <CNCL/FSetTrapez.h>
```

TYPE

`CN_FSETTRAPEZ`

BASE CLASSES

CNFSetLR

DERIVED CLASSES

CNFSetTriangle

RELATED CLASSES

None

DESCRIPTION

`CNFSetTrapez` realizes a trapezium shaped membership function on the base of an LR fuzzy set.

Constructors:

```
CNFSetTrapez()
CNFSetTrapez(CNParam *param);
CNFSetTrapez(double xm1, double xm2, double xalpha, double xbeta);
CNFSetTrapez(double min, double max, double xm1, double xm2, double xalpha, double xbeta);
CNFSetTrapez(const CNStringR name, double xm1, double xm2, double xalpha, double xbeta);
CNFSetTrapez(const CNStringR name, double min, double max, double xm1, double xm2,
double xalpha, double xbeta);
```

Initialize `CNFSetTrapez` with `name`as the object's name. `xm1` is the x-value of the left maximum, `xm2` of the right one. `xalpha` is the left and `xbeta` the right slope. `min` and `max` are the function's minimum / maximum.

In addition to the member functions required by CNCL, `CNFSetTrapez` provides:

```
virtual double get_membership(double x)const;
```

Returns the membership value for `x`.

12.9 CNFSetTriangle — Fuzzy set with triangle function

SYNOPSIS

```
#include <CNCL/FSetTriangle.h>
```

TYPE

`CN_FSETTRIANGLE`

BASE CLASSES

CNFSetTrapez

DERIVED CLASSES

None

RELATED CLASSES

CNFNumTriangle

DESCRIPTION

`CNFSetTriangle` realizes a triangle shaped membership function.

Constructors:

```
CNFSetTriangle();
CNFSetTriangle(CNParam *param);
CNFSetTriangle(double xm, double xalpha, double xbeta);
CNFSetTriangle(double xmin, double xmax, double xm, double xalpha, double xbeta);
CNFSetTriangle(const CNStringR xname, double xm, double xalpha, double xbeta);
CNFSetTriangle(const CNStringR xname, double xmin, double xmax,double xm,
double xalpha, double xbeta);
```

Initializes the `CNFSetTriangle`.

In addition to the member functions required by CNCL, `CNFSetTriangle` provides:

```
virtual double get_membership(double x)const;
```

Returns the membership value for `x`.

```
double get_mean() const;
```

Returns the mean value `m1==m2` (maximum).

12.10 CNFNumTriangle — Fuzzy number (triangle)

SYNOPSIS

```
#include <CNCL/FNumTriangle.h>
```

TYPE

`CN_FNUMTRIANGLE`

BASE CLASSES

CNFSetTriangle

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

`CNFNumTriangle` provides triangle fuzzy set the necessarry mathematical operations and methods for defuzzification.

Constructors:

```
CNFNumTriangle();
CNFNumTriangle(CNParam *param);
CNFNumTriangle(double m, double a, double b);
CNFNumTriangle(const CNFSetTriangle &n);
```

Initializes `CNFNumTriangle`.

In addition to the member functions required by CNCL, `CNFNumTriangle` provides:

```
double center_of_gravity(double min, double max)const;
```

Returns the center of gravity.

```
double defuzzy();
```

Returns the defuzzyfied variable.

```
CNFNumTriangle sqr(CNFNumTriangle);
```

Squares a `CNFNumTriangle` set, using the extension principle.

```
CNFNumTriangle abs(CNFNumTriangle);
```

Returns the positiv `CNFNumTriangle`.

```
CNFNumTriangle &operator =(const CNFNumTriangle &n);
```

Sets this `CNFNumTriangle` to `n`.

`CNFNumTriangle operator +(CNFNumTriangle, CNFNumTriangle);`

`CNFNumTriangle operator -(CNFNumTriangle, CNFNumTriangle);`

`CNFNumTriangle operator *(CNFNumTriangle, double);`

`CNFNumTriangle operator *(double, CNFNumTriangle);`

Adds/ substracts two `CNFNumTriangle` sets or multiplies one with a double (extension principle).

13 Simulation of SDL protocols

CNCL provides a set of classes which support simulation of SDL protocols. There is also a method to translate SDL/PR notation into a CNCL simulation program automatically.

INTRODUCTION

The following CNCL classes and the Translator are a first attempt to ease the integration of SDL protocols into a given simulation environment. There should be a correct, nearly complete and hopefully efficient way to simulate SDL protocols. The automatic translation tool shall make it simpler to specify protocols using SDT to create SDL specifications and the sdl2cncl to translate them into a corresponding CNCL class system.
Since both the class code and the translation tool are largely untested now there might occur some problems with them. One should test the created class system versus the SDT simulation tool if possible. Please note that there are some restrictions to the specification when using the automatic translator. If there are problems other than these restrictions mail bug reports to `olly@dfv.rwth-aachen.de`.
Restrictions caused by the sdl2cncl:

- Block substructures, channel substructures, signal refinement etc. aren't allowed.
- Only one block level is supported.
- Signal routes (even in different blocks) must not have the same name.
- Comments in text symbols must not be longer than one line.
- No signal lists allowed.
- Use of identifier enumeration is not permitted; e.g. replace `a,b:Integer;` with `a:Integer; b:Integer;`.

These probably aren't all of the restrictions.
Additionally there are some problems which might disappear in the near future:

- SDL charstring expressions are detected by the scanner but not translated into C++ equivalents.
- The translator is unable to translate procedures although the C++ construct exists (untested).
- There are problems translating SDL arrays into CNArrays because the index variable might change (C++ arrays start with index 0 while SDL allows other values).
- Transfer of variables with signals isn't implemented in the translator but is possible with the SDL classes.
- Using self defined abstract datatypes requires additions to the scanner.
- The translator is case sensitive while SDT is not; so one has to make sure that identifier names, states, signals etc. are always written the same way.

Although these lists look very long one might find out that most of the above restrictions and problems can be avoided easily. The procedure problem will be resolved first because it seems to be the hardest restriction to SDL.
For more information about the sdl2cncl read Christoph Lampe, Entwicklung und Implementierung eines Uebersetzers zur automatischen Transformation in SDL formal spezifierter Mobilfunkprotokolle nach C++, Diplomarbeit,Lehrstuhl fuer Kommunikationsnetze,RWTH Aachen,1994.

For information about SDL read Belina,Hogrefe,Sarma,SDL with Applications from Protocol Specification,Prentice Hall International (UK),1991.
For information about SDT read TeleLOGIC Malmoe AB,Sweden,SDT 2.2 Reference Manual,1992

sdl2cncl

This program translates SDL/PR notation into an equivalent system of CNCL classes. About restrictions and problems see section Introduction. It reads a SDL/PR notation from `stdin` and writes out a system of classes into the current directory. While translating the actually parsed line number is displayed on the screen. Such it is possible to determine where an error occurs. `sdl2cncl` assumes a syntactical correct SDL/PR as its input. This is always the case when the SDL/PR is created by the SDT tool.

USE

```
sdl2cncl < specification.pr
```

SDL Simulation Example

There is an example of an SDL simulation (Demon Game) to be found in the SDL directory (/source/cncl/examples/sdl).

13.1 CNSDLManager — The SDL Machine

SYNOPSIS

```
#include <CNCL/SDLManager.h>
```

TYPE

`CN_SDLMANAGER`

BASE CLASSES

CNEventHandler

DERIVED CLASSES

CNSDLEnvironment

RELATED CLASSES

CNSDLProcess, CNSDLProcessData, CNSDLTimer, CNSDLSignal, CNSDLChannel

DESCRIPTION

`CNSDLManager` is the base class for a SDL system. It was first designed to handle only the communication of SDL processes but later took much more functionality. Nearly all basic SDL functions are provided by this class. A simple method to handle dynamic SDL is included, too. `CNSDLManager` uses `CNSDLTimer`, `CNSDLProcessData` and `CNSDLChannel` to perform some of its functions.

Constructors:

```
CNSDLManager();
CNSDLManager(CNParam *param);
CNSDLManager(int sys, CNEventScheduler *sim);
```

Initializes the SDL Manager. Parameters are system number (starting with 2) and a pointer to the scheduler.

In addition to the member functions required by CNCL, `CNSDLManager` provides:

```
typedef int PId;
```

Process ID-number.

```
typedef int TimerType;
typedef int SignalType;
typedef int ChannelNo;
```

Channel's number.

```
typedef int ProcessType;
```

```
virtual void event_handler(const CNEvent *ev);
```

This function is responsible for the correct delivery of SDL signals (including handling the save list). It must not be redefined in derived classes.

```
void save(SignalType = 0);
```

Adds the current signal to the actual process' save list.

```
void set(CNSimTime time, TimerType timer);
```

```
void reset(TimerType timer);
```

```
bool active(TimerType timer);
```

```
void remove(TimerType timer);
```

These are the functions concerning timers. Set is used to set a timer to a specific time (absolut time value!), reset stops an active timer, active returns TRUE when the timer is used, and remove should be called when a timer signal is consumed (otherwise active may return an incorrect value). In SDL set implies a call to reset.

```
void output_x(SignalType name, PId receiver, CNObject* data=0);
```

```
void output(SignalType name, CNObject* data=0);
```

```
void output(SignalType name, ProcessType rec, CNObject* data=0);
```

These are the SDL output functions for signals not leaving the block. The output_x variant is for explicit addressing, the other two are implicit addressing and addressing via signalroute.

```
void output_x(SignalType name, PId receiver, CNSDLChannel* ch, CNObject* data=0);
```

```
void output(SignalType name, ProcessType rec, CNSDLChannel* ch, CNObject* data=0);
```

These are the output functions for signals between blocks. There is explicit addressing and addressing via signalroute.

```
CNSimTime now()
```

Returns the actual simulation time.

```
void create(CNSDLProcess* proc_ad, PId proc=0);
```

```
void stop();
```

Dynamic SDL. Create is used to create a process either statically (when PId is given) or dynamically. The PId variant is only allowed at system start. Stop ends the current process and deletes it.

```
CNObject* get_data();
```

Returns the data object attached to the actual signal.

```
int get_source_system();
```

Returns the system the signal came from. This is either 1 (environment) or the own system number.

```
CNSDLChannel* channel(ChannelNo chn);
```

Returns a pointer to the specified channel (necessary for output via channel).

```
CNEventScheduler* scheduler();
```

Returns pointer to scheduler (used in CNSDLProcess).

13.2 CNSDLSignal — SDL Signal

SYNOPSIS

```
#include <CNCL/SDLSignal.h>
```

TYPE

`CN_SDLSIGNAL`

BASE CLASSES

CNEvent

DERIVED CLASSES

None

RELATED CLASSES

CNSDLManager, CNSDLProcess

DESCRIPTION

`CNSDLSignal` is an enhancement to the `CNEvent` class. Since in this form of SDL simulation the EventHandler pointers sender and receiver both contain the SDLManager's address another method had to be developed to identify the sending and receiving process.
Constructors:

```
CNSDLSignal();
CNSDLSignal(CNParam* param);
CNSDLSignal(SignalType name, CNSimTime t, PId s, PId r, int sy, CNObject* data=0);
CNSDLSignal(SignalType name, CNSimTime t, CNEventHandler* se, CNEventHandler* re,
PId s, PId r, int sy, CNObject* data=0);
```

Initializes the signal setting type, time, sender, receiver etc. The last form is only used by the environment.

Destructors:

```
~CNSDLSignal();
```

Deletes the signal. The destructor is private and may only be called by `CNSDLSignal` itself, its base class `CNEvent` or the friend classes `CNEventScheduler` and `CNSDLManager`. The destructor does NOT delete the referenced object.

In addition to the member functions required by CNCL, `CNSDLSignal` provides:

```
static void set_max_signals(unsigned long n);
static unsigned long get_max_signals();
```

Gets/Sets maximum number of signals. The default value for `max-signals` is 100. If `set_max_signals()` changes this value, it MUST be called before ANY signal is created.

```
typedef int PId;
typedef int SignalType;
PId sender();
void sender(PId s);
```

Gets/sets sending process ID.

```
PId receiver();
void receiver(PId r);
```

Gets/sets receiving process ID.

```
int system();
```

Returns sending system's number.

