Python Library Reference

Guido van Rossum

Dept. AA, CWI, P.O. Box 94079

1090 GB Amsterdam, The Netherlands

E-mail: guido@cwi.nl

13 October 1995

Release 1.3

Abstract

Python is an extensible, interpreted, object-oriented programming language. It supports a wide range of applications, from simple text processing scripts to interactive WWW browsers.

While the *Python Reference Manual* describes the exact syntax and semantics of the language, it does not describe the standard library that is distributed with the language, and which greatly enhances its immediate usability. This library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs.

This library reference manual documents Python's standard library, as well as many optional library modules (which may or may not be available, depending on whether the underlying platform supports them and on the configuration choices made at compile time). It also documents the standard types of the language and its built-in functions and exceptions, many of which are not or incompletely documented in the Reference Manual.

This manual assumes basic knowledge about the Python language. For an informal introduction to Python, see the *Python Tutorial*; the Python Reference Manual remains the highest authority on syntactic and semantic questions. Finally, the manual entitled *Extending and Embedding the Python Interpreter* describes how to add new extensions to Python and how to embed it in other applications.

Contents

Chapter 1

Introduction

The “Python library” contains several different kinds of components.

It contains data types that would normally be considered part of the “core” of a language, such as numbers and lists. For these types, the Python language core defines the form of literals and places some constraints on their semantics, but does not fully define the semantics. (On the other hand, the language core does define syntactic properties like the spelling and priorities of operators.)

The library also contains built-in functions and exceptions — objects that can be used by all Python code without the need of an import statement. Some of these are defined by the core language, but many are not essential for the core semantics and are only described here.

The bulk of the library, however, consists of a collection of modules. There are many ways to dissect this collection. Some modules are written in C and built in to the Python interpreter; others are written in Python and imported in source form. Some modules provide interfaces that are highly specific to Python, like printing a stack trace; some provide interfaces that are specific to particular operating systems, like socket I/O; others provide interfaces that are specific to a particular application domain, like the World-Wide Web. Some modules are avaiable in all versions and ports of Python; others are only available when the underlying system supports or requires them; yet others are available only when a particular configuration option was chosen at the time when Python was compiled and installed.

This manual is organized “from the inside out”: it first describes the built-in data types, then the built-in functions and exceptions, and finally the modules, grouped in chapters of related modules. The ordering of the chapters as well as the ordering of the modules within each chapter is roughly from most relevant to least important.

This means that if you start reading this manual from the start, and skip to the next chapter when you get bored, you will get a reasonable overview of the available modules and application areas that are supported by the Python library. Of course, you don’t *have* to read it like a novel — you can also browse the table of contents (in front of the manual), or look for a specific function, module or term in the index (in the back). And finally, if you enjoy learning about random subjects, you choose a random page number (see module rand) and read a section or two.

Let the show begin!

Chapter 2

Built-in Types, Exceptions and Functions

Built-in Objects

Names for built-in exceptions and functions are found in a separate symbol table. This table is searched last when the interpreter looks up the meaning of a name, so local and global user-defined names can override built-in names. Built-in types are described together here for easy reference.[1] built-intypes built-inexceptions built-infunctions type

The tables in this chapter document the priorities of operators by listing them in order of ascending priority (within a table) and grouping operators that have the same priority in the same box. Binary operators of the same priority group from left to right. (Unary operators group from right to left, but there you have no real choice.) See Chapter 5 of the Python Reference Manual for the complete picture on operator priorities.

2.1 Built-in Types

The following sections describe the standard types that are built into the interpreter. These are the numeric types, sequence types, and several others, including types themselves. There is no explicit Boolean type; use integers instead. built-intypes Booleantype

Some operations are supported by several object types; in particular, all objects can be compared, tested for truth value, and converted to a string (with the ‘…‘ notation). The latter conversion is implicitly used when an object is written by the print statement. print

2.1.1 Truth Value Testing

Any object can be tested for truth value, for use in an if or while condition or as operand of the Boolean operations below. The following values are considered false: if while truthvalue Booleanoperations

- None None
- zero of any numeric type, e.g., 0, 0L, 0.0.
- any empty sequence, e.g., ”, (), [].

1 Most descriptions sorely lack explanations of the exceptions that may be raised — this will be fixed in a future version of this manual.

- any empty mapping, e.g., {}.
- instances of user-defined classes, if the class defines a \s\do5(\s\do4())nonzero\s\do5(\s\do4())() or \s\do5(\s\do4())len\s\do5(\s\do4())() method, when that method returns zero.

All other values are considered true — so objects of many types are always true.

Operations and built-in functions that have a Boolean result always return 0 for false and 1 for true, unless otherwise stated. (Important exception: the Boolean operations or and and always return one of their operands.)

2.1.2 Boolean Operations

These are the Boolean operations, ordered by ascending priority: Booleanoperations

[Sorry. Ignored `\begin{tableiii} ... \end{tableiii}`]

and or not

Notes:

(1) These only evaluate their second argument if needed for their outcome.

(2) not has a lower priority than non-Boolean operators, so e.g. not a == b is interpreted as not(a == b), and a == not b is a syntax error.

2.1.3 Comparisons

Comparison operations are supported by all objects. They all have the same priority (which is higher than that of the Boolean operations). Comparisons can be chained arbitrarily, e.g. x < y <= z is equivalent to x < y and y <= z, except that y is evaluated only once (but in both cases z is not evaluated at all when x < y is found to be false). chainingcomparisons

This table summarizes the comparison operations:

[Sorry. Ignored `\begin{tableiii} ... \end{tableiii}`]

operatorcomparison == is is not

Notes:

(1) <> and != are alternate spellings for the same operator. (I couldn't choose between and ! :-) language language

Objects of different types, except different numeric types, never compare equal; such objects are ordered consistently but arbitrarily (so that sorting a heterogeneous array yields a consistent result). Furthermore, some types (e.g., windows) support only a degenerate notion of comparison where any two objects of that type are unequal. Again, such objects are ordered arbitrarily but consistently. typesnumeric objectscomparing

(Implementation note: objects of different types except numbers are ordered by their type names; objects of the same types that don't support proper comparison are ordered by their address.)

Two more operations with the same syntactic priority, in and not in, are supported only by sequence types (below). in not in

2.1.4 Numeric Types

There are three numeric types: plain integers, long integers, and floating point numbers. Plain integers (also just called integers) are implemented using long in , which gives them at least 32 bits of precision. Long integers have unlimited precision. Floating point numbers are implemented using double in . All bets on their precision are off unless you happen to know the machine you are working with. numerictypes integertypes integertype longintegertype floating pointtype language

Numbers are created by numeric literals or as the result of built-in functions and operators. Unadorned integer literals (including hex and octal numbers) yield plain integers. Integer literals with an L or l suffix yield long integers (L is preferred because 1l looks too much like eleven!). Numeric literals containing a decimal point or an exponent sign yield floating point numbers. numericliterals integerliterals longintegerliterals floating pointliterals hexadecimalliterals octalliterals

Python fully supports mixed arithmetic: when a binary arithmetic operator has operands of different numeric types, the operand with the "smaller" type is converted to that of the other, where plain integer is smaller than long integer is smaller than floating point. Comparisons between numbers of mixed type use the same rule.[2] The functions int(), long() and float() can be used to coerce numbers to a specific type. int long float

All numeric types support the following operations, sorted by ascending priority (operations in the same box have the same priority; all numeric operations have a higher priority than comparison operations):

[Sorry. Ignored `\begin{tableiii} ... \end{tableiii}`]

operations onnumerictypes

Notes:

(1) For (plain or long) integer division, the result is an integer. The result is always rounded towards minus infinity: 1/2 is 0, (-1)/2 is -1, 1/(-2) is -1, and (-1)/(-2) is 0. integerdivision longintegerdivision

(2) Conversion from floating point to (long or plain) integer may round or truncate as in ; see functions floor() and ceil() in module math for well-defined conversions. floor ceil numericconversions math language

(3) See the section on built-in functions for an exact definition.

2 As a consequence, the list [1, 2] is considered equal to [1.0, 2.0], and similar for tuples.

Bit-string Operations on Integer Types

Bit-string Operations

Plain and long integer types support additional operations that make sense only for bit-strings. Negative numbers are treated as their 2's complement value (for long integers, this assumes a sufficiently large number of bits that no overflow occurs during the operation).

The priorities of the binary bit-wise operations are all lower than the numeric operations and higher than the comparisons; the unary operation has the same priority as the other unary numeric operations (+ and -).

This table lists the bit-string operations sorted in ascending priority (operations in the same box have the same priority):

[Sorry. Ignored `\begin{tableiii} ... \end{tableiii}`]

operations onintegertypes bit-stringoperations shiftingoperations maskingoperations

Notes:

(1) Negative shift counts are illegal.

(2) A left shift by n bits is equivalent to multiplication by pow(2, n) without overflow check.

(3) A right shift by n bits is equivalent to division by pow(2, n) without overflow check.

2.1.5 Sequence Types

There are three sequence types: strings, lists and tuples.

Strings literals are written in single or double quotes: 'xyzzy', "frobozz". See Chapter 2 of the Python Reference Manual for more about string literals. Lists are constructed with square brackets, separating items with commas: [a, b, c]. Tuples are constructed by the comma operator (not within square brackets), with or without enclosing parentheses, but an empty tuple must have the enclosing parentheses, e.g., a, b, c or (). A single item tuple must have a trailing comma, e.g., (d,). sequencetypes stringtype tupletype listtype

Sequence types support the following operations. The in and not in operations have the same priorities as the comparison operations. The + and * operations have the same priority as the corresponding numeric operations.[3]

This table lists the sequence operations sorted in ascending priority (operations in the same box have the same priority). In the table, s and t are sequences of the same type; n, i and j are integers:

[Sorry. Ignored `\begin{tableiii} ... \end{tableiii}`]

operations onsequencetypes len min max concatenationoperation repetitionoperation subscriptoperation sliceoperation in not in

Notes:

3 They must have since the parser can't tell the type of the operands.

(1) If i or j is negative, the index is relative to the end of the string, i.e., len(s) + i or len(s) + j is substituted. But note that -0 is still 0.

(2) The slice of s from i to j is defined as the sequence of items with index k such that i <= k < j. If i or j is greater than len(s), use len(s). If i is omitted, use 0. If j is omitted, use len(s). If i is greater than or equal to j, the slice is empty.

More String Operations

String objects have one unique built-in operation: the % operator (modulo) with a string left argument interprets this string as a C sprintf format string to be applied to the right argument, and returns the string resulting from this formatting operation.

The right argument should be a tuple with one item for each argument required by the format string; if the string requires a single argument, the right argument may also be a single non-tuple object.[4] The following format characters are understood: %, c, s, i, d, u, o, x, X, e, E, f, g, G. Width and precision may be a * to specify that an integer argument specifies the actual width or precision. The flag characters -, +, blank, # and 0 are understood. The size specifiers h, l or L may be present but are ignored. The %s conversion takes any Python object and converts it to a string using str() before formatting it. The ANSI features %p and %n are not supported. Since Python strings have an explicit length, %s conversions don't assume that '0' is the end of the string.

For safety reasons, floating point precisions are clipped to 50; %f conversions for numbers whose absolute value is over 1e25 are replaced by %g conversions.[5] All other errors raise exceptions.

If the right argument is a dictionary (or any kind of mapping), then the formats in the string must have a parenthesized key into that dictionary inserted immediately after the % character, and each format formats the corresponding entry from the mapping. E.g.

```
     >>> count = 2
     >>> language = 'Python'
     >>> print '%(language)s has %(count)03d quote
types.' % vars()
     Python has 002 quote types.
     >>>
```

4 A tuple object in this case should be a singleton.

5 These numbers are fairly arbitrary. They are intended to avoid printing endless strings of meaningless digits without hampering correct use and without having to know the exact precision of floating point values on a particular machine.

In this case no * specifiers may occur in a format (since they a require sequential parameter list).

Additional string operations are defined in standard module string and in built-in module regex.

Mutable Sequence Types

List objects support additional operations that allow in-place modification of the object. These operations would be supported by other mutable sequence types (when added to the language) as well. Strings and tuples are immutable sequence types and such objects cannot be modified once created. The following operations are defined on mutable sequence types (where x is an arbitrary object): mutablesequencetypes listtype

[Sorry. Ignored `\begin{tableiii} ... \end{tableiii}`]

operations onmutablesequencetypes operations onsequencetypes operations onlisttype subscriptassignment sliceassignment del

append count index insert remove reverse sort

Notes:

(1) Raises an exception when x is not found in s.

(2) The sort() method takes an optional argument specifying a comparison function of two arguments (list items) which should return -1, 0 or 1 depending on whether the first argument is considered smaller than, equal to, or larger than the second argument. Note that this slows the sorting process down considerably; e.g. to sort a list in reverse order it is much faster to use calls to sort() and reverse() than to use sort() with a comparison function that reverses the ordering of the elements.

2.1.6 Mapping Types

A mapping object maps values of one type (the key type) to arbitrary objects. Mappings are mutable objects. There is currently only one standard mapping type, the dictionary. A dictionary's keys are almost arbitrary values. The only types of values not acceptable as keys are values containing lists or dictionaries or other mutable types that are compared by value rather than by object identity. Numeric types used for keys obey the normal rules for numeric comparison: if two numbers compare equal (e.g. 1 and 1.0) then they can be used interchangeably to index the same dictionary entry.

mappingtypes dictionarytype

Dictionaries are created by placing a comma-separated list of key: varvalue pairs within braces, for example: {'jack': 4098, 'sjoerd': 4127} or {4098: 'jack', 4127: 'sjoerd'}.

The following operations are defined on mappings (where a is a mapping, k is a key and x is an arbitrary object):

[Sorry. Ignored `\begin{tableiii} ... \end{tableiii}`]

operations onmappingtypes operations ondictionarytype del len

keys has\s\do5(k)ey

Notes:

(1) Raises an exception if k is not in the map.

(2) Keys and values are listed in random order.

2.1.7 Other Built-in Types

The interpreter supports several other kinds of objects. Most of these support only one or two operations.

Modules

The only special operation on a module is attribute access: m.name, where m is a module and name accesses a name defined in m's symbol table. Module attributes can be assigned to. (Note that the import statement is not, strictly spoken, an operation on a module object; import foo does not require a module object named foo to exist, rather it requires an (external) *definition* for a module named foo somewhere.)

A special member of every module is \s\do5(\s\do4())dict\s\do5(\s\do4()). This is the dictionary containing the module's symbol table. Modifying this dictionary will actually change the module's symbol table, but direct assignment to the \s\do5(\s\do4())dict\s\do5(\s\do4()) attribute is not possible (i.e., you can write m.\s\do5(\s\do4())dict\s\do5(\s\do4())['a'] = 1, which defines m.a to be 1, but you can't write m.\s\do5(\s\do4())dict\s\do5(\s\do4())= {}.

Modules are written like this: <module 'sys'>.

Classes and Class Instances

Classes and Instances

(See Chapters 3 and 7 of the Python Reference Manual for these.)

Functions

Function objects are created by function definitions. The only operation on a function object is to call it: func(argument-list).

There are really two flavors of function objects: built-in functions and user-defined functions. Both support the same operation (to call the function), but the implementation is different, hence the different object types.

The implementation adds two special read-only attributes: f.func\s\do5(c)ode is a function's code object (see below) and f.func\s\do5(g)lobals is the dictionary used as the function's global name space (this is the same as m.\s\do5(\s\do4())dict\s\do5(\s\do4()) where m is the module in which the function f was defined).

Methods

method

Methods are functions that are called using the attribute notation. There are two flavors: built-in methods (such as append() on lists) and class instance methods. Built-in methods are described with the types that support them.

The implementation adds two special read-only attributes to class instance methods: m.im\s\do5(s)elf is the object whose method this is, and m.im\s\do5(f)unc is the function implementing the method. Calling m(arg-1, arg-2, …, arg-n) is completely equivalent to calling m.im\s\do5(f)unc(m.im\s\do5(s)elf, arg-1, arg-2, …, arg-n).

