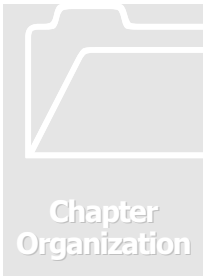
Chapter 7

7. CM3-IDE Interface Index

Read this chapter to learn about the most-frequently-used interfaces in CM3-IDE.



CM3-IDE includes many interfaces. Finding the right one for a particular task is not always easy. This index provides an overview of some of the most frequently used interfaces available in CM3-IDE.

Data types, Data Structures, and Algorithms on page 144 outlines:

- Basic Data Types
- Collections, Lists, Tables, Sets
- Linked Lists, Sorted Linked Lists
- Property lists
- Sequences
- Priority queues
- Sets
- Tables, Sorted tables
- Sorting Lists, Tables, and Arrays.

Standard Libraries on page 148 describes:

- Math, Geometry, Statistics, Random numbers
- Floating point
- Environment, Command line parameters
- I/O streams, Reading and Writing, Files
- Formatting, I/O Conversion
- Threads.

Systems Development on page 150 outlines:

- Distributed and Client/Server Development
- Databases and Persistence
- Operating System, Files, Processes, Time
- Interoperability with C
- Low-level Run-time Interfaces.

Miscellaneous on page 153 describes:

- Main interface, Weak References, Performance Tuning, Configuration.

7.1 Data Types, Data Structures, and Algorithms

7.1.1 Basic Data Types

The following table maps basic data types to the corresponding interfaces:

The on-line CM3-IDE interface index includes hypertext references to CM3-IDE interfaces. You can find it at:
`/help/interfaces.html`

Data Type	Interface
INTEGER	Integer, Int32
BOOLEAN	Boolean
CHAR	Char
REFANY	Refany
REAL	RealType, Real
LONGREAL	LongrealType, LongReal
EXTENDED	Extended
TEXT	Text

The list of all available interfaces in your CM3-IDE distribution is available at:
`/interface`

Interfaces for built-in types are used mainly as arguments for instantiating generic collections.

Text Strings. Strings are represented as values of type `TEXT`. Text strings are immutable; they are automatically garbage collected.

The `TEXT` type is used extensively throughout Modula-3 libraries. The `Text` interface defines the basic operations on this type. Operations to convert between other encodings of text strings are available in the `TextConv` interface. The internal representation of text strings is exposed by `TextF`. You should avoid using the `TextF` interface whenever possible.

To locate a particular interface within CM3-IDE, find
`/interface/name`
for example:
`/interface/Text`

ASCII Characters. ASCII includes constant definitions for the character codes of non-printable characters, such as `ASCII.NL` for new-line. It classifies characters into groups, like digits or punctuation; each group is represented as a set of characters. Finally, it provides mapping tables that translate lower-case letters into upper-case and vice versa.

Machine Words and Bit Manipulation. `Word` allows bit manipulation on machine words; `Swap` is useful for writing code that must deal explicitly with byte ordering and/or word length issues.

Atoms. While not built-in types, atoms are handy for efficient comparison of text strings. The `Atom` interface describes the set of operations available for atoms.

Symbolic Expressions. The `Sx` interface provides symbolic expressions represented as a recursive linked list structure, similar to Lisp systems. `Sx` includes routines for reading and printing symbolic expressions, as well as some convenience procedures for manipulating them.

See also **Formatting, I/O Conversion** on page 149 for interfaces that produce or parse text representations of the built-in types.

7.1.2 Collections, Lists, Tables, Sets

CM3-IDE libraries contain interfaces supporting the following data structures: linked lists, sorted linked lists, Lisp-like property lists, tables, sorted tables, sequences, priority queues, and sets.

Many of these data structures are available as generic interfaces. They can be instantiated with whatever types you need. For most of the generic data structures, the libraries already include instances for text strings, integers, atoms, and arbitrary references. Many of the generics are packaged with makefile commands that make it simply a matter of writing a line in your makefile to instantiate them.

7.1.3 Linked Lists

A generic implementation of singly-linked lists is available in the `List` interface, and implementation. There are predefined instances for atoms, integers, references, and text strings, as well as makefile commands to create custom lists.

7.1.4 Sorted Linked Lists

Lists may be sorted by using the generic `ListSort` interface and implementation. Like the `List` interfaces, there are predefined instances for atoms, integers, references, and text strings.