```
static void *operator new(size_t s);
```

The `new` operator. It is allocating an event out of a pool

```
static void operator delete(void *p);
```

The `delete` operator.

13.3 CNSDLProcess — SDL Process (dynamic)

SYNOPSIS

```
#include <CNCL/SDLProcess.h>
```

TYPE

`CN_SDLPROCESS`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNSDLProcessData, CNSDLManager

DESCRIPTION

`CNSDLProcess` is the base class for user SDL processes. Any SDL process must be derived from this class.
Constructors:

```
CNSDLProcess();
CNSDLProcess(CNParam *param);
CNSDLProcess(char* pname, CNSDLManager* SM, int type);
```

Initializes the process with name, pointer to SDLManager and process type.

In addition to the member functions required by CNCL, `CNSDLProcess` provides:

```
PId self()
PId parent()
PId offspring()
PId sender()
PId environment()
```

Get some special PId's

```
ProcessType type()
```

Get type of process.

```
int state();
```

`void state(int new_state);`

Gets/sets process state.

`char* get_name();`

Returns name of process.

`CNSimTime now()`

Returns actual simulation time.

`virtual int process(int signal);`

`Process` contains the dynamic behaviour of a SDL process. A user SDL process has to define at least this function. It is an equivalent to `event_handler` of class `CNEventHandler`. In addition the user can define an arbitrary number of procedures of any name. The actual "event handling" function (either `process` or a procedure) is called from `CNSDLManager` via a pointer to memberfunction. (Note that the compiler will issue a warning when procedure are used but the code should work!)

`typedef int(CNSDLProcess::*PMF)(int);`

`void procedure(PMF fp, SignalType name);`

Transfers control to the procedure pointed to by `fp`. The return address is stored on a stack. Even recursive calls should be possible.

`SignalType from_procedure();`

This ends a procedure. It returns control to the calling function and resumes behind the previous procedure call.

13.4 CNSDLProcessData — SDL Process (description)

SYNOPSIS

```
#include <CNCL/SDLProcessData.h>
```

TYPE

`CN_SDLPROCESSDATA`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNSDLProcess, CNSDLManager, CNSDLTimer

DESCRIPTION

`CNSDLProcessData` does the process managment. It contains a hash table with all active instances of a given process type and handles the low level parts of SDL process creation and deletion. It is also used by the class SDLManager to get the target process in sending signals.

Constructors:

`CNSDLProcessData();`

`CNSDLProcessData(CNParam* param);`

`CNSDLProcessData(ProcessType pt, int pn, PId first);`

Initializes `CNSDLProcessData`. Parameters are process type, maximum number of instances allowed and starting PId. The PId range from `first` to `first + pn - 1` is reserved (but not allocated!) for processes of the type `pt`.

In addition to the member functions required by CNCL, `CNSDLProcessData` provides:

`ProcessType type();`

Returns type of process.

`int max_number();`

Returns maximum number of instances of this process type.

`is_of_this_type(PId pid)`

Returns TRUE if pid is in range.

```
CNSDLProcess *get_process_by_PId(PId pid);
CNSDLProcess *get_process_by_type();
```

Returns a pointer to a process of the required type or the given PId.

```
PId add_process(CNSDLProcess* proc, PId pid = 0);
void delete_process(PId pid);
```

These ones are called by CNSDLManager in creation or deletion of processes.

13.5 CNSDLTimer — SDL Timer

SYNOPSIS

```
#include <CNCL/SDLTimer>
```

TYPE

`CN_SDLTIMER`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNSDLManager, CNSDLProcessData

DESCRIPTION

`CNSDLTimer` realizes the SDL timer management. This class is used by `CNSDLManager` and cannot be used alone.
Constructors:

```
CNSDLTimer();
CNSDLTimer(CNParam *param);
```

Initializes the class.

In addition to the member functions required by CNCL, `CNSDLTimer` provides:

```
CNSDLSignal* set(CNSimTime t, TimerType t_type, CNDLList* save_l, CNEventScheduler* sched,
PId proc, int sy);
CNSDLSignal* reset(TimerType t_type, CNDLList* save_l, CNEventScheduler* sched);
bool active(TimerType t_type);
void remove(TimerType t_type);
```

These functions are called by class `CNSDLManager` when its memberfunctions of the same name are executed. The user will never call the above functions directly.

```
void tell_ID(CNEventID id);
```

Is an information from class `CNSDLManager` to class `CNSDLTimer`.

```
void clear(CNDLList* save_l, CNEventScheduler* sched);
```

Resets all timers of a process. Is called when a process dies.

13.6 CNSDLEnvironment — SDL Environment

SYNOPSIS

```
#include <CNCL/SDLEnvironment.h>
```

TYPE

`CN_SDLENVIRONMENT`

BASE CLASSES

CNSDLManager

DERIVED CLASSES

None

RELATED CLASSES

CNSDLSignal, SDLProcess_Environment

DESCRIPTION

`CNSDLEnvironment` represents the environment of a SDL system. It is designed as an equivalent to a system, but in contrast to that connected to many regular SDL systems. System to system communication is only possible via the environment. There must be a user defined class `SDLProcess_Environment` which describes the dynamic behaviour of the environment. This must be derived from `CNSDLProcess`.
Note: the environment is out of the scope of SDL. It cannot be produced with the automatic translator!
Constructors:

```
CNSDLEnvironment();
CNSDLEnvironment(CNParam* param);
CNSDLEnvironment(int systems, CNSDLProcess* env_proc, CNEventScheduler* sched);
```

Initializes the SDL Environment. Parameters are the number of systems (to install), pointer to the process `SDLProcess_Environment`, and a pointer to the Event Scheduler.

`CNSDLEnvironment` redefines some functions derived from `CNSDLManager`. In addition it provides some new functions.

```
void output_x(SignalType name, PId receiver, CNObject* data=0);
void output(SignalType name, CNObject* data=0);
void output(SignalType name, ProcessType rec, CNObject* data=0);
```

These are the SDL output functions local to the environment. PId and process type of the environment are always 1 by definition.

`void output_x(SignalType name, PId receiver, CNSDLChannel* ch, CNObject* data=0, int target_system=0);`

`void output(SignalType name, ProcessType rec, CNSDLChannel* ch, CNObject* data=0, int target_system=0);`

These are the SDL output functions to communicate with other systems. The parameters are extended by the signals target system. This is the same number as the system number used in `install` and by the system definition.

`void install(int sys_no, CNSDLManager* sys);`

Connects system number `sys_no` with the environment.

`CNSDLManager *deinstall(int sys_no);`

Disconnects system `sys_no` and returns pointer to it. It doesn't however delete the system.

`CNSDLManager* S(int sys_no);`

Returns address of system `sys_no`. The environment is not a regular SDL system. It cheats by using the informations of the connected systems.

13.7 CNSDLChannel — SDL Channel

SYNOPSIS

```
#include <CNCL/SDLChannel>
```

TYPE

`CN_SDLCHANNEL`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

CNSDLManager

DESCRIPTION

`CNSDLChannel` represents a sdl channel. In this implementation only the random delay is modelled although a method to do something with the data is provided. This method can be defined a derived class.
Constructors:

```
CNSDLChannel();
CNSDLChannel(CNParam *param)
CNSDLChannel(CNRandom* distr);
```

Initializes the channel, optionally providing a random distribution. If none is given the channels are without delay.

In addition to the member functions required by CNCL `CNSDLChannel` provides:

```
void set_channel_delay(CNRandom* distr);
CNRandom* get_channel_delay();
```

Allow to alter the random distribution used by the channel.

```
double delay(double time);
```

Is called by `CNSDLManager` to get a random delay.

```
virtual void disturb(const SignalType name, CNObject* data);
```

Actually does nothing. It may however be defined to do anything on the data object in a derived class.

14 Persistent Classes

The classes described here provide the possibility of persistent objects. As in most cases it is not necessary to have all the overhead for persistency inside of a running program, no complete new class need to be designed if persistency is needed. It is sufficient to add an extension to a CNCL-kompatible class where all necessary methods are included.

To generate new persistent classes or extensions, derived from any `CNCL` compatible class, the utility `CNpersistent` is provided. It generates a class framework with all functions required by CNCL and all necesarry methods for persistency.

Usage:

`CNpersistent`*[-l] [-t template-dir] basename*

The required parameter `basename` is the name of the base class. This sript will produce the persistent extension to the mentioned base class. The result are two files in the current directory: `P`*basename*`.h` (header file) and `P`*basename*`.c` (implementation file).

The optional parameters are:

- [-l] : The base class is stored in the local directory, not in the standard CNCL-tree.
- [-t template-dir] : *template-dir* is the directory where the templates will be found (if they are not in the standard directory).

Example:

```
CNpersistent MyClass
```

creates the class/extension files `PMyClass.h` and `PMyClass.c`. MyClass and CNPObjectID will be the base classes to the new extension. The only methods that have to be defined after this are the read_from constructor and the storer-method.

NOTE:

- At some classes it will be necessary to achieve direct access to some private members of the base class, e.g. for the read_from constructor or the storer-method. In order to achieve this it is important to declare the persistent class or extension as *friend class* at the base class.
- All extensions have the complete *Default I/O member function for CNCL classes*. In most cases this I/O is not necessary because all these functions are already defined in the base class. Thus all of them can be deleted (they are marked with the note *(to be deleted if not needed)*).
- Please don't wonder that the rule of no multiple inheritance inside of CNCL seems to be broken at the persistent classes. The second base class, `CNPObjectID`, is not derived from any other CNCL-class. So the problems of multiple inheritance can not occur.

14.1 CNReaderTbl – Table for adress of reader-function

SYNOPSIS

```
#include <CNCL/ReaderTbl.h>
```

TYPE

`CN_READERTBL`

BASE CLASSES

CNCL

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

`CNReaderTbl` provides a table of all persistent classes (`Reader-Table`) and a second table of id's `ID-Table`. Both tables are realized as dynamic hash-tables.

The first one uses the classname as key for registering in the table, the object stored in it is the reader function pointer (the adress of the reader-function). This table is for checking if a class is registered to be persistent and for calling the reader constructor when objects are taken from the stream.

The other table uses the id as key and stores the according object to this id. The table is used during two occasions. First whenever an object is written to the stream the id is stored in this table. When the same object should be stored again it will be recognized. In addition to the id a special delimitation character is used for indicating this. When the objects are taken from the stream all objects and their old id are stored in this table again. When this special delimitation character apears on the stream the object can be taken out of the table by its id. Thus it is possible to restore containers even if one object appears several times at different locations, e.g at a double linked list.

Constructors:

`CNReaderTbl(char classname, Reader_ptr reader_ptr);`

Initializes a new Reader Table entry. Each given `classname` is added to the table.

Additional types provided by `CNReaderTbl` are:

`typedef CNObject*(*Reader_ptr)(CNPstream&);`

This is the type for the reader-function-pointer.

The functions provided by `CNReaderTbl` are:

`static Reader_ptr reader_pointer_by_classname(char* classname);`

This function checks if the requested `classname` is registered in the Reader Table and returns the according Reader_ptr. If it is not registered, a fatal error is yielded.

`static bool is_in(long id_key);`

Checks if the given `id_key` is in the id table.

`static void add_to_id_tbl(long id_key, CNObject* obj);`

`static void add_to_id_tbl(long id_key, CNObject& obj);`

Adds the persistent object `obj` to the id table with `id_key` as its key.

`static CNObject* get_from_id_tbl(long id_key);`

Returns the persistent object stored in the id table with `id_key` as the key.

`static void reset_id_tbl();`

Resets the id table to its initial state (all stored elemts are deleted). This method should be used if You change from the `write-to-stream` to the `read_from_stream` sequence.

14.2 PObjectID — ID-Managment for persistent Objects

SYNOPSIS

```
#include <CNCL/PObjectID.h>
```

TYPE

`CN_POBJECTID`

BASE CLASSES

None

DERIVED CLASSES

all persistent classes and extensions

RELATED CLASSES

None

DESCRIPTION

This class manages the (unique) identification number of each persistent object. All persistent classes must be derived from this class. As `CNPObjectID` is not derived from any CNCL-class `multiple inheritance` can be used.

One of the member variables, declared as static, is the counter for the actual id. It is increased whenever a new id is given to an object. The other one is the id of the object itself. This number cannot occur two times inside of one program.

Constructors:

`CNPObjectID();`

Initializes the id and increases the `static` counter. This constructor must be called in all constructors of the derived classes.

Additional types provided by `CNReaderTbl` are:

`typedef long PID`

For a possible future change from long to any other larger number this typedef has been included.

`CNPObjectID` provides the following functions:

`CNPID object_id();`

Returns the object id.

14.3 CNPIO — persistent stream Object IO-formatting

SYNOPSIS

```
#include <CNCL/PIO.h>
```

TYPE

`CN_PIO`

BASE CLASSES

CNCL

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

`CNPIO` provides the format of storing and reading the persistent objects on/from a stream.The format convention can be described as follows:

```
classname
delimitation character
data
delimitation character
```

The functions provided by `CNPIO` are:

```
static int store_object(CNPstream& stream, CNObject& obj, bool no_ptr_check = FALSE);
```

```
static CNObject* read_object(CNPstream& stream);
```

Reads/Stores a *persistent* `CNObject` on the `stream`. A `CNObject` is recognized to be persistent if it is registered at the actual `Reader Table`. `no_ptr_check` determines if the objects are checked for multiple appearence (`no_ptr_check = FALSE`) or not (`TRUE`).

Stores the id of the persistent object `obj` to the given stream.