(See the Python Reference Manual for more info.)

Code Objects

code

Code objects are used by the implementation to represent "pseudo-compiled" executable Python code such as a function body. They differ from function objects because they don't contain a reference to their global execution environment. Code objects are returned by the built-in compile() function and can be extracted from function objects through their func\s\do5(c)ode attribute. compile func\s\do5(c)ode

A code object can be executed or evaluated by passing it (instead of a source string) to the exec statement or the built-in eval() function. exec eval

(See the Python Reference Manual for more info.)

Type Objects

Type objects represent the various object types. An object's type is accessed by the built-in function type(). There are no special operations on types. The standard module types defines names for all standard built-in types. type types

Types are written like this: <type 'int'>.

The Null Object

This object is returned by functions that don't explicitly return a value. It supports no special operations. There is exactly one null object, named None (a built-in name).

It is written as None.

File Objects

File objects are implemented using 's stdio package and can be created with the built-in function open() described under Built-in Functions below. They are also returned by some other built-in functions and methods, e.g. posix.popen() and posix.fdopen() and the makefile() method of socket objects. open popen fdopen makefile

When a file operation fails for an I/O-related reason, the exception IOError is raised. This includes situations where the operation is not defined

for some reason, like seek() on a tty device or writing a file opened for reading.

Files have the following methods:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

Internal Objects

(See the Python Reference Manual for these.)

2.1.8 Special Attributes

The implementation adds a few special read-only attributes to several object types, where they are relevant:

- x.\s\do5(\s\do4())dict\s\do5(\s\do4()) is a dictionary of some sort used to store an object's (writable) attributes;
- x.\s\do5(\s\do4())methods\s\do5(\s\do4()) lists the methods of many built-in object types, e.g., [].\s\do5(\s\do4())methods\s\do5(\s\do4()) yields ['append', 'count', 'index', 'insert', 'remove', 'reverse', 'sort'];
- x.\s\do5(\s\do4())members\s\do5(\s\do4()) lists data attributes;
- x.\s\do5(\s\do4())class\s\do5(\s\do4()) is the class to which a class instance belongs;
- x.\s\do5(\s\do4())bases\s\do5(\s\do4()) is the tuple of base classes of a class object.

2.2 Built-in Exceptions

Exceptions are string objects. Two distinct string objects with the same value are different exceptions. This is done to force programmers to use exception names rather than their string value when specifying exception handlers. The string value of all built-in exceptions is their name, but this is not a requirement for user-defined exceptions or exceptions defined by library modules.

The following exceptions can be generated by the interpreter or built-in functions. Except where mentioned, they have an ‘associated value’ indicating the detailed cause of the error. This may be a string or a tuple containing several items of information (e.g., an error code and a string explaining the code).

User code can raise built-in exceptions. This can be used to test an exception handler or to report an error condition ‘just like’ the situation in which the interpreter raises the same exception; but beware that there is nothing to prevent user code from raising an inappropriate error.

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

2.3 Built-in Functions

The Python interpreter has a number of functions built into it that are always available. They are listed here in alphabetical order.

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

Chapter 3

Python Services

The modules described in this chapter provide a wide range of services related to the Python interpreter and its interaction with its environment. Here's an overview:

sys — Access system specific parameters and functions.

types — Names for all built-in types.

traceback — Print or retrieve a stack traceback.

pickle — Convert Python objects to streams of bytes and back.

shelve — Python object persistency.

copy — Shallow and deep copy operations.

marshal — Convert Python objects to streams of bytes and back (with different constraints).

imp — Access the implementation of the import statement.

\s\do5(\s\do4())builtin\s\do5(\s\do4()) — The set of built-in functions.

\s\do5(\s\do4())main\s\do5(\s\do4()) — The environment where the top-level script is run.

3.1 Built-in Module sys

sys This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It is always available.

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

3.2 Standard Module types

types

This module defines names for all object types that are used by the standard Python interpreter (but not for the types defined by various extension modules). It is safe to use "from types import *" — the module does not export any other names besides the ones listed here. New names exported by future versions of this module will all end in Type.

Typical use is for functions that do different things depending on their argument types, like the following:

```
from types import *
def delete(list, item):
     if type(item) is IntType:
         del list[item]
     else:
         list.remove(item)
```

The module defines the following names:
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]

3.3 Standard Module traceback

traceback

This module provides a standard interface to format and print stack traces of Python programs. It exactly mimics the behavior of the Python interpreter when it prints a stack trace. This is useful when you want to print stack traces under program control, e.g. in a “wrapper” around the interpreter.

The module uses traceback objects — this is the object type that is stored in the variables sys.exc\s\do5(t)raceback and sys.last\s\do5(t)raceback.

The module defines the following functions:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

3.4 Standard Module pickle

pickle persistentobjects serializingobjects marshallingobjects flatteningobjects picklingobjects

The pickle module implements a basic but powerful algorithm for “pickling” (a.k.a. serializing, marshalling or flattening) nearly arbitrary Python objects. This is the act of converting objects to a stream of bytes (and back: “unpickling”). This is a more primitive notion than persistency — although pickle reads and writes file objects, it does not handle the issue of naming persistent objects, nor the (even more complicated) area of concurrent access to persistent objects. The pickle module can transform a complex object into a byte stream and it can transform the byte stream into an object with the same internal structure. The most obvious thing to do with these byte streams is to write them onto a file, but it is also conceivable to send them across a network or store them in a database. The module shelve provides a simple interface to pickle and unpickle objects on “dbm”-style database files. shelve

Unlike the built-in module marshal, pickle handles the following correctly: marshal

- recursive objects (objects containing references to themselves)
- object sharing (references to the same object in different places)
- user-defined classes and their instances

The data format used by pickle is Python-specific. This has the advantage that there are no restrictions imposed by external standards such as CORBA (which probably can't represent pointer sharing or recursive objects); however it means that non-Python programs may not be able to reconstruct pickled Python objects.

The pickle data format uses a printable representation. This is slightly more voluminous than a binary representation. However, small integers actually take *less* space when represented as minimal-size decimal strings than when represented as 32-bit binary numbers, and strings are only much longer if they contain many control characters or 8-bit characters. The big advantage of using printable (and of some other characteristics of pickle's representation) is that for debugging or recovery purposes it is possible for a human to read the pickled file with a standard text editor. (I could have gone a step further and used a notation like S-expressions, but the parser (currently written in Python) would have been considerably more complicated and slower, and the files would probably have become much larger.)

The pickle module doesn't handle code objects, which the marshal module does. I suppose pickle could, and maybe it should, but there's probably no great need for it right now (as long as marshal continues to be used for reading and writing code objects), and at least this avoids the possibility of smuggling Trojan horses into a program. marshal

For the benefit of persistency modules written using pickle, it supports the notion of a reference to an object outside the pickled data stream. Such objects are referenced by a name, which is an arbitrary string of printable characters. The resolution of such names is not defined by the pickle module — the persistent object module will have to implement a method persistent\s\do5(l)oad. To write references to persistent objects, the persistent module must define a method persistent\s\do5(i)d which returns either None or the persistent ID of the object.

There are some restrictions on the pickling of class instances.

First of all, the class must be defined at the top level in a module.

Next, it must normally be possible to create class instances by calling the class without arguments. If this is undesirable, the class can define a method \s\do5(\s\do4())getinitargs\s\do5(\s\do4())(), which should return a *tuple* containing the arguments to be passed to the class constructor (\s\do5(\s\do4())init\s\do5(\s\do4())()). \s\do5(\s\do4())getinitargs\s\do5(\s\do4()) \s\do5(\s\do4())init\s\do5(\s\do4())

Classes can further influence how their instances are pickled — if the class defines the method \s\do5(\s\do4())getstate\s\do5(\s\do4())(), it is called and the return state is pickled as the contents for the instance, and if the class defines the method \s\do5(\s\do4())setstate\s\do5(\s\do4())(), it is called with the unpickled state. (Note that these methods can also be used to implement copying class instances.) If there is no \s\do5(\s\do4())getstate\s\do5(\s\do4())() method, the instance's \s\do5(\s\do4())dict\s\do5(\s\do4()) is pickled. If there is no \s\do5(\s\do4())setstate\s\do5(\s\do4())() method, the pickled object must be a dictionary and its items are assigned to the new instance's dictionary. (If a class defines both \s\do5(\s\do4())getstate\s\do5(\s\do4())() and \s\do5(\s\do4())setstate\s\do5(\s\do4())(), the state object needn't be a dictionary — these methods can do what they want.) This protocol is also used by the shallow and deep copying operations defined in the copy module. \s\do5(\s\do4())getstate\s\do5(\s\do4()) \s\do5(\s\do4())setstate\s\do5(\s\do4()) \s\do5(\s\do4())dict\s\do5(\s\do4())

Note that when class instances are pickled, their class's code and data are not pickled along with them. Only the instance data are pickled. This is done on purpose, so you can fix bugs in a class or add methods and still load objects that were created with an earlier version of the class. If you plan to have long-lived objects that will see many versions of a class, it may be worthwhile to put a version number in the objects so that suitable conversions can be made by the class's \s\do5(\s\do4())setstate\s\do5(\s\do4())() method.

When a class itself is pickled, only its name is pickled — the class definition is not pickled, but re-imported by the unpickling process. Therefore, the restriction that the class must be defined at the top level in a module applies to pickled classes as well.

The interface can be summarized as follows.

To pickle an object x onto a file f, open for writing:

```
p = pickle.Pickler(f)
p.dump(x)
```

A shorthand for this is:

```
pickle.dump(x, f)
```

To unpickle an object x from a file f, open for reading:

```
u = pickle.Unpickler(f)
x = u.load()
```

A shorthand is:

```
x = pickle.load(f)
```

The Pickler class only calls the method f.write with a string argument. The Unpickler calls the methods f.read (with an integer argument) and f.readline (without argument), both returning a string. It is explicitly allowed to pass non-file objects here, as long as they have the right methods. Unpickler Pickler

The following types can be pickled:

- None
- integers, long integers, floating point numbers
- strings
- tuples, lists and dictionaries containing only picklable objects
- classes that are defined at the top level in a module
- instances of such classes whose \s\do5(\s\do4())dict\s\do5(\s\do4()) or \s\do5(\s\do4())setstate\s\do5(\s\do4())() is picklable

Attempts to pickle unpicklable objects will raise the PicklingError exception; when this happens, an unspecified number of bytes may have been written to the file.

It is possible to make multiple calls to the dump() method of the same Pickler instance. These must then be matched to the same number of calls to the load() instance of the corresponding Unpickler instance. If the same object is pickled by multiple dump() calls, the load() will all yield references to the same object. *Warning*: this is intended for pickling multiple objects without intervening modifications to the objects or their parts. If you modify an object and then pickle it again using the same Pickler instance, the object is not pickled again — a reference to it is pickled and the Unpickler will return the old value, not the modified one. (There are two problems here: (a) detecting changes, and (b) marshalling a minimal set of changes. I have no answers. Garbage Collection may also become a problem here.)

Apart from the Pickler and Unpickler classes, the module defines the following functions, and an exception:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

3.5 Standard Module shelve

shelve pickle dbm gdbm

A “shelf” is a persistent, dictionary-like object. The difference with “dbm” databases is that the values (not the keys!) in a shelf can be essentially

arbitrary Python objects — anything that the pickle module can handle. This includes most class instances, recursive data types, and objects containing lots of shared sub-objects. The keys are ordinary strings.

To summarize the interface (key is a string, data is an arbitrary object):

```
import shelve

d = shelve.open(filename) # open, with (g)dbm
filename -- no suffix

d[key] = data    # store data at key (overwrites old
data if
                    # using an existing key)
data = d[key]    # retrieve data at key (raise
KeyError if no
                    # such key)
del d[key]        # delete data stored at key
(raises KeyError
                    # if no such key)
flag = d.has_key(key)    # true if the key exists
list = d.keys() # a list of all existing keys
(slow!)

d.close()         # close it
```

Restrictions:

- The choice of which database package will be used (e.g. dbm or gdbm) depends on which interface is available. Therefore it isn't safe to open the database directly using dbm. The database is also (unfortunately) subject to the limitations of dbm, if it is used — this means that (the pickled representation of) the objects stored in the database should be fairly small, and in rare cases key collisions may cause the database to refuse updates.
- Dependent on the implementation, closing a persistent dictionary may or may not be necessary to flush changes to disk.
- The shelve module does not support *concurrent* read/write access to shelved objects. (Multiple simultaneous read accesses are safe.) When a program has a shelf open for writing, no other program should have it open for reading or writing. file locking can be used to solve this, but this differs across versions and requires knowledge about the database implementation used.

3.6 Standard Module copy

copy

copy deepcopy

This module provides generic (shallow and deep) copying operations.

Interface summary:

```
import copy

x = copy.copy(y)           # make a shallow copy of y
x = copy.deepcopy(y)      # make a deep copy of y
```

For module specific errors, copy.error is raised.

The difference between shallow and deep copying is only relevant for compound objects (objects that contain other objects, like lists or class instances):

- A *shallow copy* constructs a new compound object and then (to the extent possible) inserts *references* into it to the objects found in the original.
- A *deep copy* constructs a new compound object and then, recursively, inserts *copies* into it of the objects found in the original.

Two problems often exist with deep copy operations that don't exist with shallow copy operations:

- Recursive objects (compound objects that, directly or indirectly, contain a reference to themselves) may cause a recursive loop.
- Because deep copy copies *everything* it may copy too much, e.g. administrative data structures that should be shared even between copies.

Python's deepcopy() operation avoids these problems by:

- keeping a table of objects already copied during the current copying pass; and
- letting user-defined classes override the copying operation or the set of components copied.

This version does not copy types like module, class, function, method, nor stack trace, stack frame, nor file, socket, window, nor array, nor any similar types.

Classes can use the same interfaces to control copying that they use to control pickling: they can define methods called \s\do5(\s\do4())getinitargs\s\do5(\s\do4())(), \s\do5(\s\do4())getstate\s\do5(\s\do4())() and \s\do5(\s\do4())setstate\s\do5(\s\do4())(). See the description of module pickle for information on these methods. pickle

\s\do5(\s\do4())getinitargs\s\do5(\s\do4())
\s\do5(\s\do4())getstate\s\do5(\s\do4()) \s\do5(\s\do4())setstate\s\do5(\s\do4())

3.7 Built-in Module marshal

marshal This module contains functions that can read and write Python values in a binary format. The format is specific to Python, but independent of machine architecture issues (e.g., you can write a Python value to a file on a PC, transport the file to a Sun, and read it back there). Details of the format are undocumented on purpose; it may change between Python versions (although it rarely does).[6]

This is not a general "persistency" module. For general persistency and transfer of Python objects through RPC calls, see the modules pickle and shelve. The marshal module exists mainly to support reading and writing the "pseudo-compiled" code for Python modules of .pyc files. pickle shelve code

Not all Python object types are supported; in general, only objects whose value is independent from a particular invocation of Python can be written and read by this module. The following types are supported: None, integers, long integers, floating point numbers, strings, tuples, lists, dictionaries, and code objects, where it should be understood that tuples, lists and dictionaries are only supported as long as the values contained therein are themselves supported; and recursive lists and dictionaries should not be written (they will cause infinite loops).

Caveat: On machines where C's long int type has more than 32 bits (such as the DEC Alpha), it is possible to create plain Python integers that are longer than 32 bits. Since the current marshal module uses 32 bits to transfer plain Python integers, such values are silently truncated. This particularly affects the use of very long integer literals in Python modules — these will be accepted by the parser on such machines, but will be silently be truncated when the module is read from the .pyc instead.[7]

There are functions that read/write files as well as functions operating on strings.