7.1.5 Property lists

Property lists, simple linked lists of (name, value) pairs are available from the `Property` interface. The related interfaces, `PropertyV`, `MProperty`, `PropertyF`, and `MPropertyF` provide more features.

Tables

A flexible and highly-reusable generic `Table` interface provides efficient mappings from values of one type to values of another. Like the list interfaces, the table interfaces are provided with predefined instances for the full cross product of the four basic types:

From \ To	atoms	integers	references	texts
atoms	`AtomAtomTbl`	`AtomIntTbl`	`AtomRefTbl`	`AtomTextTbl`
integers	`IntAtomTbl`	`IntIntTbl`	`IntRefTbl`	`IntTextTbl`
references	`RefAtomTbl`	`RefIntTbl`	`RefRefTbl`	`RefTextTbl`
texts	`TextAtomTbl`	`TextIntTbl`	`TextRefTbl`	`TextTextTbl`

Predefined makefile commands are available to create custom tables.

7.1.6 Sorted tables

Sorted tables are like tables with the addition of operations to iterate through the elements of the table in a sorted order. The generic `SortedTable` interface and implementation are available. Like the other table interfaces, the sorted table interfaces are provided with predefined instances for the full cross product of the four basic types:

From \ To	atoms	integers	references	texts
atoms	`Sorted-AtomAtomTbl`	`Sorted-AtomIntTbl`	`Sorted-AtomRefTbl`	`Sorted-AtomTextTbl`
integers	`Sorted-IntAtomTbl`	`Sorted-IntIntTbl`	`Sorted-IntRefTbl`	`Sorted-IntTextTbl`
references	`Sorted-RefAtomTbl`	`Sorted-RefIntTbl`	`Sorted-RefRefTbl`	`Sorted-RefTextTbl`
texts	`Sorted-TextAtomTbl`	`Sorted-TextIntTbl`	`Sorted-TextRefTbl`	`Sorted-TextTextTbl`

Predefined makefile commands are available to create custom sorted tables.

7.1.7 Sequences

The `Sequence` interface and implementation provide generic extensible arrays. Elements can be added or removed from either end or directly indexed.

The SequenceRep interface exposes the full details of the underlying representation of sequences. For maximum portability and implementation independence, programs should avoid using SequenceRep.

Predefined instances for atoms, integers, references, and text strings are available, so are the makefile commands to create custom sequences.

7.1.8 Priority queues

Priority queues, or sequences that keep their elements sorted, are available in the generic PQueue interface and implementation. The standard predefined instances for atoms, integers, references, and text strings are available.

When it is necessary to access the underlying implementation, the PQueueRep interface defines the full details. For maximum portability and implementation independence, programs should avoid using this interface. Predefined instances for atoms, integers, references, and text strings are available, as well as makefile commands to create custom priority queues.

7.1.9 Sets

Sets are collections of values without duplicates. A generic Set interface and implementation are available. Sets are implemented using two implementation strategies: SetDef uses a hash-table, and SetList uses a list representation for the set.

generic	integer	text	reference	atom
Set	IntSet	TextSet	RefSet	AtomSet
SetDef	IntSetDef	TextSetDef		AtomSetDef
SetList	IntSetList	TextSetList	RefSetList	AtomSetList

Predefined makefile commands are available to create custom sets.

See also **Atoms** on page 144 for an overview of the Atom interface, which provides a unique value for all equal text strings.

7.1.10 Sorting Lists, Tables, and Arrays

Sorted generic lists, tables, and arrays allow iteration in a sorted order.

SortedLists. The generic interface ListSort, implemented by generic module ListSort extends the List interface. As usual, there are instantiations AtomListSort, IntListSort, RefListSort, and TextListSort.

SortedTables. The interface SortedTable allows you to iterate through tables in a sorted order.

SortedArrays. `ArraySort` works similarly, but for arrays; it is instantiated as `IntArraySort` and `TextArraySort`.

7.2 Standard Libraries

7.2.1 Math, Geometry, Statistics, Random numbers

Modula-3 provides a rich set of interfaces for mathematical and statistical programming. The `Math` interface provides access to the C math libraries. Many geometric abstractions are also available: `Axis`, `Interval`, `Point`, `Rect`, `Transform`, `Path`, `Region`, `PolyRegion`, `Trapezoid`.