```
static CNPID read_id(CNPstream& stream);
```

Reads the id from a stream.

14.4 CNPstream — abstract base class for persistent stream classes

SYNOPSIS

```
#include <CNCL/Pstream.h>
```

TYPE

`CN_PSTREAM`

BASE CLASSES

CNObject

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

`CNPstream` is the abstract base class for the persistent stream classes. Thus, only IO-operators and IO-functions are defined here. The concrete definition of the methods and operators must be done in the derived classes. Thus by this approach the possibilities of polymorphism are kept.

Constructors:

```
CNPstream();
CNPstream(*param);
```

Initializes the `CNPstream`.

In addition to the member functions required by CNCL, `CNstream` provides:

```
virtual CNPstream& operator<<(const char*)=0;
virtual CNPstream& operator<<(char)=0;
virtual CNPstream& operator<<(long)=0;
virtual CNPstream& operator<<(double)=0;
virtual CNPstream& operator>>(char*)=0;
virtual CNPstream& operator>>(char&)=0;
virtual CNPstream& operator>>(long&)=0;
virtual CNPstream& operator>>(double&)=0;
```

Virtual persistent IO operators.

```
virtual CNPstream& getline(char*, int )=0;
```

Virtual `getline` function for string input.

14.5 CNPiostream — persistent iostream format

SYNOPSIS

```
#include <CNCL/Piostream.h>
```

TYPE

`CN_PIOSTREAM`

BASE CLASSES

CNPstream

DERIVED CLASSES

None

RELATED CLASSES

None

DESCRIPTION

The class `CNPiostream` manages the interface to the gnu c++ iostream library. Thus, all virtual functions of the base class `CNPstream` are defined here. For a description see the base class.

Constructors:

```
CNPiostream(iostream& s);
CNPiostream(CNParam *param);
```

Initializes the `CNPiostream`.

14.6 CNPInt — class persistent CNInt

SYNOPSIS

```
#include <CNCL/PInt.h>
```

TYPE

`CN_PINT`

BASE CLASSES

CNInt, CNPObjectID

DERIVED CLASSES

None

RELATED CLASSES

CNPString, CNPDouble

DESCRIPTION

`CNPInt` manages the persistency of `CNInt`'s. As `CNInt` is its base class, all methods of `CNInt` are available.

Constructors:

```
CNPInt(long val=0);
CNPInt(CNPstream& stream);
CNPInt(CNParam *param);
```

Initializes the `CNPInt`. Either the value is given as `val` (0 by default) or it will be read from the persistent `stream` (reader-constructor).

In addition to the member functions required by CNCL, `CNInt` provides:

```
virtual int store_on(CNPstream& s);
int store_on(CNPstream& s, bool no_ptr_check);
```

Stores the `CNPInt` on the persistent stream `s`. The boolean parameter `no_ptr_check` switches the check for multiple storing off (if set *TRUE*).

```
static CNPInt* read_from(CNPstream& s);
```

Reads the `CNPInt` from stream `s`.

```
static CNObject* object_read_from(CNPstream& s);
```

Reads from stream `s` as a `CNObject`.