The module defines these functions:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

6 The name of this module stems from a bit of terminology used by the designers of Modula-3 (amongst others), who use the term "marshalling" for shipping of data around in a self-contained form. Strictly speaking, "to marshal" means to convert some data from internal to external form (in an RPC buffer for instance) and "unmarshalling" for the reverse process.

7 A solution would be to refuse such literals in the parser, since they are inherently non-portable. Another solution would be to let the marshal module raise an exception when an integer value would be truncated. At least one of these solutions will be implemented in a future version.

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

3.8 Built-in Module imp

imp

This module provides an interface to the mechanisms used to implement the import statement. It defines the following constants and functions:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

The following constants with integer values, defined in the module, are used to indicate the search result of imp.find\s\do5(m)odule.

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

3.8.1 Examples

The following function emulates the default import statement:

```
import imp
import sys

def __import__(name, globals=None, locals=None,
fromlist=None):
     # Fast path: see if the module has already been
imported.
     if sys.modules.has_key(name):
          return sys.modules[name]

     # If any of the following calls raises an
exception,
     # there's a problem we can't handle -- let the
caller handle it.
```

```
     # See if it's a built-in module.
     m = imp.init_builtin(name)
     if m:
          return m

     # See if it's a frozen module.
     m = imp.init_frozen(name)
     if m:
          return m

     # Search the default path (i.e. sys.path).
     fp, pathname, (suffix, mode, type) =
imp.find_module(name)

     # See what we got.
     try:
          if type == imp.C_EXTENSION:
               return imp.load_dynamic(name,
pathname)
          if type == imp.PY_SOURCE:
               return imp.load_source(name,
pathname, fp)
          if type == imp.PY_COMPILED:
               return imp.load_compiled(name,
pathname, fp)

          # Shouldn't get here at all.
          raise ImportError, '%s: unknown module
type (%d)' % (name, type)
     finally:
          # Since we may exit via an exception,
close fp explicitly.
          fp.close()
```

3.9 Built-in Module parser

parser

The parser module provides an interface to Python's internal parser and byte-code compiler. The primary purpose for this interface is to allow Python code to edit the parse tree of a Python expression and create executable code from this. This can be better than trying to parse and modify an arbitrary Python code fragment as a string, and ensures that parsing is performed in a manner identical to the code forming the application. It's also faster.

There are a few things to note about this module which are important to making use of the data structures created. This is not a tutorial on editing the parse trees for Python code.

Most importantly, a good understanding of the Python grammar processed by the internal parser is required. For full information on the language syntax, refer to the Language Reference. The parser itself is created from a grammar specification defined in the file Grammar/Grammar in the standard Python distribution. The parse trees stored in the "AST objects" created by this module are the actual output from the internal parser when created by the expr() or suite() functions, described below. The AST objects created by tuple2ast() faithfully simulate those structures.

Each element of the tuples returned by ast2tuple() has a simple form. Tuples representing non-terminal elements in the grammar always have a length greater than one. The first element is an integer which identifies a production in the grammar. These integers are given symbolic names in the C header file Include/graminit.h and the Python module Lib/symbol.py. Each additional element of the tuple represents a component of the production as recognized in the input string: these are always tuples which have the same form as the parent. An important aspect of this structure which should be noted is that keywords used to identify the parent node type, such as the keyword if in an *if_stmt*, are included in the node tree without any special treatment. For example, the if keyword is represented by the tuple (1, 'if'), where 1 is the numeric value associated with all NAME elements, including variable and function names defined by the user.

Terminal elements are represented in much the same way, but without any child elements and the addition of the source text which was identified. The example of the if keyword above is representative. The various types of terminal symbols are defined in the C header file Include/token.h and the Python module Lib/token.py.

The AST objects are not actually required to support the functionality of this module, but are provided for three purposes: to allow an application to amortize the cost of processing complex parse trees, to provide a parse tree representation which conserves memory space when compared to the Python tuple representation, and to ease the creation of additional modules in C which manipulate parse trees. A simple "wrapper" module may be created in Python if desired to hide the use of AST objects.

The parser module defines the following functions:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

3.9.1 Exceptions and Error Handling

The parser module defines a single exception, but may also pass other built-in exceptions from other portions of the Python runtime environment. See each function for information about the exceptions it can raise.

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

Note that the functions compileast(), expr(), and suite() may throw exceptions which are normally thrown by the parsing and compilation process. These include the built in exceptions MemoryError, OverflowError, SyntaxError, and SystemError. In these cases, these exceptions carry all the meaning normally associated with them. Refer to the descriptions of each function for detailed information.

3.9.2 Example

A simple example:

```
>>> import parser
>>> ast = parser.expr('a + 5')
>>> code = parser.compileast(ast)
>>> a = 5
>>> eval(code)
10
```

3.9.3 AST Objects

AST objects (returned by expr(), suite(), and tuple2ast(), described above) have no methods of their own. Some of the functions defined which accept an AST object as their first argument may change to object methods in the future.

Ordered and equality comparisons are supported between AST objects.

3.10 Built-in Module \s\do5(\s\do4())builtin\s\do5(\s\do4())

\s\do5(\s\do4())builtin\s\do5(\s\do4())

This module provides direct access to all 'built-in' identifiers of Python; e.g. \s\do5(\s\do4())builtin\s\do5(\s\do4()).open is the full name for the built-in function open. See the section on Built-in Functions in the previous chapter.

3.11 Built-in Module \s\do5(\s\do4())main\s\do5(\s\do4())

\s\do5(\s\do4())main\s\do5(\s\do4()) This module represents the (otherwise anonymous) scope in which the interpreter's main program executes — commands read either from standard input or from a script file.

Chapter 4

String Services

The modules described in this chapter provide a wide range of string manipulation operations. Here's an overview:

string — Common string operations.

regex — Regular expression search and match operations.

regsub — Substitution and splitting operations that use regular expressions.

struct — Interpret strings as packed binary data.

4.1 Standard Module string

string

This module defines some constants useful for checking character classes and some useful string functions. See the modules regex and regsub for string functions based on regular expressions.

The constants defined in this module are are:

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

The functions defined in this module are:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

This module is implemented in Python. Much of its functionality has been reimplemented in the built-in module strop. However, you should *never* import the latter module directly. When string discovers that strop exists, it transparently replaces parts of itself with the implementation from strop. After initialization, there is *no* overhead in using string instead of strop. strop

4.2 Built-in Module regex

regex This module provides regular expression matching operations similar to those found in Emacs. It is always available.

By default the patterns are Emacs-style regular expressions, with one exception. There is a way to change the syntax to match that of several well-known utilities. The exception is that Emacs' s pattern is not supported, since the original implementation references the Emacs syntax tables.

This module is 8-bit clean: both patterns and strings may contain null bytes and characters whose high bit is set.

Please note: There is a little-known fact about Python string literals which means that you don't usually have to worry about doubling backslashes, even though they are used to escape special characters in string literals as well as in regular expressions. This is because Python doesn't remove backslashes from string literals if they are followed by an unrecognized escape character. *However*, if you want to include a literal backslash in a regular expression represented as a string literal, you have to *quadruple* it. E.g. to extract LaTeX section{ ...} headers from a document, you can use this pattern: 'section{(.*)}'.

The module defines these functions, and an exception:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]

Compiled regular expression objects support these methods:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
Compiled regular expressions support these data attributes:
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]

4.3 Standard Module regsub

regsub This module defines a number of functions useful for working with regular expressions (see built-in module regex).
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

4.4 Built-in Module struct

struct Cstructures

This module performs conversions between Python values and C structs represented as Python strings. It uses format strings (explained below) as compact descriptions of the lay-out of the C structs and the intended conversion to/from Python values.

See also built-in module array. array

The module defines the following exception and functions:

[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

Format characters have the following meaning; the conversion between C and Python values should be obvious given their types:

[Sorry. Ignored \begin{tableiii} ... \end{tableiii}]

A format character may be preceded by an integral repeat count; e.g. the format string ’4h’ means exactly the same as ’hhhh’.

C numbers are represented in the machine’s native format and byte order, and properly aligned by skipping pad bytes if necessary (according to the rules used by the C compiler).

Examples (all on a big-endian machine):

```
pack('hhl', 1, 2, 3) ==
'\000\001\000\002\000\000\000\003'
```

```
unpack('hhl', '\000\001\000\002\000\000\000\003') ==
(1, 2, 3)
calcsize('hhl') == 8
```

Hint: to align the end of a structure to the alignment requirement of a particular type, end the format with the code for that type with a repeat count of zero, e.g. the format ’llh0l’ specifies two pad bytes at the end, assuming longs are aligned on 4-byte boundaries.

(More format characters are planned, e.g. ’s’ for character arrays, upper case for unsigned variants, and a way to specify the byte order, which is useful for [de]constructing network packets and reading/writing portable binary file formats like TIFF and AIFF.)

Chapter 5

Miscellaneous Services

The modules described in this chapter provide miscellaneous services that are available in all Python versions. Here's an overview:

math — Mathematical functions (sin() etc.).

rand — Integer random number generator.

whrandom — Floating point random number generator.

array — Efficient arrays of uniformly typed numeric values.

5.1 Built-in Module math

math

This module is always available. It provides access to the mathematical functions defined by the C standard. They are:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

acos(x), asin(x), atan(x), atan2(x y), ceil(x), cos(x), cosh(x), exp(x), fabs(x), floor(x), fmod(x y), frexp(x), hypot(x y), ldexp(x y), log(x), log10(x), modf(x), pow(x y), sin(x), sinh(x), sqrt(x), tan(x), tanh(x). "

Note that frexp and modf have a different call/return pattern than their C equivalents: they take a single argument and return a pair of values, rather than returning their second return value through an 'output parameter' (there is no such thing in Python).

The hypot function, which is not standard C, is not available on all platforms.

The module also defines two mathematical constants:

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

pi and e. "

5.2 Standard Module rand

rand This module implements a pseudo-random number generator with an interface similar to rand() in C. It defines the following functions:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

5.3 Standard Module whrandom

whrandom This module implements a Wichmann-Hill pseudo-random number generator. It defines the following functions:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

5.4 Built-in Module array

array

This module defines a new object type which can efficiently represent an array of basic values: characters, integers, floating point numbers. Arrays are sequence types and behave very much like lists, except that the type of objects stored in them is constrained. The type is specified at object creation time by using a type code, which is a single character. The following type codes are defined:

[Sorry. Ignored \begin{tableiii} ... \end{tableiii}]

The actual representation of values is determined by the machine architecture (strictly speaking, by the C implementation). The actual size can be accessed through the itemsize attribute.

See also built-in module struct. struct

The module defines the following function:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

Array objects support the following data items and methods:

[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]

[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

When an array object is printed or converted to a string, it is represented as array(typecode, initializer). The initializer is omitted if the array is empty, otherwise it is a string if the typecode is 'c', otherwise it is a list of numbers. The string is guaranteed to be able to be converted back to an array with the same type and value using reverse quotes (“). Examples:

```
array('l')
array('c', 'hello world')
array('l', [1, 2, 3, 4, 5])
```

```
array('d', [1.0, 2.0, 3.14])
```

Chapter 6

Generic Operating System Services

The modules described in this chapter provide interfaces to operating system features that are available on (almost) all operating systems, such as files and a clock. The interfaces are generally modelled after the or C interfaces but they are available on most other systems as well. Here's an overview:

os — Miscellaneous OS interfaces.

time — Time access and conversions.

getopt — Parser for command line options.

tempfile — Generate temporary file names.

6.1 Standard Module os

os This module provides a more portable way of using operating system (OS) dependent functionality than importing an OS dependent built-in module like posix.

When the optional built-in module posix is available, this module exports the same functions and data as posix; otherwise, it searches for an OS dependent built-in module like mac and exports the same functions and data as found there. The design of all Python's built-in OS dependent modules is such that as long as the same functionality is available, it uses the same interface; e.g., the function os.stat(file) returns stat info about a file in a format compatible with the POSIX interface.

Extensions peculiar to a particular OS are also available through the os module, but using them is of course a threat to portability!

Note that after the first time os is imported, there is *no* performance penalty in using functions from os instead of directly from the OS dependent built-in module, so there should be *no* reason not to use os!

In addition to whatever the correct OS dependent module exports, the following variables and functions are always exported by os:

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

(The functions os.execv() and execve() are not documented here, since they are implemented by the OS dependent module. If the OS dependent module doesn't define either of these, the functions that rely on it will raise an exception. They are documented in the section on module posix, together with all other functions that os imports from the OS dependent module.)

6.2 Built-in Module time

time This module provides various time-related functions. It is always available.

An explanation of some terminology and conventions is in order.

- The "epoch" is the point where the time starts. On January 1st of that year, at 0 hours, the "time since the epoch" is zero. For UNIX, the epoch is 1970. To find out what the epoch is, look at gmtime(0).
- UTC is Coordinated Universal Time (formerly known as Greenwich Mean Time). The acronym UTC is not a mistake but a compromise between English and French.
- DST is Daylight Saving Time, an adjustment of the timezone by (usually) one hour during part of the year. DST rules are magic (determined by local law) and can change from year to year. The C library has a table containing the local rules (often it is read from a system file for flexibility) and is the only source of True Wisdom in this respect.
- The precision of the various real-time functions may be less than suggested by the units in which their value or argument is expressed. E.g. on most UNIX systems, the clock "ticks" only 50 or 100 times a second, and on the Mac, times are only accurate to whole seconds.

The module defines the following functions and data items:

[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]

6.3 Standard Module getopt

getopt This module helps scripts to parse the command line arguments in sys.argv. It uses the same conventions as the getopt() function (including the special meanings of arguments of the form - and –). It defines the function getopt.getopt(args, options) and the exception getopt.error.

The first argument to getopt() is the argument list passed to the script with its first element chopped off (i.e., sys.argv[1:]). The second argument is the string of option letters that the script wants to recognize, with options that require an argument followed by a colon (i.e., the same format that getopt() uses). The return value consists of two elements: the first is a list of option-and-value pairs; the second is the list of program arguments left after the option list was stripped (this is a trailing slice of the first argument). Each option-and-value pair returned has the option as its first element, prefixed with a hyphen (e.g., '-x'), and the option argument as its second element, or an empty string if the option has no argument. The options occur in the list in the same order in which they were found, thus allowing multiple occurrences. Example:

```
>>> import getopt, string
>>> args = string.split('-a -b -cfoo -d bar a1 a2')
>>> args
['-a', '-b', '-cfoo', '-d', 'bar', 'a1', 'a2']
>>> optlist, args = getopt.getopt(args, 'abc:d:')
>>> optlist
[('-a', ''), ('-b', ''), ('-c', 'foo'), ('-d',
'bar')]
>>> args
['a1', 'a2']
>>>
```

The exception getopt.error = 'getopt error' is raised when an unrecognized option is found in the argument list or when an option requiring an argument is given none. The argument to the exception is a string indicating the cause of the error.

6.4 Standard Module tempfile

tempfile temporaryfile name temporaryfile

This module generates temporary file names. It is not specific, but it may require some help on non- systems.

Note: the modules does not create temporary files, nor does it automatically remove them when the current process exits or dies.

The module defines a single user-callable function:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

The module uses two global variables that tell it how to construct a temporary name. The caller may assign values to them; by default they are initialized at the first call to mktemp().

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
TMPDIR

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

Warning: if a process uses mktemp(), then calls fork() and both parent and child continue to use mktemp(), the processes will generate conflicting temporary names. To resolve this, the child process should assign None to template, to force recomputing the default on the next call to mktemp().

Chapter 7

Optional Operating System Services

The modules described in this chapter provide interfaces to operating system features that are available on selected operating systems only. The interfaces are generally modelled after the or C interfaces but they are available on some other systems as well (e.g. Windows or NT). Here's an overview:

signal — Set handlers for asynchronous events.

socket — Low-level networking interface.

select — Wait for I/O completion on multiple streams.

thread — Create multiple threads of control within one namespace.