The generic interface `Sqrt` defines a square root operation, instantiated as `RealSqrt` and `LongSqrt` for `REAL` and `LONGREAL`s. The interface `Stat` defines a set of tools for collecting elementary statistics of a sequence of real quantities. The interface `Random` and `RandomPerm` provide random permutations of numbers. `RandomReal` includes machine specific algorithms for generating random floating-point values.

7.2.2 Floating point

`Real`, `LongReal`, and `Extended` are interfaces corresponding to the built-in floating-point types; their representations are in `RealRep` and `LongRealRep`.

The interface `FloatMode` allows you to test the behavior of rounding and of numerical exceptions. On some platforms it also allows you to change the behavior, on a per-thread basis.

The interface `FloatExtras`, `RealFloatExtras`, and `LongFloatExtras` contain miscellaneous functions useful for floating point arithmetic. The generic interface `Float` and its instantiations `RealFloat`, `LongFloat`, and `ExtendedFloat` provide interfaces to floating-point arithmetic.

`IEEESpecial` defines variables for the IEEE floating-point values `-infinity`, `+infinity`, and `NaN` (not a number) for each of the three floating-point types.

7.2.3 Environment, Command line parameters

The `Env` and `Params` interfaces provide access to the environment variables and command-line parameters given to a process when it is started.

See also the `Process` interface in **Processes, Pipes, O/S Errors** on page 152.

7.2.4 I/O streams, Reading and Writing, Files

I/O Streams allow you to read and write to disk, network, another thread, another process, etc.

Basic Input and Output. `IO` interface is a simple high-level I/O interface. `Stdio` declares the standard input, output, and error streams.

Input and Output Streams. `Rd` is the interface for input streams, known as readers. `RdClass` allows you to create new kinds of readers. `UnsafeRd` is an internal interface, providing non-serialized access to readers.

`Wr` is the safe interface to output streams, or writers. `WrClass` can be used to implement new streams. `UnsafeWr` allows unserialized access to a writer.

File Streams. `FileRd` and `FileWr` read from and write to files.

Text Streams. `TextRd` and `TextWr` read from and write to `TEXT` strings. They are designed for applying string procedures to streams or stream operations on strings.

Empty Streams. `NullRd` and `NullWr` represent empty streams.

Message Streams. `MsgRd` and `MsgWr` present message stream abstractions. A message is a sequence of bytes terminated by an end of message marker.

Stream and File Utilities. `TempFiles` creates temporary files which get deleted automatically upon termination of the process. `RdCopy` copies from readers to writers efficiently. `AutoFlushWr` flushes the output in the background. `RdUtils` adds a few utilities for manipulating readers.

7.2.5 Formatting, I/O Conversion

Most formatting interfaces in CM3-IDE work with strings, readers, and writers. `Fmt` formats basic data types to strings. `Scan` converts strings into basic data types. `FmtTime` returns a string denoting the current date and time. `FmtBuf` is similar to `Fmt` interface, with the exception that it uses character buffers instead of `TEXT` strings. `FmtBufF` exposes its representation.

`Lex` provides lexical operations for reading tokens and basic datatypes, and matching or skipping blanks from a reader. `Formatter` performs pretty-printing, the printing of structured objects with appropriate line breaks and indentation.

`Convert` converts binary and ASCII representation of basic values. `CConvert` provides lower-level access to the conversion functions in C.

7.2.6 Threads

CM3-IDE provides language-level support for multi-threaded applications. CM3-IDE's runtime and standard libraries on all platforms are multi-threaded. The `Thread` interface describes the portable interface for creating new threads (also called light-weight processes.) Interfaces `Scheduler`, `ThreadF`, and `ThreadContext` provide access to the internal representation of threads and some control over the thread scheduler.

7.3 Systems Development

7.3.1 Distributed and Client/Server Development

Network Objects. CM3-IDE's Network Objects system allows an object to be handed to another process in such away that the process receiving the object can operate on it as if it were local. The holder of a remote object can freely invoke operations on that object just as if it had created that object locally. Further, it can pass the object to other processes.

`NetObj` is the basic interface for defining network objects. A few makefile commands help you integrate network objects within your programs. The `NetObjNotifier` interface notifies a server if its clients die. `StubLib` contains procedures to be used by stub code for invoking remote object methods and servicing remote invocations.