```
CNPID object_id();
```

Returns the ID of the current Object.

`virtual int storer(CNPstream&);`

`static CNPInt* reader(CNPstream& s);`

These two functions are called by the class `CNPIO` as a connection to the persistent output/reader-construktor.

14.7 CNP<type> — persistent types

DESCRIPTION

CNCL currently provides persistent extensions for the (CNCL-) data types `CNDouble`, `CNInt`, `CNString` and the container classes `CNDLList`, `CNArrayObject`.

The only difference of functions in these extensions to the previosly described extension `CNPInt` may be some additional constructors and =operators for a better connection to the base class.

Concept Index

A

Abstract Base Class for Hash Tables 121
Additive RNG 18
Array 108, 113
Array Classes 107
Array of <Type> 113
Array of CNCL Objects 109
Array of Integer 111
Array2 ..... 114, 116
Array2 of <Type> 116
array2char 115
AVL node 65
AVL tree 62

B

Base Class 8
Basic Classes 7
Batch-Means 58
Beta Distribution 27
Binomial Distribution 28

C

Character String 138
Class Description 4, 11, 13
Class Generation 6
Class Hierarchy 2, 11
Class Templates 6
Classes 2, 7
CNarray 113, 116
CNCL Base Class 8
CNCL Classes 2
CNCL Concept 4
CNCL Hierarchy 2
CNgenclass 6
CNQueue 75
Common Member Functions 4
Container Classes 61
Coordinates 134, 136

D

Data File RNG 20
Debug Output 11
Deterministic Distribution 29
Discrete Uniform Distribution 31
Distribution from Table of CDF 30, 36, 45
Distribution Function 54, 56, 58
DLRE 56
Double 149
Doubly Linked List 67, 70, 72

E

Easy Drawing 155
Erlang-k Distribution 32
Error Handling 8
Error Messages 9
Evaluation 49
Event Base Scheduler 96
Event Driven Simulation 88, 91, 92
Event Handler 91, 92
Event Scheduler 94, 98, 100
Events 87, 88, 91, 92, 94, 96, 98, 100
Example 104
EZD 156
EZD Bit Display 172
EZD Diagram 170
EZD Diagram Window 166
EZD Drawing 161
EZD Objects 159
EZD Popup Menu 173
EZD Push-Button 162
EZD Queue 175
EZD Server 177
EZD Text 178
EZD Text Window 168
EZD Timer 180
EZD Window 164

F

Fibonacci RNG 19
FIFO Queue 75
FormFloat 145
FormInt 143
Fuzzy Array Set 192
Fuzzy Clause 184
Fuzzy Logic 183
Fuzzy Number Triangle 197
Fuzzy Rule Base 189
Fuzzy Rules 187
Fuzzy Set 191
Fuzzy Set L/R 193
Fuzzy Trapez Set 195
Fuzzy Triangle Set 196
Fuzzy Variable 185

G

Generic Container Classes ... 61
Geometric Distribution ... 33

H

Hash table ... 127
Hash Table ... 121
Hash Table with Dynamic Capacity ... 125
Hash Table with Static Capacity ... 123
Hyperexponential Distribution ... 34
Hypergeometrical Distribution ... 35

I

Int ... 147
Integer Coordinates ... 136
Integer Key ... 120
Introduction ... 1
iterate through event list ... 102
Iterator ... 72, 127

J

Job ... 83

K

Key ... 118

L

LIFO Queue ... 76
Linear Congruence RNG ... 21
Linked list ... 70
Linked List ... 67, 72
List ... 67, 70, 72
Log-normal Distribution ... 37
LRE ... 54

M

M/M/1 Simulation ... 104
Manager ... 130
Math Error ... 8
Member Functions ... 4
Miscellaneous ... 133
Mix of Distributions ... 42
Moments ... 52
Multiple Linear Congruence RNG ... 22

N

Negative Exponential Distribution ... 39
Node ... 70
Normal Distribution ... 40

O

Object Class ... 11
Object Creation ... 4
Object Management ... 117, 118, 119, 120
Object Management Frontend ... 130
Object with Name ... 142
Output ... 11

P

Parameter Class ... 14
Parameter Objects ... 14
Persistent ... 213
Persistent CNInt ... 220
Persistent IO ... 217
Persistent IO Stream ... 219
Persistent Object ID ... 216
Persistent Reader Table ... 214
Persistent Stream ... 218
Persistent Types ... 222
Poisson Distribution ... 41
Priority FIFO Queue ... 81

Q

Queue ... 74

R

Random Distribution ... 26
Random Distribution Base Class ... 26
Random Integers ... 24
Random Mix of Deterministic Values ... 38
Random Number Generator ... 16
Random Number Generators ... 15
Random Numbers ... 15
Random Queue ... 77
Range Checking ... 108, 109, 111, 113, 116
Rayleigh Distribution ... 43
Rice Distribution ... 44
RNG ... 15
RNG ACG ... 18
RNG Base Class ... 16
RNG by Data Files ... 20
RNG Fibonacci ... 19
RNG Linear Congruence ... 21
RNG Multiple Linear Congruence ... 22
RNG Tausworth ... 23
Root of Hierarchy ... 11
Runtime Type Information ... 11, 13

S

Safe Type Casts ... 4
SDL ... 199
SDL Channel ... 212

SDL Environment 210
SDL Manager 201
SDL Process 205
SDL Process Data 207
SDL Signal 203
SDL Timer 209
Simulation 87, 88, 91, 92, 96, 98, 100, 104
Simulation Time 103
Sink 84
SPT Queue 79
Stack 85
Statistical Evaluation 49
Statistics 50
String 138
String Key 119

T

Tausworth RNG 23
Templates 6
Type Casts 4
Type Checking 4, 11
Type Information 4, 11, 13

U

Uniform Distribution 46
Unix 151
Unix pipes 152
Unix Select 153

W

Weibull Distribution 47

Function Index

~

~CNArray2Char() ... 115
~CNArrayInt() ... 111
~CNArrayObject() ... 109
~CNAVLTree() ... 62
~CNDLList() ... 67
~CNEvent() ... 89
~CNHashDynamic() ... 125
~CNHashStatic() ... 123
~CNManager() ... 130
~CNSDLSignal() ... 203
~CNString() ... 138
~EZDWindow() ... 164

A

abs() ... 197
activate() ... 181
active() ... 202, 209
active_color() ... 181
add() ... 62, 139, 167, 169, 171
add_event() ... 94, 96, 98, 100
add_except() ... 153
add_handler() ... 91
add_in_var() ... 189
add_lhs() ... 187
add_out_var() ... 189
add_process() ... 208
add_read() ... 153
add_rhs() ... 187
add_rule() ... 189
add_to_id_tbl() ... 215
add_value_set() ... 186
add_write() ... 153
after() ... 90, 139
aggregate() ... 188
aggregate_all() ... 190
aggregate_value() ... 188
alpha() ... 47
angle() ... 181
append() ... 68
arc() ... 160
Array2 ... 114
as_double() ... 16
as_float() ... 16
as_int() ... 25
as_long() ... 16, 24
as_long32() ... 16, 18, 19, 20, 21, 22, 23
AVLTree ... 62

B

bayes_err() ... 59
before() ... 139
beta() ... 47
bitmap() ... 160
bool ... 10

C

capacity() ... 139
capitalize() ... 139
cast_from_object() ... 5, 13
center_of_gravity() ... 191, 192, 197
certainty() ... 187
channel() ... 202
ChannelNo ... 201
class_desc() ... 4, 11
clear() ... 85, 167, 169, 171, 209
close() ... 152
CN_ACG ... 18
CN_ARRAY ... 108
CN_ARRAY2 ... 114
CN_ARRAY2CHAR ... 115
CN_ARRAYINT ... 111
CN_ARRAYOBJECT ... 109
CN_AVLNode ... 65
CN_AVLTREE ... 62
CN_Batches ... 58
CN_BETA ... 27
CN_BINOMIAL ... 28
CN_CLASS ... 13
CN_CNFSETARRAY ... 192
CN_COORD ... 134
CN_DETERM ... 29
CN_DIRACTAB ... 30
CN_DISCUNIFORM ... 31
CN_DLITERATOR ... 72
CN_DLLIST ... 67
CN_DLREF ... 56
CN_DLREG ... 56
CN_DOUBLE ... 149
CN_ERLANG ... 32
CN_EVENT ... 88
CN_EVENTBASESCHED ... 96
CN_EVENTEXPLODER ... 91
CN_EVENTHANDLER ... 92
CN_EVENTHEAPSCHED ... 100
CN_EVENTHITERATOR ... 102
CN_EVENTLIST ... 94

CN_EVENTLITERATOR ... 102
CN_EVENTSCHEDULER ... 98
CN_EZD ... 156
CN_EZDBLOCK ... 172
CN_EZDDIAG ... 170
CN_EZDDIAGWIN ... 166
CN_EZDDOBJECT ... 159
CN_EZDDRAWING ... 161
CN_EZDPOPUP ... 173
CN_EZDPushButton ... 162
CN_EZDQUEUE ... 175
CN_EZDSERVER ... 177
CN_EZDTEXT ... 178
CN_EZDTIMER ... 180
CN_EZDTXTWIN ... 168
CN_EZDWINDOW ... 164
CN_FCLAUSE ... 184
CN_FIBOG ... 19
CN_FILEG ... 20
CN_FNUMTRIANGLE ... 197
CN_FORMFLOAT ... 145
CN_FORMINT ... 143
CN_FRULE ... 187
CN_FRULEBASE ... 189
CN_FSET ... 191
CN_FSETLR ... 193
CN_FSETTRAPEZ ... 195
CN_FSETTRIANGLE ... 196
CN_FVAR ... 185
CN_GEOMETRIC ... 33
CN_HASHDYNAMIC ... 125
CN_HASHITERATOR ... 127
CN_HASHSTATIC ... 123
CN_HASHTABLE ... 121
CN_HYPEREXP ... 34
CN_HYPERGEOM ... 35
CN_ICOORD ... 136
CN_INT ... 147
CN_INTERTAB ... 36
CN_JOB ... 83
CN_KEY ... 118
CN_KEYINT ... 120
CN_KEYSTRING ... 119
CN_LCG ... 21
CN_LOGNORMAL ... 37
CN_LREF ... 54
CN_LREG ... 54
CN_MANAGER ... 130
CN_MDETERM ... 38
CN_MLCG ... 22
CN_Moments ... 52
CN_NAMED ... 142
CN_NEGEXP ... 39
CN_NORMAL ... 40
CN_OBJECT ... 11, 70
CN_PARAM ... 14
CN_PINT ... 220
CN_PIO ... 217
CN_PIOSTREAM ... 219
CN_PIPE ... 152
CN_POBJECTID ... 216
CN_POISSON ... 41
CN_PRIOQUEUEFIFO ... 81
CN_PSTREAM ... 218
CN_QUEUE ... 74
CN_QUEUEFIFO ... 75
CN_QUEUELIFO ... 76
CN_QUEUERANDOM ... 77
CN_QUEUESPT ... 79
CN_RANDOM ... 26
CN_RANDOMMIX ... 42
CN_Rayleigh ... 43
CN_READERTBL ... 214
CN_RICE ... 44
CN_RNDINT ... 24
CN_RNG ... 16
CN_SDLCHANNEL ... 212
CN_SDLENVIRONMENT ... 210
CN_SDLMANAGER ... 201
CN_SDLPROCESS ... 205
CN_SDLPROCESSDATA ... 207
CN_SDLSIGNAL ... 203
CN_SDLTIMER ... 209
CN_SELECT ... 153
CN_SINK ... 84
CN_STACK ... 85
CN_STATISTICS ... 50
CN_STRING ... 138
CN_TAB ... 45
CN_TAUSG ... 23
CN_UNIFORM ... 46
CN_WEIBULL ... 47
CNACG ... 18
CNACG() ... 18
CNACG::as_long32() ... 18
CNACG::CNACG() ... 18
CNACG::has_long32() ... 18
CNACG::reset() ... 18
CNArray ... 108
CNArray() ... 108
CNArray::CNArray() ... 108
CNArray::get_size() ... 108
CNArray::set_size() ... 108, 114
CNArray2() ... 114

CNArray2::CNArray2() 114
CNArray2::cols() 114
CNArray2::get_cols() 114
CNArray2::get_rows() 114
CNArray2::rows() 114
CNArray2::size() 114
CNArray2Char 115
CNArray2Char() 115
CNArray2Char::~CNArray2Char() 115
CNArray2Char::CNArray2Char() 115
CNArray2Char::get() 115
CNArray2Char::put() 115
CNArray2Char::size() 115
CNArrayInt 111
CNArrayInt() 111
CNArrayInt::~CNArrayInt() 111
CNArrayInt::CNArrayInt() 111
CNArrayInt::get() 111
CNArrayInt::operator=() 112
CNArrayInt::operator[]() 112
CNArrayInt::put() 111
CNArrayInt::set_size() 111
CNArrayObject 109
CNArrayObject() 109
CNArrayObject::~CNArrayObject() 109
CNArrayObject::CNArrayObject() 109
CNArrayObject::get() 109
CNArrayObject::operator=() 110
CNArrayObject::operator[]() 109
CNArrayObject::put() 109
CNArrayObject::set_size() 109
CNAVLNode 65
CNAVLNode() 65
CNAVLNode::CNAVLNode() 65
CNAVLNode::compare() 65
CNAVLNode::find() 65
CNAVLNode::left() 65
CNAVLNode::right() 65
CNAVLTree() 62
CNAVLTree::~CNAVLTree() 62
CNAVLTree::add() 62
CNAVLTree::CNAVLTree() 62
CNAVLTree::delete_all() 63
CNAVLTree::empty() 63
CNAVLTree::find() 62
CNAVLTree::find_first() 63
CNAVLTree::length() 63
CNAVLTree::remove() 63
CNAVLTree::remove_first() 63
CNAVLTree::root() 63
CNBatches 58
CNBatches() 58
CNBatches::bayes_err() 59
CNBatches::CNBatches() 58
CNBatches::print() 59
CNBatches::rd_conf() 59
CNBatches::sigma() 59
CNBatches::status() 59
CNBeta 27
CNBeta() 27
CNBeta::CNBeta() 27
CNBeta::operator() 27
CNBinomial 28
CNBinomial() 28
CNBinomial::CNBinomial() 28
CNBinomial::n() 28
CNBinomial::operator() 28
CNBinomial::u() 28
CNCL 8
CNCL::default_exit_handler() 9
CNCL::err_abort 8
CNCL::err_default 9
CNCL::err_fatal 8
CNCL::err_ignore 8
CNCL::err_info 8
CNCL::err_warning 8
CNCL::error() 9
CNCL::ErrorType 8
CNCL::fatal() 9
CNCL::get_error() 9
CNCL::info() 9
CNCL::msg() 9
CNCL::set_error() 9