7.1 Built-in Module signal

signal This module provides mechanisms to use signal handlers in Python. Some general rules for working with signals handlers:

- A handler for a particular signal, once set, remains installed until it is explicitly reset (i.e. Python uses the BSD style interface).
- There is no way to "block" signals temporarily from critical sections (since this is not supported by all flavors).
- Although Python signal handlers are called asynchronously as far as the Python user is concerned, they can only occur between the "atomic" instructions of the Python interpreter. This means that signals arriving during long calculations implemented purely in C (e.g. regular expression matches on large bodies of text) may be delayed for an arbitrary amount of time.
- When a signal arrives during an I/O operation, it is possible that the I/O operation raises an exception after the signal handler returns. This is dependent on the underlying system's semantics regarding interrupted system calls.
- Because the C signal handler always returns, it makes little sense to catch synchronous errors like SIGFPE or SIGSEGV.
- Python installs a small number of signal handlers by default: SIGPIPE is ignored (so write errors on pipes and sockets can be reported as ordinary

Python exceptions), SIGINT is translated into a KeyboardInterrupt exception, and SIGTERM is caught so that necessary cleanup (especially sys.exitfunc) can be performed before actually terminating. All of these can be overridden.

- Some care must be taken if both signals and threads are used in the same program. The fundamental thing to remember in using signals and threads simultaneously is: always perform signal() operations in the main thread of execution. Any thread can perform an alarm(), getsignal(), or pause(); only the main thread can set a new signal handler, and the main thread will be the only one to receive signals (this is enforced by the Python signal module, even if the underlying thread implementation supports sending signals to individual threads). This means that signals can't be used as a means of interthread communication. Use locks instead.

The variables defined in the signal module are:

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

The signal module defines the following functions:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

7.2 Built-in Module socket

socket This module provides access to the BSD *socket* interface. It is available on systems that support this interface.

For an introduction to socket programming (in C), see the following papers: *An Introductory 4.3BSD Interprocess Communication Tutorial*, by Stuart Sechrest and *An Advanced 4.3BSD Interprocess Communication Tutorial*, by Samuel J. Leffler et al, both in the Programmer's Manual, Supplementary Documents 1 (sections PS1:7 and PS1:8). The manual pages for the various socket-related system calls are also a valuable source of information on the details of socket semantics.

The Python interface is a straightforward transliteration of the system call and library interface for sockets to Python's object-oriented style: the socket() function returns a socket object whose methods implement the various socket system calls. Parameter types are somewhat higer-level than in the C interface: as with read() and write() operations on Python files, buffer allocation on receive operations is automatic, and buffer length is implicit on send operations.

Socket addresses are represented as a single string for the AF\s\do5(U)NIX address family and as a pair (host, port) for the AF\s\do5(I)NET address family, where host is a string representing either a hostname in Internet domain notation like ’daring.cwi.nl’ or an IP address like ’100.50.200.5’, and port is an integral port number. Other address families are currently not supported. The address format required by a particular socket object is automatically selected based on the address family specified when the socket object was created.

All errors raise exceptions. The normal exceptions for invalid argument types and out-of-memory conditions can be raised; errors related to socket or address semantics raise the error socket.error.

Non-blocking mode is supported through the setblocking() method.

The module socket exports the following constants and functions:

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

7.2.1 Socket Objects

Socket objects have the following methods. Except for makefile() these correspond to system calls applicable to sockets.

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

Note that there are no methods read() or write(); use recv() and send() without flags argument instead.

7.2.2 Example

Socket Example

Here are two minimal example programs using the TCP/IP protocol: a server that echoes all data that it receives back (servicing only one client), and a client using it. Note that a server must perform the sequence socket, bind, listen, accept (possibly repeating the accept to service more than one client), while a client only needs the sequence socket, connect. Also note that the server does not send/receive on the socket it is listening on but on the new socket returned by accept.

```
# Echo server program
from socket import *
HOST = ''                          # Symbolic name
meaning the local host
PORT = 50007                      # Arbitrary non-
privileged server
s = socket(AF_INET, SOCK_STREAM)
s.bind(HOST, PORT)
s.listen(1)
conn, addr = s.accept()
print 'Connected by', addr
while 1:
     data = conn.recv(1024)
     if not data: break
     conn.send(data)
conn.close()
```

```
# Echo client program
from socket import *
HOST = 'daring.cwi.nl'     # The remote host
PORT = 50007                  # The same port as
used by the server
s = socket(AF_INET, SOCK_STREAM)
s.connect(HOST, PORT)
s.send('Hello, world')
data = s.recv(1024)
s.close()
```

```
print 'Received', `data`
```

7.3 Built-in Module select

select

This module provides access to the function select available in most versions. It defines the following:

[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

socket stdwin

7.4 Built-in Module thread

thread

This module provides low-level primitives for working with multiple threads (a.k.a. light-weight processes or tasks) — multiple threads of control sharing their global data space. For synchronization, simple locks (a.k.a. mutexes or binary semaphores) are provided.

The module is optional and supported on SGI IRIX 4.x and 5.x and Sun Solaris 2.x systems, as well as on systems that have a PTHREAD implementation (e.g. KSR).

It defines the following constant and functions:

[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

Lock objects have the following methods:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

Caveats:

- Threads interact strangely with interrupts: the KeyboardInterrupt exception will be received by an arbitrary thread. (When the signal module is available, interrupts always go to the main thread.)
- Calling sys.exit() or raising the SystemExit is equivalent to calling thread.exit\s\do5(t)hread().
- Not all built-in functions that may block waiting for I/O allow other threads to run. (The most popular ones (sleep, read, select) work as expected.)

Chapter 8

UNIX Specific Services

The modules described in this chapter provide interfaces to features that are unique to the operating system, or in some cases to some or many variants of it. Here's an overview:

posix — The most common Posix system calls (normally used via module os).

posixpath — Common Posix pathname manipulations (normally used via os.path).

pwd — The password database (getpwnam() and friends).

grp — The group database (getgrnam() and friends).

dbm — The standard "database" interface, based on ndbm.

gdbm — GNU's reinterpretation of dbm.

termios — Posix style tty control.

fcntl — The fcntl() and ioctl() system calls.

posixfile — A file-like object with support for locking.

8.1 Built-in Module posix

posix

This module provides access to operating system functionality that is standardized by the C Standard and the POSIX standard (a thinly disguised interface).

Do not import this module directly. Instead, import the module os, which provides a *portable* version of this interface. On , the os module provides a superset of the posix interface. On non- operating systems the posix module is not available, but a subset is always available through the os interface. Once os is imported, there is *no* performance penalty in using it instead of posix. os

The descriptions below are very terse; refer to the corresponding manual entry for more information. Arguments called path refer to a pathname given as a string.

Errors are reported as exceptions; the usual exceptions are given for type errors, while errors reported by the system calls raise posix.error, described below.

Module posix defines the following data items:

[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]

[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]

It defines the following functions and constants:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

8.2 Standard Module posixpath

posixpath

This module implements some useful functions on POSIX pathnames.

Do not import this module directly. Instead, import the module os and use os.path. os

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

8.3 Built-in Module pwd

pwd This module provides access to the password database. It is available on all versions.

Password database entries are reported as 7-tuples containing the following items from the password database (see <pwd.h>), in order: pw\s\do5(n)ame, pw\s\do5(p)asswd, pw\s\do5(u)id, pw\s\do5(g)id, pw\s\do5(g)ecos, pw\s\do5(d)ir, pw\s\do5(s)hell. The uid and gid items are integers, all others are strings. An exception is raised if the entry asked for cannot be found.

It defines the following items:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

8.4 Built-in Module grp

grp This module provides access to the group database. It is available on all versions.

Group database entries are reported as 4-tuples containing the following items from the group database (see <grp.h>), in order: gr\s\do5(n)ame, gr\s\do5(p)asswd, gr\s\do5(g)id, gr\s\do5(m)em. The gid is an integer, name and password are strings, and the member list is a list of strings. (Note that most users are not explicitly listed as members of the group they are in according to the password database.) An exception is raised if the entry asked for cannot be found.

It defines the following items:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

8.5 Built-in Module dbm

dbm

The dbm module provides an interface to the (n)dbm library. Dbm objects behave like mappings (dictionaries), except that keys and values are always strings. Printing a dbm object doesn't print the keys and values, and the items() and values() methods are not supported.

See also the gdbm module, which provides a similar interface using the GNU GDBM library. gdbm

The module defines the following constant and functions:

[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

8.6 Built-in Module gdbm

gdbm

This module is nearly identical to the dbm module, but uses GDBM instead. Its interface is identical, and not repeated here.

Warning: the file formats created by gdbm and dbm are incompatible. dbm

8.7 Built-in Module termios

termios PosixI/O control ttyI/O control

This module provides an interface to the Posix calls for tty I/O control. For a complete description of these calls, see the Posix or manual pages. It is only available for those versions that support Posix termios style tty I/O control (and then only if configured at installation time).

All functions in this module take a file descriptor fd as their first argument. This must be an integer file descriptor, such as returned by sys.stdin.fileno().

This module should be used in conjunction with the TERMIOS module, which defines the relevant symbolic constants (see the next section).

The module defines the following functions:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

8.7.1 Example

termios Example

Here's a function that prompts for a password with echoing turned off. Note the technique using a separate termios.tcgetattr() call and a try … finally statement to ensure that the old tty attributes are restored exactly no matter what happens:

```
def getpass(prompt = "Password: "):
     import termios, TERMIOS, sys
     fd = sys.stdin.fileno()
     old = termios.tcgetattr(fd)
     new = termios.tcgetattr(fd)
     new[3] = new[3] & ~TERMIOS.ECHO             #
lflags
     try:
          termios.tcsetattr(fd, TERMIOS.TCSADRAIN,
new)
          passwd = raw_input(prompt)
     finally:
          termios.tcsetattr(fd, TERMIOS.TCSADRAIN,
old)
     return passwd
```

8.8 Standard Module TERMIOS

TERMIOS PosixI/O control ttyI/O control

This module defines the symbolic constants required to use the termios module (see the previous section). See the Posix or manual pages (or the source) for a list of those constants.

Note: this module resides in a system-dependent subdirectory of the Python library directory. You may have to generate it for your particular system using the script Tools/scripts/h2py.py.

8.9 Built-in Module fcntl

fcntl file control I/O control

This module performs file control and I/O control on file descriptors. It is an interface to the fcntl() and ioctl() routines. File descriptors can be obtained with the fileno() method of a file or socket object.

The module defines the following functions:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

If the library modules FCNTL or IOCTL are missing, you can find the opcodes in the C include files sys/fcntl and sys/ioctl. You can create the modules yourself with the h2py script, found in the Tools/scripts directory. FCNTL IOCTL

Examples (all on a SVR4 compliant system):

```
import struct, FCNTL

file = open(...)
rv = fcntl(file.fileno(), FCNTL.O_NDELAY, 1)

lockdata = struct.pack('hhllhh', FCNTL.F_WRLCK, 0,
0, 0, 0, 0)
rv = fcntl(file.fileno(), FCNTL.F_SETLKW, lockdata)
```

Note that in the first example the return value variable rv will hold an integer value; in the second example it will hold a string value.

8.10 Standard Module posixfile

posixfile posixfile object

This module implements some additional functionality over the built-in file objects. In particular, it implements file locking, control over the file flags, and an easy interface to duplicate the file object. The module defines a new file object, the posixfile object. It has all the standard file object methods

and adds the methods described below. This module only works for certain flavors of , since it uses fcntl() for file locking.

To instantiate a posixfile object, use the open() function in the posixfile module. The resulting object looks and feels roughly the same as a standard file object.

The posixfile module defines the following constants:

[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]

[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]

[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]

The posixfile module defines the following functions:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

The posixfile object defines the following additional methods:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

All methods return IOError when the request fails.

Format characters for the lock() method have the following meaning:

[Sorry. Ignored \begin{tableiii} ... \end{tableiii}]

In addition the following modifiers can be added to the format:

[Sorry. Ignored \begin{tableiii} ... \end{tableiii}]

Note:

(1) The lock returned is in the format (mode, len, start, whence, pid) where mode is a character representing the type of lock (’r’ or ’w’). This modifier prevents a request from being granted; it is for query purposes only.

Format character for the flags() method have the following meaning:

[Sorry. Ignored \begin{tableiii} ... \end{tableiii}]

In addition the following modifiers can be added to the format:

[Sorry. Ignored \begin{tableiii} ... \end{tableiii}]

Note:

(1) The ! and = modifiers are mutually exclusive.

(2) This string represents the flags after they may have been altered by the same call.

Examples:

```
from posixfile import *

file = open('/tmp/test', 'w')
file.lock('w|')
...
file.lock('u')
file.close()
```

8.11 Built-in Module syslog

syslog

This module provides an interface to the Unix syslog library routines. Refer to the manual pages for a detailed description of the syslog facility.

The module defines the following functions:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

The module defines the following constants:

Priority levels (high to low): LOG\s\do5(E)MERG, LOG\s\do5(A)LERT, LOG\s\do5(C)RIT, LOG\s\do5(E)RR, LOG\s\do5(W)ARNING, LOG\s\do5(N)OTICE, LOG\s\do5(I)NFO, LOG\s\do5(D)EBUG.

Facilities: LOG\s\do5(K)ERN, LOG\s\do5(U)SER, LOG\s\do5(M)AIL, LOG\s\do5(D)AEMON, LOG\s\do5(A)UTH, LOG\s\do5(L)PR, LOG\s\do5(N)EWS, LOG\s\do5(U)UCP, LOG\s\do5(C)RON and LOG\s\do5(L)OCAL0 to LOG\s\do5(L)OCAL7.

Log options: LOG\s\do5(P)ID, LOG\s\do5(C)ONS, LOG\s\do5(N)DELAY, LOG\s\do5(N)OWAIT and LOG\s\do5(P)ERROR if defined in syslog.h.

Chapter 9

The Python Debugger

pdb

The module pdb defines an interactive source code debugger for Python programs. It supports setting breakpoints and single stepping at the source line level, inspection of stack frames, source code listing, and evaluation of arbitrary Python code in the context of any stack frame. It also supports post-mortem debugging and can be called under program control.

The debugger is extensible — it is actually defined as a class Pdb. This is currently undocumented but easily understood by reading the source. The extension interface uses the (also undocumented) modules bdb and cmd. Pdb bdb cmd

A primitive windowing version of the debugger also exists — this is module wdb, which requires STDWIN (see the chapter on STDWIN specific modules). wdb

The debugger's prompt is "(Pdb) ". Typical usage to run a program under control of the debugger is:

```
>>> import pdb
>>> import mymodule
>>> pdb.run('mymodule.test()')
> <string>(0)?()
(Pdb) continue
> <string>(1)?()
(Pdb) continue
NameError: 'spam'
> <string>(1)?()
(Pdb)
```

Typical usage to inspect a crashed program is:

```
>>> import pdb
>>> import mymodule
>>> mymodule.test()
Traceback (innermost last):
   File "<stdin>", line 1, in ?
   File "./mymodule.py", line 4, in test
      test2()
   File "./mymodule.py", line 3, in test2
      print spam
NameError: spam
```

```
>>> pdb.pm()
> ./mymodule.py(3)test2()
-> print spam
(Pdb)
```

The module defines the following functions; each enters the debugger in a slightly different way:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

9.1 Debugger Commands

The debugger recognizes the following commands. Most commands can be abbreviated to one or two letters; e.g. “h(elp)” means that either “h” or “help” can be used to enter the help command (but not “he” or “hel”, nor “H” or “Help or “HELP”). Arguments to commands must be separated by whitespace (spaces or tabs). Optional arguments are enclosed in square brackets (“[]”) in the command syntax; the square brackets must not be typed. Alternatives in the command syntax are separated by a vertical bar (“—”).

Entering a blank line repeats the last command entered. Exception: if the last command was a “list” command, the next 11 lines are listed.