The current implementation of Network Objects is built on top of TCP. `TCPNetObj` implements network objects on top of TCP/IP. Network Objects are designed to make adaptation to specialized network protocols easy.

Network Streams. Network Streams provide a set of high-level abstractions for sending and receiving messages across the network. `ConnRW` creates reader and writer streams from a connection; `ConnMsgRW` creates message streams from a connection.

TCP / IP Socket Interfaces. Using the TCP/IP interfaces, you can write safe, multi-threaded clients and servers for client/server computing. The same programs work whether you use Unix sockets or Windows winsock. The interface `IP` defines the addresses used for communicating with the internet protocol family. `TCP` provides bidirectional byte streams between two programs, implemented using internet protocols. `TCPSpecial` is a utility interface.

7.3.2 Databases and Persistence

Databases, Persistent Storage. CM3-IDE includes a number of facilities for saving data in persistent forms: Relational Databases, Pickles, Simple Snapshot Persistence, Stable Objects, and Bundles.

Relational Database Interface. The `DB` interface provides serialized access to relational databases. `DB` allows multiple connections within one application and each may be used concurrently by multiple threads. An implementation based on Microsoft's ODBC is available for both Windows and Unix; a sample implementation for Postgres'95 on Unix is also included. You can modify the backend of the interface to suit any relational database.

Pickles: ObjectTranscription (or "Serialization"). The `Pickle` interface provides operations for reading and writing arbitrary values as streams of bytes. Writing a value as a pickle and then reading it back produces a value equivalent to the original value. In

other words, pickles preserve value, shape and sharing. You can write pickles for values that have cyclic references (such as doubly-linked lists), or that are arbitrary graph structures.

Two implementations of the `Pickle` interface are available. `Pickle2` is an implementation geared toward heterogeneous platforms; it works across platforms, reconciling machine word encoding, little-endian, big-endian, size differences. `Pickle` is a more efficient implementation for homogeneous setups. Pickles are used by Network Objects for transferring objects across processes.

SmallDB: Simple Snapshot Persistence. `SmallDB` stores objects in a file in a recoverable fashion. If a crash occurs while the objects are being written to disk, their state can be restored from the latest consistent snapshot the next time they are used. For binary snapshots, use the combination of `SmallDB` and `Pickle`.

Stable Objects. Stable Objects extends the lightweight object storage provided by `Pickle`s and `SmallDB` to allow for recoverable storage of objects through logging and check-pointing. Updates to objects are logged to stable storage automatically. When the state of an object is restored from disk, the restoration process checks to see if a crash occurred before the entire state of the object was written to disk. If so, the state of the object is recovered from the log of modifications to the object.

The generic interface `Stable` defines a subtype of a given object type that is just like that object type, but stable. Makefile operations are provided to create stable versions arbitrary object types. `LogManager` manages readers and writers for the log and checkpoint files used by stable objects. `StableRep` defines the representation of stable objects. `Log` provides debugging operations for the log. `StableLog` contains procedures for reading and writing logs for stable objects.

Logs are written on readers and writers. `StableError` defines the various error scenarios and corresponding exceptions.

Bundles. Bundles package up arbitrary files at compile-time so that their contents can be retrieved by a program at run-time without accessing the file system. The interface `Bundle` allows runtime access to the stored data.

You can bundle files with your program by using operations defined in the bundle makefile templates. `BundleRep` exposes the representation of bundles.

7.3.3 Operating System, Files, Processes, Time

CM3-IDE provides a set of high-level, portable interfaces to the underlying OS facilities such as files, processes, directories, terminals, and keyboards. The interfaces to these operating system functions are identical whether you are running on Windows or Unix.

See also **Microsoft Windows** on page 152 and **Unix** on page 153 for lower-level, non-portable interfaces to operating system services.

File-system Interfaces. `File` defines a source and/or sink of bytes. File handles provide an operating-system independent way to perform raw I/O. For buffered I/O, use the `FileRd` and `FileWr` interfaces instead.

`Pathname` defines procedures for manipulating pathnames in a portable fashion. The `FS` interface provides access to persistent storage (files) and naming (directories).

`RegularFile` defines regular file handles which provide access to persistent extensible sequences of bytes—usually disks. A `Terminal` handle is a file handle that provides access to a duplex communication channel usually connected to a user terminal.

Processes, Pipes, O/S Errors. The `Process` interface manages operating-system processes (e.g., creating processes, awaiting their exit). A `Pipe` is a file handle that can be used to communicate between a parent and a child process or two sibling processes.