CNCL::set_exit_handler() 9
CNCL::warning() 9
CNClass 13
CNClass::cast_from_object() 13
CNClass::get_name() 13
CNClass::get_version() 13
CNClass::name() 13
CNClass::new_object() 13
CNClass::version() 13
CNCoord 134
CNCoord() 134
CNCoord::CNCoord() 134
CNCoord::get_scale() 135
CNCoord::operator-= 134
CNCoord::operator+= 134
CNCoord::set_scale() 135
CNCoord::x 134
CNCoord::y 134
CNDeterm 29
CNDeterm() 29
CNDeterm::CNDeterm() 29

CNDeterm::mean() ... 29
CNDeterm::operator() ... 29
CNDeterm::value() ... 29
CNDiracTab ... 30
CNDiracTab() ... 30
CNDiracTab::CNDiracTab() ... 30
CNDiracTab::operator() ... 30
CNDiscUniform ... 31
CNDiscUniform() ... 31
CNDiscUniform::CNDiscUniform() ... 31
CNDiscUniform::high() ... 31
CNDiscUniform::low() ... 31
CNDiscUniform::operator() ... 31
CNDLIterator ... 72
CNDLIterator() ... 72
CNDLIterator::CNDLIterator() ... 72
CNDLIterator::first() ... 73
CNDLIterator::first_object() ... 73
CNDLIterator::get_object() ... 72
CNDLIterator::get_position() ... 72
CNDLIterator::last() ... 73
CNDLIterator::last_object() ... 73
CNDLIterator::next() ... 73
CNDLIterator::next_object() ... 73
CNDLIterator::object() ... 72
CNDLIterator::operator -- ... 73
CNDLIterator::operator ++ ... 73
CNDLIterator::position() ... 72
CNDLIterator::prev() ... 73
CNDLIterator::prev_object() ... 73
CNDLIterator::reset() ... 72
CNDLList ... 67
CNDLList() ... 67
CNDLList::~CNDLList() ... 67
CNDLList::append() ... 68
CNDLList::CNDLList() ... 67
CNDLList::delete_all() ... 68
CNDLList::delete_all_w_obj() ... 68
CNDLList::delete_object() ... 68
CNDLList::empty() ... 67
CNDLList::first() ... 67
CNDLList::insert_after() ... 68
CNDLList::insert_before() ... 68
CNDLList::last() ... 67
CNDLList::length() ... 68
CNDLList::next() ... 67
CNDLList::ok() ... 68
CNDLList::prepend() ... 68
CNDLList::prev() ... 67
CNDLList::remove_object() ... 68
CNDLObject ... 70
CNDLObject() ... 70
CNDLObject::CNDLObject() ... 70
CNDLObject::delete_object() ... 71
CNDLObject::get_next() ... 70
CNDLObject::get_object() ... 71
CNDLObject::get_prev() ... 70
CNDLObject::next() ... 70
CNDLObject::object() ... 71
CNDLObject::prev() ... 70
CNDLObject::set_next() ... 70
CNDLObject::set_object() ... 71
CNDLObject::set_prev() ... 70
CNDLREF ... 56
CNDLREF() ... 56
CNDLREF::CNDLREF() ... 56
CNDLREF::status() ... 57
CNDLREG ... 56
CNDLREG() ... 56
CNDLREG::CNDLREG() ... 56
CNDLREG::status() ... 57
CNDouble ... 149
CNDouble() ... 149
CNDouble::CNDouble() ... 149
CNDouble::operator *= ... 150
CNDouble::operator - ... 149
CNDouble::operator -= ... 149
CNDouble::operator /= ... 150
CNDouble::operator + ... 149
CNDouble::operator += ... 149
CNDouble::operator double() ... 149
CNErlang ... 32
CNErlang() ... 32
CNErlang::CNErlang() ... 32
CNErlang::mean() ... 32
CNErlang::operator() ... 32
CNErlang::Variance() ... 32
CNEvent ... 88
CNEvent() ... 88
CNEvent::~CNEvent() ... 89
CNEvent::after() ... 90
CNEvent::CNEvent() ... 88
CNEvent::from() ... 89
CNEvent::get_from() ... 89
CNEvent::get_id() ... 90
CNEvent::get_issued() ... 89
CNEvent::get_max_events() ... 89
CNEvent::get_object() ... 89
CNEvent::get_priority() ... 89
CNEvent::get_scheduled() ... 89
CNEvent::get_state() ... 92
CNEvent::get_to() ... 89
CNEvent::get_type() ... 89
CNEvent::id() ... 90

CNEvent::issued() ... 89
CNEvent::object() ... 89
CNEvent::operator delete() ... 90
CNEvent::operator new() ... 90
CNEvent::priority() ... 89
CNEvent::scheduled() ... 89
CNEvent::set_from() ... 89
CNEvent::set_max_events() ... 89
CNEvent::set_object() ... 89
CNEvent::set_priority() ... 89
CNEvent::set_scheduled() ... 89
CNEvent::set_state() ... 92
CNEvent::set_to() ... 89
CNEvent::set_type() ... 89
CNEvent::state() ... 92
CNEvent::to() ... 89
CNEvent::type() ... 89
CNEventBaseSched ... 96
CNEventBaseSched() ... 96
CNEventBaseSched::add_event() ... 96
CNEventBaseSched::CNEventBaseSched() ... 96
CNEventBaseSched::delete_event() ... 96
CNEventBaseSched::delete_events() ... 96
CNEventBaseSched::peek_event() ... 96
CNEventBaseSched::send_event() ... 96
CNEventBaseSched::start() ... 97
CNEventBaseSched::statistics() ... 96
CNEventBaseSched::stop() ... 97
CNEventBaseSched::time() ... 96
CNEventExploder ... 91
CNEventExploder() ... 91
CNEventExploder::add_handler() ... 91
CNEventExploder::CNEventHandler() ... 91
CNEventExploder::event_handler() ... 91
CNEventExploder::rem_handler() ... 91
CNEventHandler ... 92
CNEventHandler() ... 92
CNEventHandler::CNEventHandler() ... 92
CNEventHandler::delete_event() ... 93
CNEventHandler::delete_events() ... 93
CNEventHandler::event_handler() ... 92
CNEventHandler::now() ... 93
CNEventHandler::scheduler() ... 92
CNEventHandler::send() ... 93
CNEventHandler::send_delay() ... 93
CNEventHandler::send_event() ... 93
CNEventHandler::send_now() ... 93
CNEventHeapSched ... 100
CNEventHeapSched() ... 100
CNEventHeapSched::add_event() ... 100
CNEventHeapSched::CNEventHeapSched() ... 100
CNEventHeapSched::create_iterator() ... 101
CNEventHeapSched::delete_event() ... 100
CNEventHeapSched::delete_events() ... 100
CNEventHeapSched::peek_event() ... 101
CNEventHeapSched::send_event() ... 100
CNEventHeapSched::start() ... 101
CNEventHeapSched::stop() ... 101
CNEventHeapSched::time() ... 101
CNEventHIterator ... 102
CNEventHIterator::delete_current_event() ... 102
CNEventHIterator::next_event() ... 102
CNEventID ... 90
CNEventList ... 94
CNEventList() ... 94
CNEventList::add_event() ... 94
CNEventList::CNEventList() ... 94
CNEventList::delete_all() ... 94
CNEventList::delete_event() ... 94
CNEventList::delete_events() ... 94
CNEventList::next_event() ... 94
CNEventList::peek_event() ... 94
CNEventLIterator ... 102
CNEventLIterator::delete_current_event() ... 102
CNEventLIterator::next_event() ... 102
CNEventScheduler ... 98
CNEventScheduler() ... 98
CNEventScheduler::add_event() ... 98
CNEventScheduler::CNEventScheduler() ... 98
CNEventScheduler::create_iterator() ... 99
CNEventScheduler::delete_event() ... 98
CNEventScheduler::delete_events() ... 98
CNEventScheduler::peek_event() ... 99
CNEventScheduler::send_event() ... 98
CNEventScheduler::start() ... 99
CNEventScheduler::stop() ... 99
CNEventScheduler::time() ... 99
CNEZDTimer ... 180
CNFClause ... 184
CNFClause() ... 184
CNFClause::CNFClause() ... 184
CNFClause::print_clause() ... 184
CNFiboG ... 19
CNFiboG() ... 19
CNFiboG::as_long32() ... 19
CNFiboG::CNFiboG() ... 19
CNFiboG::has_long32() ... 19
CNFiboG::reset() ... 19, 20
CNFiboG::seed() ... 19
CNFileG ... 20
CNFileG() ... 20
CNFileG::as_long32() ... 20
CNFileG::CNFileG() ... 20
CNFileG::has_long32() ... 20

CNFileG::newfile() ... 20
CNFileG::wrong_parity() ... 20
CNFNumTriangle ... 197
CNFNumTriangle() ... 197
CNFNumTriangle::abs() ... 197
CNFNumTriangle::center_of_gravity() ... 197
CNFNumTriangle::CNFNumTriangle() ... 197
CNFNumTriangle::defuzzy() ... 197
CNFNumTriangle::operator * ... 197
CNFNumTriangle::operator - ... 197
CNFNumTriangle::operator = ... 197
CNFNumTriangle::operator + ... 197
CNFNumTriangle::sqr() ... 197
CNFormFloat ... 145
CNFormFloat() ... 145
CNFormFloat::CNFormFloat() ... 145
CNFormFloat::fill() ... 146
CNFormFloat::fixed ... 145
CNFormFloat::format() ... 146
CNFormFloat::get_fill() ... 146
CNFormFloat::get_format() ... 146
CNFormFloat::get_precision() ... 146
CNFormFloat::get_value() ... 146
CNFormFloat::get_width() ... 146
CNFormFLoat::left ... 145
CNFormFloat::precision() ... 146
CNFormFloat::right ... 145
CNFormFloat::scientific ... 145
CNFormFloat::set_fill() ... 146
CNFormFloat::set_format() ... 146
CNFormFloat::set_precision() ... 146
CNFormFloat::set_value() ... 146
CNFormFloat::set_width() ... 146
CNFormFloat::showpoint ... 145
CNFormFloat::value() ... 146
CNFormFloat::width() ... 146
CNFormInt ... 143
CNFormInt() ... 143
CNFormInt::CNFormInt() ... 143
CNFormInt::fill() ... 143, 144
CNFormInt::format() ... 144
CNFormInt::get_fill() ... 143
CNFormInt::get_format() ... 144
CNFormInt::get_value() ... 143
CNFormInt::get_width() ... 144
CNFormInt::left ... 143
CNFormInt::right ... 143
CNFormInt::set_fill() ... 144
CNFormInt::set_format() ... 144
CNFormInt::set_value() ... 143
CNFormInt::set_width() ... 144
CNFormInt::value() ... 143
CNFormInt::width() ... 144
CNFRileBase::aggregate_all() ... 190
CNFRule ... 187
CNFRule() ... 187
CNFRule::add_lhs() ... 187
CNFRule::add_rhs() ... 187
CNFRule::aggregate() ... 188
CNFRule::aggregate_value() ... 188
CNFRule::certainty() ... 187
CNFRule::CNFRule() ... 187
CNFRule::get_certainty() ... 187
CNFRule::get_lhs() ... 187
CNFRule::get_n_lhs() ... 187
CNFRule::get_n_rhs() ... 187
CNFRule::get_rhs() ... 187
CNFRule::set_certainty() ... 187
CNFRuleBase ... 189
CNFRuleBase() ... 189
CNFRuleBase::add_in_var() ... 189
CNFRuleBase::add_out_var() ... 189
CNFRuleBase::add_rule() ... 189
CNFRuleBase::CNFRuleBase() ... 189
CNFRuleBase::debug_rules() ... 190
CNFRuleBase::defuzzy_all() ... 190
CNFRuleBase::evaluate() ... 190
CNFRuleBase::evaluate_all() ... 190
CNFRuleBase::get_n_in_vars() ... 189
CNFRuleBase::get_n_out_vars() ... 189
CNFRuleBase::get_n_rules() ... 189
CNFRuleBase::inference() ... 190
CNFRuleBase::resolution() ... 189
CNFSet ... 191
CNFSet() ... 191
CNFSet::center_of _gravity() ... 191
CNFSet::CNFSet ... 191
CNFSet::get_membership() ... 191
CNFSetArray ... 192
CNFSetArray() ... 192
CNFSetArray::center_of_gravity() ... 192
CNFSetArray::CNFSetArray() ... 192
CNFSetArray::get() ... 192
CNFSetArray::get_membership() ... 192
CNFSetArray::get_n() ... 192
CNFSetArray::operator[] ... 192
CNFSetArray::put() ... 192
CNFSetLR ... 193
CNFSetLR() ... 193
CNFSetLR::CNFSetLR() ... 193
CNFSetLR::get_alpha() ... 194
CNFSetLR::get_beta() ... 194
CNFSetLR::get_m1() ... 194
CNFSetLR::get_m2() ... 194

CNFSetLR::get_membership() 194
CNFSetTrapez 195
CNFSetTrapez() 195
CNFSetTrapez::CNFSetTrapez() 195
CNFSetTrapez::get_membership() 195
CNFSetTriangle 196
CNFSetTriangle() 196
CNFSetTriangle::CNFSetTriangle() 196
CNFSetTriangle::get_mean() 196
CNFSetTriangle::get_membership() 196
CNFuncType 194
CNFVar 185
CNFVar() 185
CNFVar::add_value_set() 186
CNFVar::CNFVar() 185
CNFVar::fuzzy_value() 186
CNFVar::get_fuzzy_value() 186
CNFVar::get_membership() 186
CNFVar::get_value() 185
CNFVar::get_value_set() 186
CNFVar::get_xmax() 186
CNFVar::get_xmin() 186
CNFVar::print_membership() 186
CNFVar::set_fuzzy_value() 186
CNFVar::set_value() 185
CNFVar::value() 185
CNFVar::xmax() 186
CNFVar::xmin() 186
CNgenclass 6, 113, 116
CNGeometric 33
CNGeometric() 33
CNGeometric::CNGeometric() 33
CNGeometric::mean() 33
CNGeometric::operator() 33
CNHashDynamic 125