Commands that the debugger doesn’t recognize are assumed to be Python statements and are executed in the context of the program being debugged. Python statements can also be prefixed with an exclamation point (“!”). This is a powerful way to inspect the program being debugged; it is even possible to change a variable or call a function. When an exception occurs in such a statement, the exception name is printed but the debugger’s state is not changed.

h(elp) [command]

> Without argument, print the list of available commands. With a command as argument, print help about that command. “help pdb” displays the full documentation file; if the environment variable PAGER is defined, the file is piped through that command instead. Since the command argument must be an identifier, “help exec” must be entered to get help on the “!” command.

w(here) Print a stack trace, with the most recent frame at the bottom. An arrow indicates the current frame, which determines the context of most commands.

d(own) Move the current frame one level down in the stack trace (to an older frame).

u(p) Move the current frame one level up in the stack trace (to a newer frame).

b(reak) [lineno—function]

With a lineno argument, set a break there in the current file. With a function argument, set a break at the entry of that function. Without argument, list all breaks.

cl(ear) [lineno]

With a lineno argument, clear that break in the current file. Without argument, clear all breaks (but first ask confirmation).

s(tep) Execute the current line, stop at the first possible occasion (either in a function that is called or on the next line in the current function).

n(ext) Continue execution until the next line in the current function is reached or it returns. (The difference between next and step is that step stops inside a called function, while next executes called functions at (nearly) full speed, only stopping at the next line in the current function.)

r(eturn) Continue execution until the current function returns.

c(ont(inue)) Continue execution, only stop when a breakpoint is encountered.

l(ist) [first [, last]]

List source code for the current file. Without arguments, list 11 lines around the current line or continue the previous listing. With one argument, list 11 lines around at that line. With two arguments, list the given range; if the second argument is less than the first, it is interpreted as a count.

a(rgs) Print the argument list of the current function.

p expression Evaluate the expression in the current context and print its value. (Note: print can also be used, but is not a debugger command — this executes the Python print statement.)

[! statement]

Execute the (one-line) statement in the context of the current stack frame. The exclamation point can be omitted unless the first word of the statement resembles a debugger command. To set a global variable, you can prefix the assignment command with a “global” command on the same line, e.g.:

```
(Pdb) global list_options; list_options = ['-
l']
```

```
(Pdb)
```

q(uit) Quit from the debugger. The program being executed is aborted.

9.2 How It Works

Some changes were made to the interpreter:

- sys.settrace(func) sets the global trace function
- there can also a local trace function (see later)

Trace functions have three arguments: (frame, event, arg)

frame is the current stack frame

event is a string: 'call', 'line', 'return' or 'exception'

arg is dependent on the event type

A trace function should return a new trace function or None. Class methods are accepted (and most useful!) as trace methods.

The events have the following meaning:

'call' A function is called (or some other code block entered). The global trace function is called; arg is the argument list to the function; the return value specifies the local trace function.

'line' The interpreter is about to execute a new line of code (sometimes multiple line events on one line exist). The local trace function is called; arg in None; the return value specifies the new local trace function.

'return' A function (or other code block) is about to return. The local trace function is called; arg is the value that will be returned. The trace function's return value is ignored.

'exception' An exception has occurred. The local trace function is called; arg is a triple (exception, value, traceback); the return value specifies the new local trace function

Note that as an exception is propagated down the chain of callers, an 'exception' event is generated at each level.

Stack frame objects have the following read-only attributes:

f\s\do5(c)ode the code object being executed

f\s\do5(l)ineno the current line number (-1 for 'call' events)

f\s\do5(b)ack the stack frame of the caller, or None

f\s\do5(l)ocals dictionary containing local name bindings

f\s\do5(g)lobals dictionary containing global name bindings

Code objects have the following read-only attributes:

co\s\do5(c)ode the code string

co\s\do5(n)ames the list of names used by the code

co\s\do5(c)onsts the list of (literal) constants used by the code

co\s\do5(f)ilename the filename from which the code was compiled

Chapter 10

The Python Profiler

profile pstats

Written by James Roskind[8]

The profiler was written after only programming in Python for 3 weeks. As a result, it is probably clumsy code, but I don't know for sure yet 'cause I'm a beginner :-). I did work hard to make the code run fast, so that profiling would be a reasonable thing to do. I tried not to repeat code fragments, but I'm sure I did some stuff in really awkward ways at times. Please send suggestions for improvements to: jar@infoseek.com. I won't promise *any* support. ...but I'd appreciate the feedback.

10.1 Introduction to the profiler

Profiler Introduction

A profiler is a program that describes the run time performance of a program, providing a variety of statistics. This documentation describes the

8 Updated and converted to LaTeX by Guido van Rossum. The references to the old profiler are left in the text, although it no longer exists.

profiler functionality provided in the modules profile and pstats. This profiler provides deterministic profiling of any Python programs. It also provides a series of report generation tools to allow users to rapidly examine the results of a profile operation.

10.2 How Is This Profiler Different From The Old Profiler?

Profiler Changes

The big changes from old profiling module are that you get more information, and you pay less CPU time. It's not a trade-off, it's a trade-up.

To be specific:

Bugs removed: Local stack frame is no longer molested, execution time is now charged to correct functions.

Accuracy increased: Profiler execution time is no longer charged to user's code, calibration for platform is supported, file reads are not done *by* profiler *during* profiling (and charged to user's code!).

Speed increased: Overhead CPU cost was reduced by more than a factor of two (perhaps a factor of five), lightweight profiler module is all that must be loaded, and the report generating module (pstats) is not needed during profiling.

Recursive functions support: Cumulative times in recursive functions are correctly calculated; recursive entries are counted.

Large growth in report generating UI: Distinct profiles runs can be added together forming a comprehensive report; functions that import statistics take arbitrary lists of files; sorting criteria is now based on keywords (instead of 4 integer options); reports shows what functions were profiled as well as what profile file was referenced; output format has been improved.

10.3 Instant Users Manual

This section is provided for users that "don't want to read the manual." It provides a very brief overview, and allows a user to rapidly perform profiling on an existing application.

To profile an application with a main entry point of foo(), you would add the following to your module:

```
import profile
profile.run("foo()")
```

The above action would cause foo() to be run, and a series of informative lines (the profile) to be printed. The above approach is most useful when working with the interpreter. If you would like to save the results of a profile into a file for later examination, you can supply a file name as the second argument to the run() function:

```
import profile
profile.run("foo()", 'fooprof')
```

When you wish to review the profile, you should use the methods in the pstats module. Typically you would load the statistics data as follows:

```
import pstats
p = pstats.Stats('fooprof')
```

The class Stats (the above code just created an instance of this class) has a variety of methods for manipulating and printing the data that was just read into p. When you ran profile.run() above, what was printed was the result of three method calls:

```
p.strip_dirs().sort_stats(-1).print_stats()
```

The first method removed the extraneous path from all the module names. The second method sorted all the entries according to the standard module/line/name string that is printed (this is to comply with the semantics of the old profiler). The third method printed out all the statistics. You might try the following sort calls:

```
p.sort_stats('name')
p.print_stats()
```

The first call will actually sort the list by function name, and the second call will print out the statistics. The following are some interesting calls to experiment with:

```
p.sort_stats('cumulative').print_stats(10)
```

This sorts the profile by cumulative time in a function, and then only prints the ten most significant lines. If you want to understand what algorithms are taking time, the above line is what you would use.

If you were looking to see what functions were looping a lot, and taking a lot of time, you would do:

```
p.sort_stats('time').print_stats(10)
```

to sort according to time spent within each function, and then print the statistics for the top ten functions.

You might also try:

```
p.sort_stats('file').print_stats('__init__')
```

This will sort all the statistics by file name, and then print out statistics for only the class init methods (’cause they are spelled with \s\do5(\s\do4())init\s\do5(\s\do4()) in them). As one final example, you could try:

```
p.sort_stats('time', 'cum').print_stats(.5,
'init')
```

This line sorts statistics with a primary key of time, and a secondary key of cumulative time, and then prints out some of the statistics. To be specific, the list is first culled down to 50% (re: .5) of its original size, then only lines containing init are maintained, and that sub-sub-list is printed.

If you wondered what functions called the above functions, you could now (p is still sorted according to the last criteria) do:

```
p.print_callers(.5, 'init')
```

and you would get a list of callers for each of the listed functions.

If you want more functionality, you’re going to have to read the manual, or guess what the following functions do:

```
p.print_callees()
p.add('fooprof')
```

10.4 What Is Deterministic Profiling?

Deterministic Profiling

Deterministic profiling is meant to reflect the fact that all function call, function return, and exception events are monitored, and precise timings are made for the intervals between these events (during which time the user’s code is executing). In contrast, statistical profiling (which is not done by this module) randomly samples the effective instruction pointer, and deduces where time is being spent. The latter technique traditionally involves less overhead (as the code does not need to be instrumented), but provides only relative indications of where time is being spent.

In Python, since there is an interpreter active during execution, the presence of instrumented code is not required to do deterministic profiling.

Python automatically provides a hook (optional callback) for each event. In addition, the interpreted nature of Python tends to add so much overhead to execution, that deterministic profiling tends to only add small processing overhead in typical applications. The result is that deterministic profiling is not that expensive, yet provides extensive run time statistics about the execution of a Python program.

Call count statistics can be used to identify bugs in code (surprising counts), and to identify possible inline-expansion points (high call counts). Internal time statistics can be used to identify "hot loops" that should be carefully optimized. Cumulative time statistics should be used to identify high level errors in the selection of algorithms. Note that the unusual handling of cumulative times in this profiler allows statistics for recursive implementations of algorithms to be directly compared to iterative implementations.

10.5 Reference Manual

The primary entry point for the profiler is the global function profile.run(). It is typically used to create any profile information. The reports are formatted and printed using methods of the class pstats.Stats. The following is a description of all of these standard entry points and functions. For a more in-depth view of some of the code, consider reading the later section on Profiler Extensions, which includes discussion of how to derive "better" profilers from the classes presented, or reading the source code for these modules.

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

10.5.1 The Stats Class

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

10.6 Limitations

There are two fundamental limitations on this profiler. The first is that it relies on the Python interpreter to dispatch call, return, and exception events. Compiled C code does not get interpreted, and hence is "invisible" to the profiler. All time spent in C code (including builtin functions) will be charged

to the Python function that invoked the C code. If the C code calls out to some native Python code, then those calls will be profiled properly.

The second limitation has to do with accuracy of timing information. There is a fundamental problem with deterministic profilers involving accuracy. The most obvious restriction is that the underlying "clock" is only ticking at a rate (typically) of about .001 seconds. Hence no measurements will be more accurate that that underlying clock. If enough measurements are taken, then the "error" will tend to average out. Unfortunately, removing this first error induces a second source of error...

The second problem is that it "takes a while" from when an event is dispatched until the profiler's call to get the time actually *gets* the state of the clock. Similarly, there is a certain lag when exiting the profiler event handler from the time that the clock's value was obtained (and then squirreled away), until the user's code is once again executing. As a result, functions that are called many times, or call many functions, will typically accumulate this error. The error that accumulates in this fashion is typically less than the accuracy of the clock (i.e., less than one clock tick), but it *can* accumulate and become very significant. This profiler provides a means of calibrating itself for a given platform so that this error can be probabilistically (i.e., on the average) removed. After the profiler is calibrated, it will be more accurate (in a least square sense), but it will sometimes produce negative numbers (when call counts are exceptionally low, and the gods of probability work against you :-).) Do *NOT* be alarmed by negative numbers in the profile. They should *only* appear if you have calibrated your profiler, and the results are actually better than without calibration.

10.7 Calibration

The profiler class has a hard coded constant that is added to each event handling time to compensate for the overhead of calling the time function, and socking away the results. The following procedure can be used to obtain this constant for a given platform (see discussion in section Limitations above).

```
import profile
pr = profile.Profile()
pr.calibrate(100)
pr.calibrate(100)
pr.calibrate(100)
```

The argument to calibrate() is the number of times to try to do the sample calls to get the CPU times. If your computer is *very* fast, you might have to do:

```
pr.calibrate(1000)
```

or even:

```
pr.calibrate(10000)
```

The object of this exercise is to get a fairly consistent result. When you have a consistent answer, you are ready to use that number in the source code. For a Sun Sparcstation 1000 running Solaris 2.3, the magical number is about .00053. If you have a choice, you are better off with a smaller constant, and your results will "less often" show up as negative in profile statistics.

The following shows how the trace\s\do5(d)ispatch() method in the Profile class should be modified to install the calibration constant on a Sun Sparcstation 1000:

```
     def trace_dispatch(self, frame, event, arg):
          t = self.timer()
          t = t[0] + t[1] - self.t - .00053 #
Calibration constant

          if self.dispatch[event](frame,t):
               t = self.timer()
               self.t = t[0] + t[1]
          else:
               r = self.timer()
               self.t = r[0] + r[1] - t # put back
unrecorded delta
          return
```

Note that if there is no calibration constant, then the line containing the callibration constant should simply say:

```
          t = t[0] + t[1] - self.t   # no
calibration constant
```

You can also achieve the same results using a derived class (and the profiler will actually run equally fast!!), but the above method is the simplest to use. I could have made the profiler "self calibrating", but it would have made the initialization of the profiler class slower, and would have required some *very* fancy coding, or else the use of a variable where the constant .00053 was placed in the code shown. This is a VERY critical performance section, and there is no reason to use a variable lookup at this point, when a constant can be used.

10.8 Extensions — Deriving Better Profilers

Profiler Extensions

The Profile class of module profile was written so that derived classes could be developed to extend the profiler. Rather than describing all the details of such an effort, I'll just present the following two examples of derived classes that can be used to do profiling. If the reader is an avid Python programmer, then it should be possible to use these as a model and create similar (and perchance better) profile classes.

If all you want to do is change how the timer is called, or which timer function is used, then the basic class has an option for that in the constructor for the class. Consider passing the name of a function to call into the constructor:

```
pr = profile.Profile(your_time_func)
```

The resulting profiler will call your\s\do5(t)ime\s\do5(f)unc() instead of os.times(). The function should return either a single number or a list of numbers (like what os.times() returns). If the function returns a single time number, or the list of returned numbers has length 2, then you will get an especially fast version of the dispatch routine.

Be warned that you *should* calibrate the profiler class for the timer function that you choose. For most machines, a timer that returns a lone integer value will provide the best results in terms of low overhead during profiling. (os.times is *pretty* bad, 'cause it returns a tuple of floating point values, so all arithmetic is floating point in the profiler!). If you want to substitute a better timer in the cleanest fashion, you should derive a class, and simply put in the replacement dispatch method that better handles your timer call, along with the appropriate calibration constant :-).