`OSError` defines an exception raised by a number of operating system interfaces.

Time, Date, Ticks. `Time` defines a moment in time, reckoned as a number of seconds since some epoch or starting point. `Date` defines a moment in time, expressed according to the standard (Gregorian) calendar, as observed in some time zone. `Tick` defines a value of a clock with sub-second resolution, typically one sixtieth of a second or smaller.

Timestamps, Capabilities, Fingerprints. Timestamps provided by the `TimeStamp` interface are totally ordered in a relation that approximates the real time when the value was generated. If two timestamps are generated in the same process then the ordering of the timestamps is consistent with the order that `TimeStamp.New` was called. The interface `Capability` defines unique global identifiers that are extremely difficult for an adversary to guess. The `Fingerprint` interface allows efficient comparison of large strings, and more general data structures such as graphs. `MachineID` returns a unique number for the machine running the Modula-3 program.

Platform-Specific Interfaces. Not all programs are portable. CM3-IDE allows access to lower-level interfaces available from the host operating system. Most of these interfaces are unsafe.

Microsoft Windows. The Windows distribution of CM3-IDE includes interfaces for accessing many of the calls from the Win32 API: `WinBaseTypes`, `WinDef`, `WinError`, `WinNT`, `WinBase`, `WinCon`, `WinGDI`, `WinNetwk`, `WinReg`, `WinUser`, `WinVer`, `NB30`, `CDErr`, `CommDlg`, `WinSock`.

Intermediate interfaces provide access to middle layers of Modula-3 libraries on Win32 are also available: `FileWin32`, `TimeWin32`, `OSWin32`, `TCPWin32`, `OSErrorWin32`.

Unix. The Unix distribution of CM3-IDE includes interfaces for accessing many of the calls from common Unix APIs: `Udir`, `Uipc`, `Uprocess`, `Usignal`, `Uugid`, `Uerror`, `Umman`, `Upwd`, `Usocket`, `Uuio`, `Uexec`, `Umsg`, `Uresource`, `Ustat`, `Uutmp`, `Ugrp`, `Unetdb`, `Usem`, `Utime`, `Uin`, `Unix`, `Ushm`, `Utypes`.

Intermediate interfaces provide access to middle layers of Modula-3 libraries on Unix are also available: `FilePosix`, `TimePosix`, `OSPosix`, `TCPPPosix`, `OSErrorPosix`.

7.3.4 Interoperability with C

Several standard C libraries are available from CM3-IDE. `Cerrno`, `Cstddef`, `Cstdlib`, `Ctypes`, `Cstdarg`, `Csetjmp`, `Cstdio`, and `Cstring` are Modula-3 interfaces for C standard libraries. `M3toC` converts between C strings and Modula-3 `TEXT` types.

7.3.5 Low-level Run-time Interfaces

Several interfaces provide low-level access to the run-time.

Allocator	`RTAllocator, RTAllocStats`
Heap Management	`RTHeap, RTHepDep, RTHeapInfo, RTHeapMap, RTHeapRep, RTHeapDebug, RTHeapStats, RTHeapEvent`
Garbage Collector	`RTCollector, RTCollectorSRC`
Type Management	`RTTipe, RTType, RTMapOp, RTTypeFP, RTTypeMap, RTUtils, RTTypeSRC`
Code and Execution	`RTLinker, RTProcedureSRC, RTModule, RTProcedure, RTException`
System Interface	`RTParams, RTArgs, RTProcess, RTStack, RTMachine`
Low-level	`RTHooks, RTO, RTSignal`
Miscellaneous	`RTIO, RTPacking, RTMisc`

7.4 Miscellaneous

Main interface. The `Main` interface is the entry point for executable programs. All programs must include a module that exports this interface.

Weak References. Using the `WeakRef` interface, you can register cleanup procedures to be run when the garbage collector is about to collect an object.

Performance Tuning. `ETimer` keeps track of elapsed time. It can be used for performance measurements. `PerfTool` and `LowPerfTool` control access to performance monitoring tools.

Configuration. The `M3Config` interface exports the configuration constants defined when the system was built.

Where to Go Next?

There are many more Modula-3 interfaces than described in this index. You may continue learning about Modula-3 interfaces by browsing the list of interfaces available in your CM3-IDE system.