CNHashDynamic() 125
CNHashDynamic::~CNHashDynamic() 125
CNHashDynamic::CNHashDynamic() 125
CNHashDynamic::delete_key() 125
CNHashDynamic::delete_key_absolutely() 126
CNHashDynamic::store_key() 125
CNHashIterator 127
CNHashIterator() 127
CNHashIterator::CNHashIterator() 127
CNHashIterator::first() 127
CNHashIterator::first_key() 127
CNHashIterator::first_object() 128
CNHashIterator::get_key() 127
CNHashIterator::get_object() 128
CNHashIterator::key() 127
CNHashIterator::last() 128
CNHashIterator::last_key() 128
CNHashIterator::last_object() 128
CNHashIterator::next() 128
CNHashIterator::next_key() 128
CNHashIterator::next_object() 128
CNHashIterator::object() 128
CNHashIterator::operator -- 128
CNHashIterator::operator ++ 128
CNHashIterator::prev() 128
CNHashIterator::prev_key() 128
CNHashIterator::prev_object() 128
CNHashIterator::reset() 127
CNHashStatic 123
CNHashStatic() 123
CNHashStatic::~CNHashStatic() 123
CNHashStatic::CNHashStatic() 123
CNHashStatic::delete_key() 123
CNHashStatic::delete_key_absolutely() 124
CNHashStatic::store_key() 123
CNHashTable 121
CNHashTable() 121
CNHashTable::CNHashTable() 121
CNHashTable::DEFAULT_HASH_TABLE_CAPACITY 121
CNHashTable::delete_key() 122
CNHashTable::delete_key_absolutely() 122
CNHashTable::get_capacity() 122
CNHashTable::get_key() 121
CNHashTable::get_num_entries() 122
CNHashTable::get_object() 121
CNHashTable::is_empty() 122
CNHashTable::is_full() 122
CNHashTable::reset() 122
CNHashTable::reset_absolutely() 122
CNHashTable::store_key() 121
CNHyperExp 34
CNHyperExp() 34
CNHyperExp::CNHyperExp() 34
CNHyperExp::operator() 34
CNHyperGeom 35
CNHyperGeom() 35
CNHyperGeom::CNHyperGeom() 35
CNHyperGeom::mean() 35
CNHyperGeom::operator() 35
CNHyperGeom::variance() 35
CNICoord 136
CNICoord() 136
CNICoord::CNICoord() 136
CNICoord::get_scale() 137
CNICoord::operator -= 136
CNICoord::operator += 136
CNICoord::set_scale() 137
CNICoord::x 136
CNICoord::y 136

CNInt 147
CNInt() 147
CNInt::CNInt() 147
CNInt::operator != 148
CNInt::operator %= 148
CNInt::operator &= 148
CNInt::operator *= 148
CNInt::operator - 148
CNInt::operator -- 147
CNInt::operator -= 148
CNInt::operator /= 148
CNInt::operator |= 148
CNInt::operator + ... 148
CNInt::operator += 148
CNInt::operator ++ 147
CNInt::operator >>= 148
CNInt::operator ^= 148
CNInt::operator <<= 148
CNInt::operator long() 147
CNInterTab 36
CNInterTab() 36
CNInterTab::CNInterTab() 36
CNInterTab::operator() 36
CNJob 83
CNJob() 83
CNJob::CNJob() 83
CNKey 118
CNKey() 118
CNKey::CNKey() 118
CNKey::compare() 118
CNKey::get_object() 118
CNKey::hash() 118
CNKey::set_object() 118
CNKeyInt 120
CNKeyInt() 120
CNKeyInt::CNKeyInt() 120
CNKeyInt::compare() 120
CNKeyInt::get_key() 120
CNKeyInt::hash() 120
CNKeyString 119
CNKeyString() 119
CNKeyString::CNKeyString() 119
CNKeyString::compare() 119
CNKeyString::get_key() 119
CNKeyString::hash() 119
CNLCG 21
CNLCG() 21
CNLCG::as_long32() 21
CNLCG::CNLCG() 21
CNLCG::has_long32() 21
CNLCG::reset() 21
CNLCG::seed() 21
CNLogNormal 37
CNLogNormal() 37
CNLogNormal::CNLogNormal() 37
CNLogNormal::mean() 37
CNLogNormal::operator() 37
CNLogNormal::variance() 37
CNLREF 54
CNLREF() 54
CNLREF::CNLREF() 54
CNLREF::status() 55
CNLREG 54
CNLREG() 54
CNLREG::CNLREG() 54
CNLREG::status() 55
CNManager 130
CNManager() 130
CNManager::~CNManager() 130
CNManager::CNManager() 130
CNManager::delete_object() 130
CNManager::get_name() 131
CNManager::get_object() 131
CNManager::is_empty() 131
CNManager::new_object() 130
CNMDeterm 38
CNMDeterm() 38
CNMDeterm::CNMDeterm() 38
CNMDeterm::operator() 38
CNMLCG 22
CNMLCG() 22
CNMLCG::as_long32() 22
CNMLCG::CNMLCG() 22
CNMLCG::has_long32() 22
CNMLCG::reseed() 22
CNMLCG::reset() 22
CNMLCG::seed1() 22
CNMLCG::seed2() 22
CNMoments 52
CNMoments() 52
CNMoments::M_2() 52
CNMoments::M_3() 52
CNMoments::mean() 52
CNMoments::Moments() 52
CNMoments::relative_deviation() 53
CNMoments::relative_variance(); 53
CNMoments::skewness() 53
CNMoments::variance() 52
CNMoments::Z_3() 53
CNNamed 142
CNNamed() 142
CNNamed::CNNamed() 142
CNNamed::get_name() 142
CNNamed::name() 142

CNNamed::set_name() ... 142
CNNegExp ... 39
CNNegExp() ... 39
CNNegExp::CNNegExp() ... 39
CNNegExp::mean() ... 39
CNNegExp::operator() ... 39
CNNormal ... 40
CNNormal() ... 40
CNNormal::CNNormal() ... 40
CNNormal::mean() ... 40
CNNormal::operator() ... 40
CNNormal::variance() ... 40
CNObjecct::store_on() ... 11
CNObject ... 11
CNObject::class_desc() ... 11
CNObject::dump() ... 11
CNObject::is_a() ... 11
CNObject::print() ... 11
CNObject::storer() ... 11
CNParam ... 14
CNpersistent ... 213
CNPInt ... 220
CNPInt() ... 220
CNPInt::CNPInt() ... 220
CNPInt::object_id() ... 220
CNPInt::object_read_from() ... 220
CNPInt::read_from() ... 220
CNPInt::reader() ... 221
CNPInt::store_on() ... 220
CNPInt::storer() ... 220
CNPIO ... 217
CNPIO::read_id() ... 217
CNPIO::read_object() ... 217
CNPIO::store_id() ... 217
CNPIO::store_object() ... 217
CNPiostream ... 219
CNPiostream() ... 219
CNPiostream::CNPiostream() ... 219
CNPipe ... 152
CNPipe() ... 152
CNPipe::close() ... 152
CNPipe::CNPipe() ... 152
CNPipe::fd_in() ... 152
CNPipe::fd_out ... 152
CNPipe::open() ... 152
CNPObjectID ... 216
CNPObjectID() ... 216
CNPObjectID::CNPObjectID() ... 216
CNPObjectID::object_id() ... 216
CNPObjectID::PID ... 216
CNPoisson ... 41
CNPoisson() ... 41
CNPoisson::CNPoisson() ... 41
CNPoisson::mean() ... 41
CNPoisson::operator() ... 41
CNPrioQueueFIFO ... 81
CNPrioQueueFIFO() ... 81
CNPrioQueueFIFO::CNPrioQueueFIFO() ... 81
CNPrioQueueFIFO::delete_all() ... 82
CNPrioQueueFIFO::empty() ... 81
CNPrioQueueFIFO::full() ... 81
CNPrioQueueFIFO::get() ... 81
CNPrioQueueFIFO::length() ... 81
CNPrioQueueFIFO::put() ... 81
CNPstream ... 218
CNPstream() ... 218
CNPstream::CNPstream() ... 218
CNPstream::getline() ... 218
CNQueue ... 74
CNQueue() ... 74
CNQueue::CNQueue() ... 74
CNQueue::delete_all() ... 74
CNQueue::empty() ... 74
CNQueue::full() ... 74
CNQueue::get() ... 74
CNQueue::length() ... 74
CNQueue::put() ... 74
CNQueueFIFO ... 75
CNQueueFIFO() ... 75
CNQueueFIFO::CNQueueFIFO() ... 75
CNQueueFIFO::delete_all() ... 75
CNQueueFIFO::empty() ... 75
CNQueueFIFO::full() ... 75
CNQueueFIFO::get() ... 75
CNQueueFIFO::length() ... 75
CNQueueFIFO::put() ... 75
CNQueueLIFO ... 76
CNQueueLIFO() ... 76
CNQueueLIFO::CNQueueLIFO() ... 76
CNQueueLIFO::delete_all() ... 76
CNQueueLIFO::empty() ... 76
CNQueueLIFO::full() ... 76
CNQueueLIFO::get() ... 76
CNQueueLIFO::length() ... 76
CNQueueLIFO::put() ... 76
CNQueueRandom ... 77
CNQueueRandom() ... 77
CNQueueRandom::CNQueueRandom() ... 77
CNQueueRandom::delete_all() ... 77
CNQueueRandom::empty() ... 77
CNQueueRandom::full() ... 77
CNQueueRandom::get() ... 77
CNQueueRandom::length() ... 77
CNQueueRandom::put() ... 77

CNQueueSPT 79
CNQueueSPT() 79
CNQueueSPT::CNQueueSPT() 79
CNQueueSPT::delete_all() 80
CNQueueSPT::empty() 79
CNQueueSPT::full() 79
CNQueueSPT::get() 79
CNQueueSPT::length() 79
CNQueueSPT::put() 79
CNRandom 26
CNRandom() 26
CNRandom::CNRandom() 26
CNRandom::generator() 26
CNRandom::operator() 26
CNRandomMix 42
CNRandomMix() 42
CNRandomMix::CNRandomMix() 42
CNRandomMix::operator() 42
CNRayleigh 43
CNRayleigh() 43
CNRayleigh::CNRayleigh() 43
CNRayleigh::operator() 43
CNReaderTbl 214
CNReaderTbl() 214
CNReaderTbl::add_to_id_tbl() 215
CNReaderTbl::CNReaderTbl() 214
CNReaderTbl::get_from_id_tbl() 215
CNReaderTbl::is_in() 215
CNReaderTbl::reader_pointer_by_classname() ... 215
CNReaderTbl::ReaderPtr 214
CNReaderTbl::reset_id_tbl() 215
CNRice 44
CNRice() 44
CNRice::CNRice() 44
CNRice::operator() 44
CNRndInt 24
CNRndInt() 24
CNRndInt::as_int() 25
CNRndInt::as_long() 24
CNRndInt::CNRndInt() 24
CNRndInt::generator() 24
CNRndInt::high() 24
CNRndInt::low() 24
CNRndInt::operator() 24
CNRNG 16
CNRNG() 16
CNRNG::as_double() 16
CNRNG::as_float() 16
CNRNG::as_long() 16
CNRNG::as_long32() 16
CNRNG::CNRNG() 16
CNRNG::has_long32() 16
CNRNG::reset() 16
CNSatistics::Phase 50
CNSDLChannel 201, 212
CNSDLChannel() 212
CNSDLChannel::CNSDLChannel() 212
CNSDLChannel::delay() 212
CNSDLChannel::disturb() 212
CNSDLChannel::get_channel_delay() 212
CNSDLChannel::set_channel_delay() 212
CNSDLEnvironment 210
CNSDLEnvironment() 210
CNSDLEnvironment::CNSDLEnvironment() 210
CNSDLEnvironment::deinstall() 211
CNSDLEnvironment::install() 211
CNSDLEnvironment::output() 210
CNSDLEnvironment::output_x() 210
CNSDLEnvironment::S() 211
CNSDLManager 201
CNSDLManager() 201
CNSDLManager::active() 202
CNSDLManager::channel() 202
CNSDLManager::CNSDLManager() 201
CNSDLManager::create() 202
CNSDLManager::event_handler() 202
CNSDLManager::get_data() 202
CNSDLManager::get_source_system() 202
CNSDLManager::now() 202
CNSDLManager::output() 202
CNSDLManager::output_x() 202
CNSDLManager::remove() 202
CNSDLManager::reset() 202
CNSDLManager::save() 202
CNSDLManager::scheduler() 202
CNSDLManager::set() 202
CNSDLManager::stop() 202
CNSDLProcess 205
CNSDLProcess() 205
CNSDLProcess::CNSDLProcess() 205
CNSDLProcess::environment() 205
CNSDLProcess::from_procedure() 206
CNSDLProcess::get_name() 206
CNSDLProcess::now() 206
CNSDLProcess::offspring() 205
CNSDLProcess::parent() 205
CNSDLProcess::procedure() 206
CNSDLProcess::process() 206
CNSDLProcess::self() 205
CNSDLProcess::sender() 205
CNSDLProcess::state() 205
CNSDLProcess::type() 205
CNSDLProcessData 207
CNSDLProcessData() 207

CNSDLProcessData::add_process() ... 208
CNSDLProcessData::CNSDLProcessData() ... 207
CNSDLProcessData::delete_process() ... 208
CNSDLProcessData::get_process_by_PId() ... 207
CNSDLProcessData::get_process_by_type() ... 207
CNSDLProcessData::is_of_this_type() ... 207
CNSDLProcessData::max_number() ... 207
CNSDLProcessData::type() ... 207
CNSDLSignal ... 203
CNSDLSignal() ... 203
CNSDLSignal::~CNSDLSignal() ... 203
CNSDLSignal::CNSDLSignal() ... 203
CNSDLSignal::get_max_signals() ... 203
CNSDLSignal::operator delete() ... 204
CNSDLSignal::operator new() ... 204
CNSDLSignal::receiver() ... 204
CNSDLSignal::sender() ... 204
CNSDLSignal::set_max_signals() ... 203
CNSDLSignal::system() ... 204
CNSDLTimer ... 209
CNSDLTimer() ... 209
CNSDLTimer::active() ... 209
CNSDLTimer::clear() ... 209
CNSDLTimer::CNSDLTimer() ... 209
CNSDLTimer::remove() ... 209
CNSDLTimer::reset() ... 209
CNSDLTimer::set() ... 209
CNSDLTimer::tell_ID() ... 209
CNSelect ... 153
CNSelect() ... 153
CNSelect::add_except() ... 153
CNSelect::add_read() ... 153
CNSelect::add_write() ... 153
CNSelect::CNSelect() ... 153
CNSelect::del_except() ... 153
CNSelect::del_read() ... 153
CNSelect::del_write() ... 153
CNSelect::select() ... 154