10.8.1 OldProfile Class

The following derived profiler simulates the old style profiler, providing errant results on recursive functions. The reason for the usefulness of this profiler is that it runs faster (i.e., less overhead) than the old profiler. It still creates all the caller stats, and is quite useful when there is *no* recursion in the user's code. It is also a lot more accurate than the old profiler, as it does not charge all its overhead time to the user's code.

```
class OldProfile(Profile):

     def trace_dispatch_exception(self, frame, t):
          rt, rtt, rct, rfn, rframe, rcur = self.cur
          if rcur and not rframe is frame:
```

```
                return
self.trace_dispatch_return(rframe, t)
          return 0

     def trace_dispatch_call(self, frame, t):
          fn = `frame.f_code`

          self.cur = (t, 0, 0, fn, frame, self.cur)
          if self.timings.has_key(fn):
               tt, ct, callers = self.timings[fn]
               self.timings[fn] = tt, ct, callers
          else:
               self.timings[fn] = 0, 0, {}
          return 1

     def trace_dispatch_return(self, frame, t):
          rt, rtt, rct, rfn, frame, rcur = self.cur
          rtt = rtt + t
          sft = rtt + rct

          pt, ptt, pct, pfn, pframe, pcur = rcur
          self.cur = pt, ptt+rt, pct+sft, pfn,
pframe, pcur

          tt, ct, callers = self.timings[rfn]
          if callers.has_key(pfn):
               callers[pfn] = callers[pfn] + 1
          else:
               callers[pfn] = 1
          self.timings[rfn] = tt+rtt, ct + sft,
callers

          return 1


     def snapshot_stats(self):
          self.stats = {}
          for func in self.timings.keys():
               tt, ct, callers = self.timings[func]
               nor_func = self.func_normalize(func)
               nor_callers = {}
               nc = 0
               for func_caller in callers.keys():

nor_callers[self.func_normalize(func_caller)]=\
                            callers[func_caller]
```

```
                    nc = nc + callers[func_caller]
               self.stats[nor_func] = nc, nc, tt,
ct, nor_callers
```

10.8.2 HotProfile Class

This profiler is the fastest derived profile example. It does not calculate caller-callee relationships, and does not calculate cumulative time under a function. It only calculates time spent in a function, so it runs very quickly (re: very low overhead). In truth, the basic profiler is so fast, that is probably not worth the savings to give up the data, but this class still provides a nice example.

```
class HotProfile(Profile):

     def trace_dispatch_exception(self, frame, t):
          rt, rtt, rfn, rframe, rcur = self.cur
          if rcur and not rframe is frame:
               return
self.trace_dispatch_return(rframe, t)
          return 0

     def trace_dispatch_call(self, frame, t):
          self.cur = (t, 0, frame, self.cur)
          return 1

     def trace_dispatch_return(self, frame, t):
          rt, rtt, frame, rcur = self.cur

          rfn = `frame.f_code`

          pt, ptt, pframe, pcur = rcur
          self.cur = pt, ptt+rt, pframe, pcur

          if self.timings.has_key(rfn):
               nc, tt = self.timings[rfn]
               self.timings[rfn] = nc + 1, rt + rtt
+ tt
          else:
               self.timings[rfn] =       1, rt +
rtt

          return 1


     def snapshot_stats(self):
```

```
            self.stats = {}
            for func in self.timings.keys():
                 nc, tt = self.timings[func]
                 nor_func = self.func_normalize(func)
                 self.stats[nor_func] = nc, nc, tt,
0, {}
```

Chapter 11

Internet and WWW Services

Internet and WWW

The modules described in this chapter provide various services to World-Wide Web (WWW) clients and/or services, and a few modules related to news and email. They are all implemented in Python. Some of these modules require the presence of the system-dependent module sockets, which is currently only fully supported on Unix and Windows NT. Here is an overview:

cgi — Common Gateway Interface, used to interpret forms in server-side scripts.

urllib — Open an arbitrary object given by URL (requires sockets).

httplib — HTTP protocol client (requires sockets).

ftplib — FTP protocol client (requires sockets).

gopherlib — Gopher protocol client (requires sockets).

nntplib — NNTP protocol client (requires sockets).

urlparse — Parse a URL string into a tuple (addressing scheme identifier, network location, path, parameters, query string, fragment identifier).

htmllib — A (slow) parser for HTML files.

sgmllib — Only as much of an SGML parser as needed to parse HTML.

rfc822 — Parse RFC-822 style mail headers.

mimetools — Tools for parsing MIME style message bodies.

11.1 Standard Module cgi

cgi WWWserver CGIprotocol HTTPprotocol MIMEheaders

This module makes it easy to write Python scripts that run in a WWW server using the Common Gateway Interface. It was written by Michael McLay and subsequently modified by Steve Majewski and Guido van Rossum.

When a WWW server finds that a URL contains a reference to a file in a particular subdirectory (usually /cgibin), it runs the file as a subprocess. Information about the request such as the full URL, the originating host etc.,

is passed to the subprocess in the shell environment; additional input from the client may be read from standard input. Standard output from the subprocess is sent back across the network to the client as the response from the request. The CGI protocol describes what the environment variables passed to the subprocess mean and how the output should be formatted. The official reference documentation for the CGI protocol can be found on the World-Wide Web at <URL:http://hoohoo.ncsa.uiuc.edu/cgi/overview.html>. The cgi module was based on version 1.1 of the protocol and should also work with version 1.0.

The cgi module defines several classes that make it easy to access the information passed to the subprocess from a Python script; in particular, it knows how to parse the input sent by an HTML “form” using either a POST or a GET request (these are alternatives for submitting forms in the HTTP protocol).

The formatting of the output is so trivial that no additional support is needed. All you need to do is print a minimal set of MIME headers describing the output format, followed by a blank line and your actual output. E.g. if you want to generate HTML, your script could start as follows:

```
# Header -- one or more lines:
print "Content-type: text/html"
# Blank line separating header from body:
print
# Body, in HTML format:
print "<TITLE>The Amazing SPAM Homepage!</TITLE>"
# etc...
```

The server will add some header lines of its own, but it won’t touch the output following the header.

The cgi module defines the following functions:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

The module defines the following classes. Since the base class initializes itself by calling parse(), at most one instance of at most one of these classes should be created per script invocation:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

(It currently defines some more classes, but these are experimental and/or obsolescent, and are thus not documented — see the source for more informations.)

The module defines the following variable:

[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]

11.1.1 Example

CGI Example

This example assumes that you have a WWW server up and running, e.g. NCSA's httpd.

Place the following file in a convenient spot in the WWW server's directory tree. E.g., if you place it in the subdirectory test of the root directory and call it test.html, its URL will be http://yourservername/test/test.html.

```
<TITLE>Test Form Input</TITLE>
<H1>Test Form Input</H1>
<FORM METHOD="POST" ACTION="/cgi-bin/test.py">
<INPUT NAME=Name> (Name)<br>
<INPUT NAME=Address> (Address)<br>
<INPUT TYPE=SUBMIT>
</FORM>
```

Selecting this file's URL from a forms-capable browser such as Mosaic or Netscape will bring up a simple form with two text input fields and a "submit" button.

But wait. Before pressing "submit", a script that responds to the form must also be installed. The test file as shown assumes that the script is called test.py and lives in the server's cgi-bin directory. Here's the test script:

```
#!/usr/local/bin/python

import cgi

print "Content-type: text/html"
print                                               #
End of headers!
print "<TITLE>Test Form Output</TITLE>"
print "<H1>Test Form Output</H1>"

form = cgi.SvFormContentDict()              # Load
the form

name = addr = None                                  #
Default: no name and address

# Extract name and address from the form, if given

if form.has_key('Name'):
          name = form['Name']
```

```
if form.has_key('Address'):
          addr = form['Address']

# Print an unnumbered list of the name and address,
if present

print "<UL>"
if name is not None:
          print "<LI>Name:", cgi.escape(name)
if addr is not None:
          print "<LI>Address:", cgi.escape(addr)
print "</UL>"
```

The script should be made executable (chmod +x script). If the Python interpreter is not located at /usr/local/bin/python but somewhere else, the first line of the script should be modified accordingly.

Now that everything is installed correctly, we can try out the form. Bring up the test form in your WWW browser, fill in a name and address in the form, and press the “submit” button. The script should now run and its output is sent back to your browser. This should roughly look as follows:

Test Form Output

- Name: the name you entered
- Address: the address you entered

If you didn’t enter a name or address, the corresponding line will be missing (since the browser doesn’t send empty form fields to the server).

11.2 Standard Module urllib

urllib

This module provides a high-level interface for fetching data across the World-Wide Web. In particular, the urlopen function is similar to the built-in function open, but accepts URLs (Universal Resource Locators) instead of filenames. Some restrictions apply — it can only open URLs for reading, and no seek operations are available.

it defines the following public functions:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

Restrictions:

- Currently, only the following protocols are supported: HTTP, (versions 0.9 and 1.0), Gopher (but not Gopher-+), FTP, and local files.

- The caching feature of urlretrieve() has been disabled until I find the time to hack proper processing of Expiration time headers.
- There should be a function to query whether a particular URL is in the cache.
- For backward compatibility, if a URL appears to point to a local file but the file can't be opened, the URL is re-interpreted using the FTP protocol. This can sometimes cause confusing error messages.
- The urlopen() and urlretrieve() functions can cause arbitrarily long delays while waiting for a network connection to be set up. This means that it is difficult to build an interactive web client using these functions without using threads.
- The data returned by urlopen() or urlretrieve() is the raw data returned by the server. This may be binary data (e.g. an image), plain text or (for example) HTML. The HTTP protocol provides type information in the reply header, which can be inspected by looking at the Content-type header. For the Gopher protocol, type information is encoded in the URL; there is currently no easy way to extract it. If the returned data is HTML, you can use the module htmllib to parse it. htmllib
- Although the urllib module contains (undocumented) routines to parse and unparse URL strings, the recommended interface for URL manipulation is in module urlparse. urlparse

11.3 Standard Module httplib

httplib

This module defines a class which implements the client side of the HTTP protocol. It is normally not used directly — the module urllib uses it to handle URLs that use HTTP. urllib

The module defines one class, HTTP. An HTTP instance represents one transaction with an HTTP server. It should be instantiated passing it a host and optional port number. If no port number is passed, the port is extracted from the host string if it has the form host:port, else the default HTTP port (80) is used. If no host is passed, no connection is made, and the connect method should be used to connect to a server. For example, the following calls all create instances that connect to the server at the same host and port:

```
>>> h1 = httplib.HTTP('www.cwi.nl')
>>> h2 = httplib.HTTP('www.cwi.nl:80')
>>> h3 = httplib.HTTP('www.cwi.nl', 80)
```

Once an HTTP instance has been connected to an HTTP server, it should be used as follows:

1. Make exactly one call to the putrequest() method.
2. Make zero or more calls to the putheader() method.
3. Call the endheaders() method (this can be omitted if step 4 makes no calls).
4. Optional calls to the send() method.
5. Call the getreply() method.
6. Call the getfile() method and read the data off the file object that it returns.

11.3.1 HTTP Objects

HTTP instances have the following methods:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

11.3.2 Example

HTTP Example

Here is an example session:

```
>>> import httplib
>>> h = httplib.HTTP('www.cwi.nl')
>>> h.putrequest('GET', '/index.html')
>>> h.putheader('Accept', 'text/html')
>>> h.putheader('Accept', 'text/plain')
>>> h.endheaders()
>>> errcode, errmsg, headers = h.getreply()
>>> print errcode # Should be 200
>>> f = h.getfile()
>>> data f.read() # Get the raw HTML
>>> f.close()
>>>
```

11.4 Standard Module ftplib

ftplib

This module defines the class FTP and a few related items. The FTP class implements the client side of the FTP protocol. You can use this to write Python programs that perform a variety of automated FTP jobs, such as mirroring other ftp servers. It is also used by the module urllib to handle URLs that use FTP. For more information on FTP (File Transfer Protocol), see Internet RFC 959.

Here's a sample session using the ftplib module:

```
>>> from ftplib import FTP
>>> ftp = FTP('ftp.cwi.nl')    # connect to host,
default port
>>> ftp.login()                   # user anonymous,
passwd user@hostname
>>> ftp.retrlines('LIST')       # list directory
contents
total 24418
drwxrwsr-x    5 ftp-usr   pdmaint      1536 Mar 20
09:48 .
dr-xr-srwt 105 ftp-usr   pdmaint      1536 Mar 21
14:32 ..
-rw-r--r--    1 ftp-usr   pdmaint      5305 Mar 20
09:48 INDEX
 .
 .
 .
>>> ftp.quit()
```

The module defines the following items:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]

[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]

[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]

[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]

[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]

11.4.1 FTP Objects

FTP instances have the following methods:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

11.5 Standard Module gopherlib

gopherlib

This module provides a minimal implementation of client side of the the Gopher protocol. It is used by the module urllib to handle URLs that use the Gopher protocol.

The module defines the following functions:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

Note that the data returned by the Gopher server can be of any type, depending on the first character of the selector string. If the data is text (first character of the selector is 0), lines are terminated by CRLF, and the data is terminated by a line consisting of a single ., and a leading . should be stripped from lines that begin with ... Directory listings (first charactger of the selector is 1) are transferred using the same protocol.

11.6 Standard Module nntplib

nntplib

This module defines the class NNTP which implements the client side of the NNTP protocol. It can be used to implement a news reader or poster, or automated news processors. For more information on NNTP (Network News Transfer Protocol), see Internet RFC 977.

Here are two small examples of how it can be used. To list some statistics about a newsgroup and print the subjects of the last 10 articles:

```
>>> s = NNTP('news.cwi.nl')
```

```
>>> resp, count, first, last, name =
s.group('comp.lang.python')
>>> print 'Group', name, 'has', count, 'articles, range',
first, 'to', last
Group comp.lang.python has 59 articles, range 3742 to
3803
>>> resp, subs = s.xhdr('subject', first + '-' + last)
>>> for id, sub in subs[-10:]: print id, sub
...
3792 Re: Removing elements from a list while iterating...
3793 Re: Who likes Info files?
3794 Emacs and doc strings
3795 a few questions about the Mac implementation
3796 Re: executable python scripts
3797 Re: executable python scripts
3798 Re: a few questions about the Mac implementation
3799 Re: PROPOSAL: A Generic Python Object Interface for
Python C Modules
3802 Re: executable python scripts
3803 Re: POSIX wait and SIGCHLD
>>> s.quit()
'205 news.cwi.nl closing connection.  Goodbye.'
>>>
```

To post an article from a file (this assumes that the article has valid headers):

```
>>> s = NNTP('news.cwi.nl')
>>> f = open('/tmp/article')
>>> s.post(f)
'240 Article posted successfully.'
>>> s.quit()
'205 news.cwi.nl closing connection.  Goodbye.'
>>>
```

The module itself defines the following items:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]

[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]

[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]

[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]

11.6.1 NNTP Objects

NNTP instances have the following methods. The response that is returned as the first item in the return tuple of almost all methods is the server's response: a string beginning with a three-digit code. If the server's response indicates an error, the method raises one of the above exceptions.

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

11.7 Standard Module urlparse

urlparse URLparsing relativeURL

This module defines a standard interface to break URL strings up in components (addessing scheme, network location, path etc.), to combine the components back into a URL string, and to convert a “relative URL” to an absolute URL given a “base URL”.

The module has been designed to match the current Internet draft on Relative Uniform Resource Locators (and discovered a bug in an earlier draft!).

It defines the following functions:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

11.8 Standard Module htmllib

htmllib

This module defines a number of classes which can serve as a basis for parsing text files formatted in HTML (HyperText Mark-up Language). The classes are not directly concerned with I/O — the have to be fed their input in string form, and will make calls to methods of a “formatter” object in order to produce output. The classes are designed to be used as base classes for other classes in order to add functionality, and allow most of their methods to be extended or overridden. In turn, the classes are derived from and extend the class SGMLParser defined in module sgmllib. sgmllib SGMLParser

The following is a summary of the interface defined by sgmllib.SGMLParser:

- The interface to feed data to an instance is through the feed() method, which takes a string argument. This can be called with as little or as much text at a time as desired; p.feed(a); p.feed(b) has the same effect as p.feed(a+b). When the data contains complete HTML elements, these are processed immediately; incomplete elements are saved in a buffer. To force processing of all unprocessed data, call the close() method.

  Example: to parse the entire contents of a file, do parser.feed(open(file).read()); parser.close().
- The interface to define semantics for HTML tags is very simple: derive a class and define methods called start\s\do5()tag(), end\s\do5()tag(), or do\s\do5()tag(). The parser will call these at appropriate moments: start\s\do5()tag or do\s\do5()tag is called when an opening tag of the form <tag ...> is encountered; end\s\do5()tag is called when a closing tag of the form <tag> is encountered. If an opening tag requires a corresponding closing tag, like <H1> ... </H1>, the class should define the start\s\do5()tag method; if a tag requires no closing tag, like <P>, the class should define the do\s\do5()tag method.

The module defines the following classes:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

Instances of CollectingParser (and thus also instances of FormattingParser and AnchoringParser) have the following instance variables:

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

The anchors, anchornames and anchortypes lists are “parallel arrays”: items in these lists with the same index pertain to the same anchor. Missing attributes default to the empty string. Anchors with neither a HREF nor a NAME attribute are not entered in these lists at all.

The module also defines a number of style sheet classes. These should never be instantiated — their class variables are the only behavior required. Note that style sheets are specifically designed for a particular formatter implementation. The currently defined style sheets are:

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]

Style sheets have the following class variables:

[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]

Although no documented implementation of a formatter exists, the FormattingParser class assumes that formatters have a certain interface. This interface requires the following methods:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

A sample formatter implementation can be found in the module fmt, which in turn uses the module Para. These modules are not intended as standard library modules; they are available as an example of how to write a formatter. fmt Para

11.9 Standard Module sgmllib

sgmllib

This module defines a class SGMLParser which serves as the basis for parsing text files formatted in SGML (Standard Generalized Mark-up Language). In fact, it does not provide a full SGML parser — it only parses SGML insofar as it is used by HTML, and the module only exists as a basis for the htmllib module. htmllib

In particular, the parser is hardcoded to recognize the following elements:

- Opening and closing tags of the form “<tag attr="value" ...>” and “</tag>”, respectively.
- Character references of the form “&#name;”.

- Entity references of the form “&name;”.
- SGML comments of the form “<!–text>”.

The SGMLParser class must be instantiated without arguments. It has the following interface methods:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

Apart from overriding or extending the methods listed above, derived classes may also define methods of the following form to define processing of specific tags. Tag names in the input stream are case independent; the tag occurring in method names must be in lower case:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

Note that the parser maintains a stack of opening tags for which no matching closing tag has been found yet. Only tags processed by start\s\do5()tag() are pushed on this stack. Definition of a end\s\do5()tag() method is optional for these tags. For tags processed by do\s\do5()tag() or by unknown\s\do5(t)ag(), no end\s\do5()tag() method must be defined.

11.10 Standard Module rfc822

rfc822

This module defines a class, Message, which represents a collection of “email headers” as defined by the Internet standard RFC 822. It is used in various contexts, usually to read such headers from a file.

A Message instance is instantiated with an open file object as parameter. Instantiation reads headers from the file up to a blank line and stores them in the instance; after instantiation, the file is positioned directly after the blank line that terminates the headers.

Input lines as read from the file may either be terminated by CR-LF or by a single linefeed; a terminating CR-LF is replaced by a single linefeed before the line is stored.

All header matching is done independent of upper or lower case; e.g. m['From'], m['from'] and m['FROM'] all yield the same result.

11.10.1 Message Objects

A Message instance has the following methods:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

Message instances also support a read-only mapping interface. In particular: m[name] is the same as m.getheader(name); and len(m), m.has\s\do5(k)ey(name), m.keys(), m.values() and m.items() act as expected (and consistently).

Finally, Message instances have two public instance variables:

[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]

11.11 Standard Module mimetools

mimetools

This module defines a subclass of the class rfc822.Message and a number of utility functions that are useful for the manipulation for MIME style multipart or encoded message.

It defines the following items:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

11.11.1 Additional Methods of Message objects

mimetools.Message Methods

The mimetools.Message class defines the following methods in addition to the rfc822.Message class:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

11.12 Standard module binhex

binhex

This module encodes and decodes files in binhex4 format, a format allowing representation of Macintosh files in ASCII. On the macintosh, both forks of a file and the finder information are encoded (or decoded), on other platforms only the data fork is handled.

The binhex module defines the following functions:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

11.12.1 notes

There is an alternative, more powerful interface to the coder and decoder, see the source for details.

If you code or decode textfiles on non-Macintosh platforms they will still use the macintosh newline convention (carriage-return as end of line).

As of this writing, hexbin appears to not work in all cases.

11.13 Standard module uu

uu

This module encodes and decodes files in uuencode format, allowing arbitrary binary data to be transferred over ascii-only connections. Whereever a file argument is expected, the methods accept either a pathname ('-' for stdin/stdout) or a file-like object.

Normally you would pass filenames, but there is one case where you have to open the file yourself: if you are on a non-unix platform and your binary file is actually a textfile that you want encoded unix-compatible you will have to open the file yourself as a textfile, so newline conversion is performed.

This code was contributed by Lance Ellinghouse, and modified by Jack Jansen.

The uu module defines the following functions:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

11.14 Built-in Module binascii

binascii

The binascii module contains a number of methods to convert between binary and various ascii-encoded binary representations. Normally, you will

not use these modules directly but use wrapper modules like uu or hexbin in stead, this module solely exists because bit-manipuation of large amounts of data is slow in python.

The binascii module defines the following functions:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]
[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]

Chapter 12

Multimedia Services

The modules described in this chapter implement various algorithms or interfaces that are mainly useful for multimedia applications. They are available at the discretion of the installation. Here's an overview:

audioop — Manipulate raw audio data.

imageop — Manipulate raw image data.

aifc — Read and write audio files in AIFF or AIFC format.

jpeg — Read and write image files in compressed JPEG format.

rgbimg — Read and write image files in "SGI RGB" format (the module is *not* SGI specific though)!

12.1 Built-in Module audioop

audioop

The audioop module contains some useful operations on sound fragments. It operates on sound fragments consisting of signed integer samples 8, 16 or 32 bits wide, stored in Python strings. This is the same format as used by the al and sunaudiodev modules. All scalar items are integers, unless specified otherwise.

A few of the more complicated operations only take 16-bit samples, otherwise the sample size (in bytes) is always a parameter of the operation.

The module defines the following variables and functions:

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

Note that operations such as mul or max make no distinction between mono and stereo fragments, i.e. all samples are treated equal. If this is a problem the stereo fragment should be split into two mono fragments first and recombined later. Here is an example of how to do that:

```
def mul_stereo(sample, width, lfactor, rfactor):
     lsample = audioop.tomono(sample, width, 1, 0)
     rsample = audioop.tomono(sample, width, 0, 1)
     lsample = audioop.mul(sample, width, lfactor)
     rsample = audioop.mul(sample, width, rfactor)
     lsample = audioop.tostereo(lsample, width, 1,
0)
     rsample = audioop.tostereo(rsample, width, 0,
1)
     return audioop.add(lsample, rsample, width)
```

If you use the ADPCM coder to build network packets and you want your protocol to be stateless (i.e. to be able to tolerate packet loss) you should not only transmit the data but also the state. Note that you should send the initial state (the one you passed to lin2adpcm) along to the decoder, not the final state (as returned by the coder). If you want to use struct to store the state in binary you can code the first element (the predicted value) in 16 bits and the second (the delta index) in 8.

The ADPCM coders have never been tried against other ADPCM coders, only against themselves. It could well be that I misinterpreted the standards in which case they will not be interoperable with the respective standards.

The find... routines might look a bit funny at first sight. They are primarily meant to do echo cancellation. A reasonably fast way to do this is to pick the most energetic piece of the output sample, locate that in the input sample and subtract the whole output sample from the input sample:

```
def echocancel(outputdata, inputdata):
     pos = audioop.findmax(outputdata, 800)      #
one tenth second
     out_test = outputdata[pos*2:]
     in_test = inputdata[pos*2:]
     ipos, factor = audioop.findfit(in_test,
out_test)
     # Optional (for better cancellation):
     # factor =
audioop.findfactor(in_test[ipos*2:ipos*2+len(out_tes
t)],
     #                  out_test)
     prefill = '\0'*(pos+ipos)*2
     postfill = '\0'*(len(inputdata)-len(prefill)-
len(outputdata))
     outputdata = prefill +
audioop.mul(outputdata,2,-factor) + postfill
     return audioop.add(inputdata, outputdata, 2)
```

12.2 Built-in Module imageop

imageop

The imageop module contains some useful operations on images. It operates on images consisting of 8 or 32 bit pixels stored in Python strings. This is the same format as used by gl.lrectwrite and the imgfile module.

The module defines the following variables and functions:

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

12.3 Standard Module aifc

aifc

This module provides support for reading and writing AIFF and AIFF-C files. AIFF is Audio Interchange File Format, a format for storing digital audio samples in a file. AIFF-C is a newer version of the format that includes the ability to compress the audio data.

Audio files have a number of parameters that describe the audio data. The sampling rate or frame rate is the number of times per second the sound is sampled. The number of channels indicate if the audio is mono, stereo, or quadro. Each frame consists of one sample per channel. The sample size is the size in bytes of each sample. Thus a frame consists of nchannels*samplesize bytes, and a second's worth of audio consists of nchannels*samplesize*framerate bytes.

For example, CD quality audio has a sample size of two bytes (16 bits), uses two channels (stereo) and has a frame rate of 44,100 frames/second. This gives a frame size of 4 bytes (2*2), and a second's worth occupies 2*2*44100 bytes, i.e. 176,400 bytes.

Module aifc defines the following function:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

Objects returned by aifc.open() when a file is opened for reading have the following methods:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

Objects returned by aifc.open() when a file is opened for writing have all the above methods, except for readframes and setpos. In addition the following methods exist. The get methods can only be called after the corresponding set methods have been called. Before the first writeframes or writeframesraw, all parameters except for the number of frames must be filled in.

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

12.4 Built-in Module jpeg

jpeg

The module jpeg provides access to the jpeg compressor and decompressor written by the Independent JPEG Group. JPEG is a (draft?) standard for compressing pictures. For details on jpeg or the Independent JPEG Group software refer to the JPEG standard or the documentation provided with the software.

The jpeg module defines these functions:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

Compress and uncompress raise the error jpeg.error in case of errors.

12.5 Built-in Module rgbimg

rgbimg

The rgbimg module allows python programs to access SGI imglib image files (also known as .rgb files). The module is far from complete, but is provided anyway since the functionality that there is is enough in some cases. Currently, colormap files are not supported.

The module defines the following variables and functions:

[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

Chapter 13

Cryptographic Services

The modules described in this chapter implement various algorithms of a cryptographic nature. They are available at the discretion of the installation. Here's an overview:

md5 — RSA's MD5 message digest algorithm.

mpz — Interface to the GNU MP library for arbitrary precision arithmetic.

rotor — Enigma-like encryption and decryption.

Hardcore cypherpunks will probably find the Python Cryptography Kit of further interest; the package adds built-in modules for DES and IDEA encryption, and provides a Python module for reading and decrypting PGP files. The Python Cryptography Kit is not distributed with Python but available separately. See the URL http://www.cs.mcgill.ca/%7Efnord/crypt.html for more information. DEScipher IDEAcipher

13.1 Built-in Module md5

md5

This module implements the interface to RSA's MD5 message digest algorithm (see also Internet RFC 1321). Its use is quite straightforward: use the md5.new() to create an md5 object. You can now feed this object with arbitrary strings using the update() method, and at any point you can ask it for the digest (a strong kind of 128-bit checksum, a.k.a. "fingerprint") of the contatenation of the strings fed to it so far using the digest() method.

For example, to obtain the digest of the string `"Nobody inspects the spammish repetition"`:

```
>>> import md5
>>> m = md5.new()
>>> m.update("Nobody inspects")
>>> m.update(" the spammish repetition")
>>> m.digest()
'\273d\234\203\335\036\245\311\331\336\311\241\215\3
60\377\351'
```

More condensed:

```
>>> md5.new("Nobody inspects the spammish
repetition").digest()
'\273d\234\203\335\036\245\311\331\336\311\241\215\3
60\377\351'
```

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

An md5 object has the following methods:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

13.2 Built-in Module mpz

mpz

This is an optional module. It is only available when Python is configured to include it, which requires that the GNU MP software is installed.

This module implements the interface to part of the GNU MP library, which defines arbitrary precision integer and rational number arithmetic routines. Only the interfaces to the *integer* (mpz\s\do5(...)) routines are provided. If not stated otherwise, the description in the GNU MP documentation can be applied.

In general, mpz-numbers can be used just like other standard Python numbers, e.g. you can use the built-in operators like +, *, etc., as well as the standard built-in functions like abs, int, …, divmod, pow. Please note: the *bitwise-xor* operation has been implemented as a bunch of *and*s, *invert*s and *or*s, because the library lacks an mpz\s\do5(x)or function, and I didn't need one.

You create an mpz-number by calling the function called mpz (see below for an exact description). An mpz-number is printed like this: mpz(value).

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

A number of *extra* functions are defined in this module. Non mpz-arguments are converted to mpz-values first, and the functions return mpz-numbers.

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

An mpz-number has one method:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

13.3 Built-in Module rotor

rotor

This module implements a rotor-based encryption algorithm, contributed by Lance Ellinghouse. The design is derived from the Enigma device, a machine used during World War II to encipher messages. A rotor is simply a permutation. For example, if the character 'A' is the origin of the rotor, then a given rotor might map 'A' to 'L', 'B' to 'Z', 'C' to 'G', and so on. To encrypt, we choose several different rotors, and set the origins of the rotors to known positions; their initial position is the ciphering key. To encipher a character, we permute the original character by the first rotor, and then apply the second rotor's permutation to the result. We continue until we've applied all the rotors; the resulting character is our ciphertext. We then change the origin of the final rotor by one position, from 'A' to 'B'; if the final rotor has made a complete revolution, then we rotate the next-to-last rotor by one position, and apply the same procedure recursively. In other words, after enciphering one character, we advance the rotors in the same fashion as a car's odometer. Decoding works in the same way, except we reverse the permutations and apply them in the opposite order. Enigmacipher

The available functions in this module are:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

Rotor objects have the following methods:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

An example usage:

```
>>> import rotor
>>> rt = rotor.newrotor('key', 12)
>>> rt.encrypt('bar')
'\2534\363'
>>> rt.encryptmore('bar')
'\357\375$'
>>> rt.encrypt('bar')
'\2534\363'
>>> rt.decrypt('\2534\363')
'bar'
>>> rt.decryptmore('\357\375$')
'bar'
>>> rt.decrypt('\357\375$')
'l(\315'
```

```
>>> del rt
```

The module's code is not an exact simulation of the original Enigma device; it implements the rotor encryption scheme differently from the original. The most important difference is that in the original Enigma, there were only 5 or 6 different rotors in existence, and they were applied twice to each character; the cipher key was the order in which they were placed in the machine. The Python rotor module uses the supplied key to initialize a random number generator; the rotor permutations and their initial positions are then randomly generated. The original device only enciphered the letters of the alphabet, while this module can handle any 8-bit binary data; it also produces binary output. This module can also operate with an arbitrary number of rotors.

The original Enigma cipher was broken in 1944. The version implemented here is probably a good deal more difficult to crack (especially if you use many rotors), but it won't be impossible for a truly skilful and determined attacker to break the cipher. So if you want to keep the NSA out of your files, this rotor cipher may well be unsafe, but for discouraging casual snooping through your files, it will probably be just fine, and may be somewhat safer than using the Unix crypt command.

Chapter 14

Macintosh Specific Services

The modules in this chapter are available on the Apple Macintosh only.

14.1 Built-in Module mac

mac This module provides a subset of the operating system dependent functionality provided by the optional built-in module posix. It is best accessed through the more portable standard module os.

The following functions are available in this module: chdir, getcwd, listdir, mkdir, rename, rmdir, stat, sync, unlink, as well as the exception error.

14.2 Standard Module macpath

macpath This module provides a subset of the pathname manipulation functions available from the optional standard module posixpath. It is best accessed through the more portable standard module os, as os.path.

The following functions are available in this module: normcase, isabs, join, split, isdir, isfile, exists.

14.3 Built-in Module ctb

ctb

This module provides a partial interface to the Macintosh Communications Toolbox. Currently, only Connection Manager tools are supported. It may not be available in all Mac Python versions.

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

14.3.1 connection object

For all connection methods that take a timeout argument, a value of -1 is indefinite, meaning that the command runs to completion.

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

14.4 Built-in Module macconsole

macconsole

This module is available on the Macintosh, provided Python has been built using the Think C compiler. It provides an interface to the Think console package, with which basic text windows can be created.