CNSelect::test_except() ... 154
CNSelect::test_read() ... 153
CNSelect::test_write() ... 153
CNSink ... 84
CNSink() ... 84
CNSink::CNSink() ... 84
CNSink::delete_all() ... 84
CNSink::empty() ... 84
CNSink::full() ... 84
CNSink::get() ... 84
CNSink::length() ... 84
CNSink::put() ... 84
CNStack ... 85
CNStack() ... 85
CNStack::clear() ... 85
CNStack::CNStack() ... 85
CNStack::depth() ... 85
CNStack::empty() ... 85
CNStack::pull() ... 85
CNStack::push() ... 85
CNStack::size() ... 85
CNStatistics ... 50
CNStatistics() ... 50
CNStatistics::CNStatistics() ... 50
CNStatistics::end() ... 51
CNStatistics::max() ... 51
CNStatistics::mean() ... 50
CNStatistics::min() ... 50
CNStatistics::put() ... 50
CNStatistics::reset() ... 51
CNStatistics::status() ... 51
CNStatistics::trials() ... 50
CNStatistics::Type ... 50
CNStatistics::variance() ... 50
CNString ... 138
CNString() ... 138
CNString::~CNString() ... 138
CNString::add() ... 139
CNString::after() ... 139
CNString::before() ... 139
CNString::capacity() ... 139
CNString::capitalize() ... 139
CNString::CNString() ... 138
CNString::default_extra ... 138
CNString::del() ... 139
CNString::downsearch() ... 140
CNString::icopy() ... 141
CNString::insert() ... 139
CNString::length() ... 139
CNString::matches() ... 140
CNString::operator != ... 141
CNString::operator () ... 140
CNString::operator = ... 140
CNString::operator == ... 141
CNString::operator [] ... 140
CNString::operator + ... 140
CNString::operator += ... 140
CNString::operator > ... 141
CNString::operator >= ... 141
CNString::operator >> ... 141
CNString::operator < ... 141
CNString::operator <= ... 141
CNString::operator char * ... 140
CNString::operator const char * ... 140
CNString::replace() ... 140
CNString::resize() ... 138

CNString::to_ascii() 139
CNString::to_lower() 139
CNString::to_upper() 139
CNString::upsearch() 140
CNStringR 141
CNTab 45
CNTab() 45
CNTab::CNTab() 45
CNTab::operator() 45
CNTausG 23
CNTausG() 23
CNTausG::as_long32() 23
CNTausG::CNTausG() 23
CNTausG::has_long32() 23
CNTausG::reset() 23
CNTextWin::cols() 168
CNTextWin::height() 168
CNTextWin::incx() 168
CNTextWin::incy() 168
CNTextWin::rows() 168
CNTextWin::width() 168
CNUniform 46
CNUniform() 46
CNUniform::CNUniform() 46
CNUniform::high() 46
CNUniform::low() 46
CNUniform::operator() 46
CNWeibull 47
CNWeibull() 47
CNWeibull::alpha() 47
CNWeibull::beta() 47
CNWeibull::CNWeibull() 47
CNWeibull::operator() 47
col_to_x() 169
color() 176, 177, 181
cols() 114, 168
compare() 65, 118, 119, 120
create() 202
create_iterator() 99, 101

D

deactivate() 181
debug_rules() 190
default_exit_handler() 9
default_extra 138
DEFAULT_HASH_TABLE_CAPACITY 121
defuzzy() 197
defuzzy_all() 190
deinstall() 211
del() 139
del_except() 153
del_read() 153
del_write() 153
delay() 212
delete_all() 63, 68, 74, 75, 76, 77, 80, 82, 84, 94
delete_all_w_obj() 68
delete_current_event() 102
delete_drawing() 165
delete_event() 93, 94, 96, 98, 100
delete_events() 93, 94, 96, 98, 100
delete_key() 122, 123, 125
delete_key_absolutely() 122, 124, 126
delete_obj() 160
delete_object() 68, 71, 130
delete_process() 208
depth() 85
deviation() 53
disturb() 212
downsearch() 140
draw_arc() 156
draw_bitmap() 157
draw_clear() 158
draw_dash_line() 156
draw_fill_arc() 157
draw_fill_rectangle() 157
draw_line() 156
draw_now() 157
draw_pie_arc() 157
draw_point() 156
draw_rectangle() 157
draw_text() 157
dump() 5, 11

E

empty() 63, 67, 74, 75, 76, 77, 79, 81, 84, 85
end() 51, 160
environment() 205
err_abort 8
err_default 9
err_fatal 8
err_ignore 8
err_warning 8
error() 9
ErrorType 8
evaluate() 190
evaluate_all() 190
event() 158
event_handler() 91, 92, 202
EZD 156
EZD() 156
EZD::delete_obj() 160
EZD::draw-now() 157
EZD::draw_arc() 156
EZD::draw_bitmap() 157

EZD::draw_clear() ... 158
EZD::draw_dash_line() ... 156
EZD::draw_fill_arc() ... 157
EZD::draw_fill_rctangle() ... 157
EZD::draw_line() ... 156
EZD::draw_pie_arc() ... 157
EZD::draw_point() ... 156
EZD::draw_rectangle() ... 157
EZD::draw_text() ... 157
EZD::event() ... 158
EZD::EZD() ... 156
EZD::get_lastxb() ... 160
EZD::get_lastxe() ... 160
EZD::get_lastyb() ... 160
EZD::get_lastye() ... 160
EZD::in() ... 156
EZD::out() ... 156
EZD::pause() ... 158
EZD::print_window() ... 158
EZD::restore_drawing() ... 158
EZD::save_drawing() ... 158
EZD::set_auto_resize() ... 165
EZD::set_scale() ... 158
EZD::test_event() ... 158
EZD::x2pix() ... 158
EZD::y2pix() ... 158
EZDBlock ... 172
EZDBlock() ... 172
EZDBlock::EZDBlock() ... 172
EZDBlock::off() ... 172
EZDBlock::on() ... 172
EZDBlock::redraw() ... 172
EZDDiag ... 170
EZDDiag() ... 170
EZDDiag::add() ... 171
EZDDiag::clear() ... 171
EZDDiag::EZDDiag() ... 170
EZDDiag::redraw() ... 171
EZDDiag::set_color_bar() ... 171
EZDDiag::set_color_draw() ... 171
EZDDiag::set_frame() ... 171
EZDDiag::stzle() ... 171
EZDDiagWin ... 166
EZDDiagWin() ... 166
EZDDiagWin::add() ... 167
EZDDiagWin::clear() ... 167
EZDDiagWin::EZDDiagWin() ... 166
EZDDiagWin::set() ... 167
EZDDiagWin::style() ... 167
EZDDrawing ... 161
EZDDrawing() ... 161
EZDDrawing::EZDDrawing() ... 161
EZDDrawing::set() ... 161
EZDObject ... 159
EZDObject() ... 159
EZDObject::arc() ... 160
EZDObject::bitmap() ... 160
EZDObject::end() ... 160
EZDObject::EZDObject() ... 159
EZDObject::fill_arc() ... 160
EZDObject::fill_rectangle() ... 160
EZDObject::line() ... 160
EZDObject::pie_arc() ... 160
EZDObject::point() ... 160
EZDObject::rectangle() ... 160
EZDObject::redraw() ... 160
EZDObject::start() ... 159
EZDObject::text() ... 160
EZDObject::x() ... 159
EZDObject::y() ... 159
EZDPopUp ... 173
EZDPopUp() ... 173
EZDPopUp::EZDPopUp() ... 173
EZDPushButton ... 162
EZDPushButton() ... 162
EZDPushButton::EZDPushbutton() ... 162
EZDPushButton::set_text() ... 163
EZDQueue ... 175
EZDQueue() ... 175
EZDQueue::color() ... 176
EZDQueue::EZDQueue() ... 175
EZDQueue::get_length() ... 176
EZDQueue::left() ... 175
EZDQueue::length() ... 176
EZDQueue::lower() ... 175
EZDQueue::redraw() ... 176
EZDQueue::rigth() ... 175
EZDQueue::set_length() ... 176
EZDQueue::upper() ... 175
EZDServer ... 177
EZDServer() ... 177
EZDServer::color() ... 177
EZDServer::EZDServer() ... 177
EZDServer::redraw() ... 177
EZDText ... 178
EZDText() ... 178
EZDText::redraw() ... 178
EZDText::set_text() ... 178
EZDText::set_text_val() ... 178
EZDTextt::EZDText() ... 178
EZDTextWin ... 168
EZDTextWin() ... 168
EZDTextWin::add() ... 169
EZDTextWin::clear() ... 169

EZDTextWin::col_to_x() ... 169
EZDTextWin::EZDTextWin() ... 168
EZDTextWin::hline() ... 169
EZDTextWin::row_to_y() ... 169
EZDTextWin::set() ... 169
EZDTextWin::vline() ... 169
EZDTimer() ... 180
EZDTimer::activate() ... 181
EZDTimer::active_color() ... 181
EZDTimer::angle() ... 181
EZDTimer::color() ... 181
EZDTimer::deactivate() ... 181
EZDTimer::EZDTimer() ... 180
EZDTimer::get_angle() ... 181
EZDTimer::get_offset_angle() ... 181
EZDTimer::left() ... 180
EZDTimer::lower() ... 180
EZDTimer::offset_angle() ... 181
EZDTimer::redraw() ... 180
EZDTimer::right() ... 180
EZDTimer::set_angle() ... 181
EZDTimer::set_offset_angle() ... 181
EZDTimer::upper() ... 180
EZDWindow ... 164
EZDWindow() ... 164
EZDWindow::~EZDWindow() ... 164
EZDWindow::delete_drawing() ... 165
EZDWindow::EZDWindow() ... 164
EZDWindow::heigth() ... 164
EZDWindow::overlay() ... 165
EZDWindow::print_window() ... 165
EZDWindow::scale_drawing() ... 165
EZDWindow::set_norm() ... 165
EZDWindow::set_zoom() ... 165
EZDWindow::underlay() ... 165
EZDWindow::wherex() ... 164
EZDWindow::wherey() ... 164
EZDWindow::width() ... 164

F

FALSE ... 9
fatal() ... 9
fd_in() ... 152
fd_out() ... 152
fill() ... 143, 144, 146
fill_arc() ... 160
fill_rectangle() ... 160
find() ... 62, 65
find_first() ... 63
first() ... 67, 73, 127
first_key() ... 127
first_object() ... 73, 128
fixed ... 145
format() ... 144, 146
from() ... 89
from_procedure() ... 206
full() ... 74, 75, 76, 77, 79, 81, 84
fuzzy_value() ... 186

G

generator() ... 24, 26
get() ... 74, 75, 76, 77, 79, 81, 84, 109, 111, 115, 192
get_alpha() ... 194
get_angle() ... 181
get_beta() ... 194
get_capacity() ... 122
get_certainty() ... 187
get_channel_delay() ... 212
get_cols() ... 114
get_data() ... 202
get_error() ... 9
get_fill() ... 143, 146
get_format() ... 144, 146
get_from() ... 89
get_from_id_tbl() ... 215
get_fuzzy_value() ... 186
get_id() ... 90
get_issued() ... 89
get_key() ... 119, 120, 121, 127
get_lastxb() ... 160
get_lastxe() ... 160
get_lastyb() ... 160
get_lastye() ... 160
get_length ... 176
get_lhs() ... 187
get_m1() ... 194
get_m2() ... 194
get_max_events() ... 89
get_max_signals() ... 203
get_mean() ... 196
get_membership() ... 186, 191, 192, 194, 195, 196
get_n() ... 192
get_n_in_vars() ... 189
get_n_lhs() ... 187
get_n_out_vars() ... 189
get_n_rhs() ... 187
get_n_rules() ... 189
get_name() ... 13, 131, 142, 206
get_next() ... 70
get_num_entries() ... 122
get_object() ... 71, 72, 89, 118, 121, 128, 131
get_offset_angle() ... 181
get_position() ... 72
get_precision() ... 146

get_prev() 70
get_priority() 89
get_process_by_PId() 207
get_process_by_type() 207
get_rhs() 187
get_rows() 114
get_scale() 135, 137
get_scheduled() 89
get_size() 108
get_source_system() 202
get_state() 92
get_to() 89
get_type() 89
get_value() 143, 146, 185
get_value_set() 186
get_version() 13
get_width() 144, 146
get_xmax() 186
get_xmin() 186
getline() 218

H

has_long32() 16, 18, 19, 20, 21, 22, 23
hash() 118, 119, 120
height() 168
heigth() 164
high() 24, 31, 46
hline() 169

I

icopy() 141
id() 90
in() 156
incx() 168
incy() 168
inference() 190
info() 9
insert() 139
insert_after() 68
insert_before() 68
install() 211
is_a() 4, 11
is_empty() 122, 131
is_full() 122
is_in() 215
is_of_this_type() 207
issued() 89

K

key() 127

L

last() 67, 73, 128
last_key() 128
last_object() 73, 128
left 143, 145
left() 65, 175, 180
length() 63, 68, 74, 75, 76, 77, 79, 81, 84, 139, 176
line() 160
low() 24, 31, 46
lower() 175, 180

M

M_2() 52
M_3() 52
matches() 140
matherr() 8
max() 51
max_number() 207
mean() 29, 32, 33, 35, 37, 39, 40, 41, 50, 52
MENU 173
min() 50
msg() 9

N

n() 28
name() 13, 142
new_object() 5, 13, 130
newfile() 20
next() 67, 70, 73, 128
next_event() 94, 102
next_key() 128
next_object() 73, 128
NIL 9
now() 93, 202, 206

O

object() 71, 72, 89, 128
object_id() 216, 220
object_read_from() 220
off() 172
offset_angle() 181
offspring() 205
ok() 68
on() 172
open() 152
operator != 141, 148
operator %= 148
operator &= 148
operator () 140
operator * 197
operator *= 148, 150
operator - 135, 137, 148, 149, 197

operator -- ... 73, 128, 147
operator -= ... 134, 136, 148, 149
operator /= ... 148, 150
operator = ... 140, 197
operator =() ... 110, 112
operator == ... 141
operator [] ... 140, 192
operator [] () ... 109, 112
operator |= ... 148
operator + ... 135, 137, 140, 148, 149, 197
operator += ... 134, 136, 140, 148, 149
operator ++ ... 73, 128, 147
operator > ... 141
operator >= ... 141
operator >> ... 141, 218
operator >>= ... 148
operator ^= ... 148
operator < ... 141
operator <= ... 141
operator << ... 12, 218
operator <<= ... 148
operator char * ... 140
operator const char * ... 140
operator delete() ... 90, 204
operator double() ... 149
operator long() ... 147
operator new() ... 90, 204
operator() ... 24, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47
out() ... 156
output() ... 202, 210
output_x() ... 202, 210
overlay() ... 165

P

parent() ... 205
pause() ... 158
peek_event() ... 94, 96, 99, 101
Phase ... 50
PId ... 201, 204
PID ... 216
pie_arc() ... 160
PMF ... 206
point() ... 160
position() ... 72
precision() ... 146
prepend() ... 68
prev() ... 67, 70, 73, 128
prev_key() ... 128
prev_object() ... 73, 128
print() ... 5, 11, 59
print_clause() ... 184
print_membership() ... 186
print_window() ... 158, 165
priority() ... 89
procedure() ... 206
process() ... 206
ProcessType ... 201
pull() ... 85
push() ... 85
put() ... 50, 74, 75, 76, 77, 79, 81, 84, 109, 111, 115, 192

R

rd_conf() ... 59
read_from() ... 220
read_id() ... 217
read_object() ... 217
reader() ... 221
reader_pointer_by_classname() ... 215
Reader_ptr ... 214
receiver() ... 204
rectangle() ... 160
redraw() ... 160, 171, 172, 176, 177, 178, 180
relative_variance() ... 53
rem_handler() ... 91
remove() ... 63, 202, 209
remove_first() ... 63
remove_object() ... 68
replace() ... 140
reseed() ... 22
reset() ... 16, 18, 19, 20, 21, 22, 23, 51, 72, 122, 127, 202, 209
reset_absolutely() ... 122
reset_id_tbl() ... 215
resize() ... 138
resolution() ... 189
restore_drawing() ... 158
right ... 143, 145
right() ... 65, 180
rigth() ... 175
root() ... 63
row_to_y() ... 169
rows() ... 114, 168

S

S() ... 211
save() ... 202
save_drawing() ... 158
scale_drawing() ... 165
scheduled() ... 89
scheduler() ... 92, 202
scientific ... 145
seed() ... 19, 21
seed1() ... 22

seed2() 22
select() 154
self() 205
send() 93
send_delay() 93
send_event() 93, 96, 98, 100
send_now() 93
sender() 204, 205
set() 161, 167, 169, 202, 209
set_angle() 181
set_auto_resize() 165
set_certainty() 187
set_channel_delay() 212
set_color_bar() 171
set_color_draw() 171
set_error() 9
set_exit_handler() 9
set_fill() 144, 146
set_format() 144, 146
set_frame() 171
set_from() 89
set_fuzzy_value() 186
set_length() 176
set_max_events() 89
set_max_signals() 203
set_name() 142
set_next() 70
set_norm() 165
set_object() 71, 89, 118
set_offset_angle() 181
set_precision() 146
set_prev() 70
set_priority() 89
set_scale() 135, 137, 158
set_scheduled() 89
set_size() 108, 109, 111, 114
set_state() 92
set_text() 163, 178
set_text_val() 178
set_to() 89
set_type() 89
set_value() 143, 146, 185
set_width() 144, 146
set_zoom() 165
showpoint 145
sigma() 59
SignalType 201, 204
size() 85, 114, 115
skewness() 53
sqr() 197
start() 97, 99, 101, 159
state() 92, 205
static int store_id(CNPstream& stream, CNObject& obj); 217
statistics() 96
status() 51, 55, 57, 59
stop() 97, 99, 101, 202
store_id() 217
store_key() 121, 123, 125
store_object() 217
store_on() 11, 220
storer() 11, 220
style() 167, 171
system() 204

T

tell_ID() 209
test_event() 158
test_except() 154
test_read() 153
test_write() 153
text() 160
time() 96, 99, 101
TimerType 201
to() 89
to_ascii() 139
to_lower() 139
to_upper() 139
trials() 50
TRUE 9
Type 50
type() 89, 205, 207

U

u() 28
undelay() 165
upper() 175, 180
upsearch() 140

V

value() 29, 143, 146, 185
variance() 32, 35, 37, 40, 50, 52
version() 13
vline() 169

W

warning() 9
wherex() 164
wherey() 164
width() 144, 146, 164, 168
wrong_parity() 20

X

x 134, 136

x() 159
x2pix() 158
xmax() 186
xmin() 186

Y

y 134, 136
y() 159
y2pix() 158

Z

Z_3() 53

Table of Contents

1 Introduction to the CNCL 1

1.1 CNCL Class Hierarchy 2
1.2 Common CNCL Member Functions 4
1.3 `CNgenclass` Script 6

2 The Basic Classes of the CNCL Hierarchy 7

2.1 CNCL — CNCL Static Members and Functions 8
2.2 CNObject — Root of the CNCL Hierarchy 11
2.3 CNClass — Class Description 13
2.4 CNParam — Abstract Parameter Base Class 14

3 Random Numbers 15

3.1 CNRNG — Abstract Random Number Generator Base Class 16
3.2 CNACG — Additive RNG 18
3.3 CNFiboG — Fibonacci RNG 19
3.4 CNFileG — Data File RNG 20
3.5 CNLCG — Linear Congruence RNG 21
3.6 CNMLCG — Multiple Linear Congruence RNG 22
3.7 CNTausG — Tausworth RNG 23
3.8 CNRndInt — Random Integers 24
3.9 CNRandom — Abstract Random Distribution Base Class 26
3.10 CNBeta — Beta Distribution 27
3.11 CNBinomial — Binomial Distribution 28
3.12 CNDeterm — Deterministic Distribution 29
3.13 CNDiracTab — Distribution from Table of CDF 30
3.14 CNDiscUniform — Discrete Uniform Distribution 31
3.15 CNErlang — Erlang-k Distribution 32
3.16 CNGeometric — Geometric Distributed Random Numbers 33
3.17 CNHyperExp — Hyperexponential Distribution 34
3.18 CNHyperGeom — Hypergeometrical Distribution 35
3.19 CNInterTab – Distribution from Table of CDF (Interpolated) 36
3.20 CNLogNormal — Log-normal Distribution 37
3.21 CNMDeterm — Random Mix of Deterministic Values 38
3.22 CNNegExp — Negative Exponential Distribution 39
3.23 CNNormal — Normal Distribution 40
3.24 CNPoisson — Poisson Distribution 41
3.25 CNRandomMix — Mix of Several `CNRandom` Distributions 42
3.26 CNRayleigh — Rayleigh Distribution 43
3.27 CNRice — Rice Distribution 44
3.28 CNTab — Distribution from Table of CDF 45
3.29 CNUniform — Uniform Distribution 46
3.30 CNWeibull — Weibull Distribution 47

4 Statistical Evaluation in CNCL 49

4.1 CNStatistics — Abstract Statistics Base Class 50

4.2 CNMoments — Moments Evaluation 52
4.3 CNLREF, CNLREG — Evaluation by LRE 54
4.4 CNDLRE — Evaluation by Discrete–LRE 56
4.5 CNBatches — Evaluation by Batch Means 58

5 Container Classes 61

5.1 CNAVLTree — AVL balanced binary search tree 62
5.2 CNAVLNode — Node for CNAVLTree 65
5.3 CNDLList — Doubly Linked List of Objects 67
5.4 CNDLObject — Node of Doubly Linked List 70
5.5 CNDLIterator — Iterator of Doubly Linked List 72
5.6 CNQueue — Abstract Queue Base Class 74
5.7 CNQueueFIFO — FIFO Queue 75
5.8 CNQueueLIFO — LIFO queue 76
5.9 CNQueueRandom — Random queue 77
5.10 CNQueueSPT — SPT queue 79
5.11 CNPrioQueueFIFO — Queue with priority 81
5.12 CNJob — Job object 83
5.13 CNSink — Kitchen Sink 84
5.14 CNStack — Stack 85

6 Event Driven Simulation 87

6.1 CNEvent — Generic Event 88
6.2 CNEventExploder — Send Events to multiple EventHandlers 91
6.3 CNEventHandler — Abstract Base Class for Event Handlers 92
6.4 CNEventList — List of Events 94
6.5 CNEventBaseSched — Abstract scheduler base class 96
6.6 CNEventScheduler — Event Scheduler 98
6.7 CNEventHeapSched — Event Scheduler using a heap 100
6.8 CNEventIterator — iterate through event list 102
6.9 CNSimTime — Simulation Time 103
6.10 Example of an Event Driven Simulation 104

7 Array Classes 107

7.1 CNArray — Abstract Array Base Class 108
7.2 CNArrayObject — Array of Pointer to CNObject 109
7.3 CNArrayInt — Array of Integer 111
7.4 CNArray<type> — Arrays of Other <Type>s 113
7.5 CNArray2 — Base class for 2-dimensional arrays 114
7.6 CNArray2Char — char array class 115
7.7 CNArray2<type> — 2 dimensional Arrays of Other <Type>s 116

8 Object Management 117

8.1 CNKey — Abstract Base Class for Object Management via Keys 118
8.2 CNKeyString — Object Management via String Keys 119
8.3 CNKeyInt — Object Management via Integer Keys 120
8.4 CNHashTable — Abstract Base Class for Hash Tables 121
8.5 CNHashStatic — Hash Tables with Static Capacity 123
8.6 CNHashDynamic — Hash Tables with Dynamic Capacity 125

8.7 CNHashIterator — Sequential Iterator for Hash Tables 127
8.8 CNManager – Object Management Frontend 130

9 Miscellaneous Classes 133

9.1 CNCoord — 2-Dimensional Coordinates 134
9.2 CNICoord — 2-Dimensional Integer Coordinates 136
9.3 CNString — Character String 138
9.4 CNNamed — CNObject with Name 142
9.5 CNFormInt — Integers as CNStrings 143
9.6 CNFormFloat — Doubles as CNStrings 145
9.7 CNInt — Integers derived from CNObject 147
9.8 CNDouble — Doubles derived from CNObject 149

10 Unix Classes 151

10.1 CNPipe — Unix Pipe 152
10.2 CNSelect — Class Interface to Select(2) System Call 153

11 EZD Interface Classes 155

11.1 EZD — Base Class for EZD Graphic Objects 156
11.2 EZDObject — Interface to EZD Object 159
11.3 EZDDrawing — Interface to EZD Drawings 161
11.4 EZDPushButton — Interface to EZD Push-Button 162
11.5 EZDWindow — Interface to EZD Window 164
11.6 EZDDiagWin — Extra window with x-y diagram 166
11.7 EZDTextWin — ezd window for easy text display 168
11.8 EZDDiag — x-y diagram as an EZDObject 170
11.9 EZDBlock — Block with small rectangles for bit display 172
11.10 EZDPopUp — Interface to EZD popup menu 173
11.11 EZDQueue — Graphical Representation of a Queue 175
11.12 EZDServer — Graphical Representation of a Server 177
11.13 EZDText — EZD Object with Text 178
11.14 EZDTimer — Graphical Representation of a Timer 180

12 Fuzzy classes 183

12.1 CNFClause — Clause of a fuzzy rule 184
12.2 CNFVar — Fuzzy variable 185
12.3 CNFRule — Fuzzy Rule 187
12.4 CNFRuleBase — Rule base and Fuzzy inference engine 189
12.5 CNFSet — Fuzzy set abstract base class 191
12.6 CNFSetArray — Fuzzy set based on array with membership values 192
12.7 CNFSetLR — Fuzzy set with L and R functions 193
12.8 CNFSetTrapez — Fuzzy set with trapezium function 195
12.9 CNFSetTriangle — Fuzzy set with triangle function 196
12.10 CNFNumTriangle — Fuzzy number (triangle) 197

13 Simulation of SDL protocols 199

13.1 CNSDLManager — The SDL Machine 201
13.2 CNSDLSignal — SDL Signal 203
13.3 CNSDLProcess — SDL Process (dynamic) 205
13.4 CNSDLProcessData — SDL Process (description) 207
13.5 CNSDLTimer — SDL Timer 209
13.6 CNSDLEnvironment — SDL Environment 210
13.7 CNSDLChannel — SDL Channel 212

14 Persistent Classes 213

14.1 CNReaderTbl – Table for adress of reader-function 214
14.2 PObjectID — ID-Managment for persistent Objects 216
14.3 CNPIO — persistent stream Object IO-formatting 217
14.4 CNPstream — abstract base class for persistent stream classes .. 218
14.5 CNPiostream — persistent iostream format 219
14.6 CNPInt — class persistent CNInt 220
14.7 CNP<type> — persistent types 222

Concept Index 223

Function Index 227