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

14.4.1 macconsole options object

These options are examined when a window is created:

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

14.4.2 console window object

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

14.5 Built-in Module macdnr

macdnr

This module provides an interface to the Macintosh Domain Name Resolver. It is usually used in conjunction with the mactcp module, to map hostnames to IP-addresses. It may not be available in all Mac Python versions.

The macdnr module defines the following functions:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

14.5.1 dnr result object

Since the DNR calls all execute asynchronously you do not get the results back immediately. Instead, you get a dnr result object. You can check this object to see whether the query is complete, and access its attributes to obtain the information when it is.

Alternatively, you can also reference the result attributes directly, this will result in an implicit wait for the query to complete.

The rtnCode and cname attributes are always available, the others depend on the type of query (address, hinfo or mx).

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]

The simplest way to use the module to convert names to dotted-decimal strings, without worrying about idle time, etc:

```
>>> def gethostname(name):
...       import macdnr
```

```
...        dnrr = macdnr.StrToAddr(name)
...        return macdnr.AddrToStr(dnrr.ip0)
```

14.6 Built-in Module macfs

macfs

This module provides access to macintosh FSSpec handling, the Alias Manager, finder aliases and the Standard File package.

Whenever a function or method expects a file argument, this argument can be one of three things: (1) a full or partial Macintosh pathname, (2) an FSSpec object or (3) a 3-tuple (wdRefNum, parID, name) as described in Inside Mac VI. A description of aliases and the standard file package can also be found there.

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

14.6.1 FSSpec objects

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

14.6.2 alias objects

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

Note that it is currently not possible to directly manipulate a resource as an alias object. Hence, after calling Update or after Resolve indicates that the alias has changed the Python program is responsible for getting the data from the alias object and modifying the resource.

14.6.3 FInfo objects

See Inside Mac for a complete description of what the various fields mean.

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

14.7 Built-in Module MacOS

MacOS

This module provides access to MacOS specific functionality in the python interpreter, such as how the interpreter eventloop functions and the like. Use with care.

Note the capitalisation of the module name, this is a historical artefact.

[Sorry. Ignored `\begin{excdesc} ... \end{excdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

14.8 Standard module macostools

macostools

This module contains some convenience routines for file-manipulation on the Macintosh.

The macostools module defines the following functions:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

Note that the process of creating finder aliases is not specified in the Apple documentation. Hence, aliases created with mkalias could conceivably have incompatible behaviour in some cases.

14.9 Built-in Module mactcp

mactcp

This module provides an interface to the Macintosh TCP/IP driver MacTCP. There is an accompanying module macdnr which provides an interface to the name-server (allowing you to translate hostnames to ip-addresses), a module MACTCP which has symbolic names for constants constants used by MacTCP and a wrapper module socket which mimics the socket interface (as far as possible). It may not be available in all Mac Python versions.

A complete description of the MacTCP interface can be found in the Apple MacTCP API documentation.

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

14.9.1 TCP Stream Objects

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

14.9.2 TCP Status Objects

This object has no methods, only some members holding information on the connection. A complete description of all fields in this objects can be found in the Apple documentation. The most interesting ones are:

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]
[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

14.9.3 UDP Stream Objects

Note that, unlike the name suggests, there is nothing stream-like about UDP.

[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

14.10 Built-in Module macspeech

macspeech

This module provides an interface to the Macintosh Speech Manager, allowing you to let the Macintosh utter phrases. You need a version of the speech manager extension (version 1 and 2 have been tested) in your Extensions folder for this to work. The module does not provide full access to all features of the Speech Manager yet. It may not be available in all Mac Python versions.

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

14.10.1 voice objects

Voice objects contain the description of a voice. It is currently not yet possible to access the parameters of a voice.

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

14.10.2 speech channel objects

A speech channel object allows you to speak strings with slightly more control than SpeakString(), and allows you to use multiple speakers at the same time. Please note that channel pitch and rate are interrelated in some way, so that to make your Macintosh sing you will have to adjust both.

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

14.11 Standard module EasyDialogs

EasyDialogs

The EasyDialogs module contains some simple dialogs for the Macintosh, modelled after the stdwin dialogs with similar names.

The EasyDialogs module defines the following functions:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

Note that EasyDialogs does not currently use the notification manager. This means that displaying dialogs while the program is in the background will need to unexpected results and possibly crashes.

14.12 Standard module FrameWork

FrameWork

The FrameWork module contains classes that together provide a framework for an interactive Macintosh application. The programmer builds an application by creating subclasses that override various methods of the bases classes, thereby implementing the functionality wanted. Overriding functionality can often be done on various different levels, i.e. to handle clicks in a single dialog window in a non-standard way it is not necessary to override the complete event handling.

The FrameWork is still very much work-in-progress, and the documentation describes only the most important functionality, and not in the most logical manner at that. Examine the source for more esoteric needs.

The EasyDialogs module defines the following functions:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

14.12.1 Application objects

Application objects have the following methods, among others:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

14.12.2 Window Objects

Window objects have the following methods, among others:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

14.12.3 DialogWindow Objects

DialogWindow objects have the following methods besides those of Window objects:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

Chapter 15

Standard Windowing Interface

The modules in this chapter are available only on those systems where the STDWIN library is available. STDWIN runs on under X11 and on the Macintosh. See CWI report CS-R8817.

Warning: Using STDWIN is not recommended for new applications. It has never been ported to Microsoft Windows or Windows NT, and for X11 or the Macintosh it lacks important functionality — in particular, it has no tools for the construction of dialogs. For most platforms, alternative, native solutions exist (though none are currently documented in this manual): Tkinter for under X11, native Xt with Motif or Athena widgets for under X11, Win32 for Windows and Windows NT, and a collection of native toolkit interfaces for the Macintosh.

15.1 Built-in Module stdwin

stdwin

This module defines several new object types and functions that provide access to the functionality of STDWIN.

On Unix running X11, it can only be used if the DISPLAY environment variable is set or an explicit -display displayname argument is passed to the Python interpreter.

Functions have names that usually resemble their C STDWIN counterparts with the initial 'w' dropped. Points are represented by pairs of integers; rectangles by pairs of points. For a complete description of STDWIN please refer to the documentation of STDWIN for C programmers (aforementioned CWI report).

15.1.1 Functions Defined in Module stdwin

STDWIN Functions

The following functions are defined in the stdwin module:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

15.1.2 Window Objects

Window objects are created by stdwin.open(). They are closed by their close() method or when they are garbage-collected. Window objects have the following methods:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

15.1.3 Drawing Objects

Drawing objects are created exclusively by the window method begindrawing(). Only one drawing object can exist at any given time; the drawing object must be deleted to finish drawing. No drawing object may exist when stdwin.getevent() is called. Drawing objects have the following methods:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

15.1.4 Menu Objects

A menu object represents a menu. The menu is destroyed when the menu object is deleted. The following methods are defined:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

15.1.5 Bitmap Objects

A bitmap represents a rectangular array of bits. The top left bit has coordinate (0, 0). A bitmap can be drawn with the bitmap method of a drawing object. Bitmaps are currently not available on the Macintosh.

The following methods are defined:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

15.1.6 Text-edit Objects

A text-edit object represents a text-edit block. For semantics, see the STDWIN documentation for C programmers. The following methods exist:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

15.1.7 Example

STDWIN Example

Here is a minimal example of using STDWIN in Python. It creates a window and draws the string “Hello world” in the top left corner of the window. The window will be correctly redrawn when covered and re-exposed. The program quits when the close icon or menu item is requested.

```
import stdwin
from stdwinevents import *

def main():
     mywin = stdwin.open('Hello')
     #
     while 1:
          (type, win, detail) = stdwin.getevent()
          if type == WE_DRAW:
               draw = win.begindrawing()
               draw.text((0, 0), 'Hello, world')
               del draw
          elif type == WE_CLOSE:
               break

main()
```

15.2 Standard Module stdwinevents

stdwinevents

This module defines constants used by STDWIN for event types (WE\s\do5(A)CTIVATE etc.), command codes (WC\s\do5(L)EFT etc.) and selection types (WS\s\do5(P)RIMARY etc.). Read the file for details. Suggested usage is

```
>>> from stdwinevents import *
>>>
```

15.3 Standard Module rect

rect

This module contains useful operations on rectangles. A rectangle is defined as in module stdwin: a pair of points, where a point is a pair of integers. For example, the rectangle

```
(10, 20), (90, 80)
```

is a rectangle whose left, top, right and bottom edges are 10, 20, 90 and 80, respectively. Note that the positive vertical axis points down (as in stdwin).

The module defines the following objects:

[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]

[Sorry. Ignored \begin{datadesc} ... \end{datadesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

Chapter 16

SGI IRIX Specific Services

The modules described in this chapter provide interfaces to features that are unique to SGI's IRIX operating system (versions 4 and 5).

16.1 Built-in Module al

al

This module provides access to the audio facilities of the SGI Indy and Indigo workstations. See section 3A of the IRIX man pages for details. You'll need to read those man pages to understand what these functions do! Some of the functions are not available in IRIX releases before 4.0.5. Again, see the manual to check whether a specific function is available on your platform.

All functions and methods defined in this module are equivalent to the C functions with AL prefixed to their name.

Symbolic constants from the C header file <audio.h> are defined in the standard module AL, see below.

Warning: the current version of the audio library may dump core when bad argument values are passed rather than returning an error status. Unfortunately, since the precise circumstances under which this may happen are undocumented and hard to check, the Python interface can provide no protection against this kind of problems. (One example is specifying an excessive queue size — there is no documented upper limit.)

The module defines the following functions:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

16.1.1 Configuration Objects

Configuration objects (returned by al.newconfig() have the following methods:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

16.1.2 Port Objects

Port objects (returned by al.openport() have the following methods:
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

16.2 Standard Module AL

AL (uppercase) AL

This module defines symbolic constants needed to use the built-in module al (see above); they are equivalent to those defined in the C header file <audio.h> except that the name prefix AL\s\do5()is omitted. Read the module source for a complete list of the defined names. Suggested use:

```
import al
from AL import *
```

16.3 Built-in Module cd

cd

This module provides an interface to the Silicon Graphics CD library. It is available only on Silicon Graphics systems.

The way the library works is as follows. A program opens the CD-ROM device with cd.open() and creates a parser to parse the data from the CD with cd.createparser(). The object returned by cd.open() can be used to read data from the CD, but also to get status information for the CD-ROM device, and to get information about the CD, such as the table of contents. Data from the

CD is passed to the parser, which parses the frames, and calls any callback functions that have previously been added.

An audio CD is divided into tracks or programs (the terms are used interchangeably). Tracks can be subdivided into indices. An audio CD contains a table of contents which gives the starts of the tracks on the CD. Index 0 is usually the pause before the start of a track. The start of the track as given by the table of contents is normally the start of index 1.

Positions on a CD can be represented in two ways. Either a frame number or a tuple of three values, minutes, seconds and frames. Most functions use the latter representation. Positions can be both relative to the beginning of the CD, and to the beginning of the track.

Module cd defines the following functions and constants:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

The module defines the following variables:

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

The following variables are states as returned by getstatus:

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

[Sorry. Ignored `\begin{datadesc} ... \end{datadesc}`]

Player objects (returned by cd.open()) have the following methods:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

Parser objects (returned by cd.createparser()) have the following methods:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

16.4 Built-in Module fl

fl

This module provides an interface to the FORMS Library by Mark Overmars. The source for the library can be retrieved by anonymous ftp from host ftp.cs.ruu.nl, directory SGI/FORMS. It was last tested with version 2.0b.

Most functions are literal translations of their C equivalents, dropping the initial fl\s\do5()from their name. Constants used by the library are defined in module FL described below.

The creation of objects is a little different in Python than in C: instead of the 'current form' maintained by the library to which new FORMS objects are added, all functions that add a FORMS object to a form are methods of the Python object representing the form. Consequently, there are no Python equivalents for the C functions fl\s\do5(a)ddto\s\do5(f)orm and fl\s\do5(e)nd\s\do5(f)orm, and the equivalent of fl\s\do5(b)gn\s\do5(f)orm is called fl.make\s\do5(f)orm.

Watch out for the somewhat confusing terminology: FORMS uses the word object for the buttons, sliders etc. that you can place in a form. In Python, 'object' means any value. The Python interface to FORMS introduces two new Python object types: form objects (representing an entire form) and FORMS objects (representing one button, slider etc.). Hopefully this isn't too confusing...

There are no 'free objects' in the Python interface to FORMS, nor is there an easy way to add object classes written in Python. The FORMS interface to GL event handling is available, though, so you can mix FORMS with pure GL windows.

Please note: importing fl implies a call to the GL function foreground() and to the FORMS routine fl\s\do5(i)nit().

16.4.1 Functions Defined in Module fl

FL Functions

Module fl defines the following functions. For more information about what they do, see the description of the equivalent C function in the FORMS documentation:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

16.4.2 Form Objects

Form objects (returned by fl.make\s\do5(f)orm() above) have the following methods. Each method corresponds to a C function whose name is prefixed with fl\s\do5(); and whose first argument is a form pointer; please refer to the official FORMS documentation for descriptions.

All the add\s\do5(...) functions return a Python object representing the FORMS object. Methods of FORMS objects are described below. Most kinds of FORMS object also have some methods specific to that kind; these methods are listed here.

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

Form objects have the following data attributes; see the FORMS documentation:

[Sorry. Ignored `\begin{tableiii} ... \end{tableiii}`]

16.4.3 FORMS Objects

Besides methods specific to particular kinds of FORMS objects, all FORMS objects also have the following methods:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

FORMS objects have these data attributes; see the FORMS documentation:

[Sorry. Ignored `\begin{tableiii} ... \end{tableiii}`]

16.5 Standard Module FL

FL (uppercase) FL

This module defines symbolic constants needed to use the built-in module fl (see above); they are equivalent to those defined in the C header file <forms.h> except that the name prefix FL\s\do5()is omitted. Read the module source for a complete list of the defined names. Suggested use:

```
import fl
from FL import *
```

16.6 Standard Module flp

flp

This module defines functions that can read form definitions created by the 'form designer' (fdesign) program that comes with the FORMS library (see module fl above).

For now, see the file flp.doc in the Python library source directory for a description.

XXX A complete description should be inserted here!

16.7 Built-in Module fm

fm

This module provides access to the IRIS *Font Manager* library. It is available only on Silicon Graphics machines. See also: 4Sight User's Guide, Section 1, Chapter 5: Using the IRIS Font Manager.

This is not yet a full interface to the IRIS Font Manager. Among the unsupported features are: matrix operations; cache operations; character operations (use string operations instead); some details of font info; individual glyph metrics; and printer matching.

It supports the following operations:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

Font handle objects support the following operations:

[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]
[Sorry. Ignored `\begin{funcdesc} ... \end{funcdesc}`]

16.8 Built-in Module gl

gl

This module provides access to the Silicon Graphics *Graphics Library*. It is available only on Silicon Graphics machines.

Warning: Some illegal calls to the GL library cause the Python interpreter to dump core. In particular, the use of most GL calls is unsafe before the first window is opened.

The module is too large to document here in its entirety, but the following should help you to get started. The parameter conventions for the C functions are translated to Python as follows:

- All (short, long, unsigned) int values are represented by Python integers.
- All float and double values are represented by Python floating point numbers. In most cases, Python integers are also allowed.
- All arrays are represented by one-dimensional Python lists. In most cases, tuples are also allowed.
- All string and character arguments are represented by Python strings, for instance, winopen('Hi There!') and rotate(900, 'z').
- All (short, long, unsigned) integer arguments or return values that are only used to specify the length of an array argument are omitted. For example, the C call

  ```
  lmdef(deftype, index, np, props)
  ```

  is translated to Python as

  ```
  lmdef(deftype, index, props)
  ```

- Output arguments are omitted from the argument list; they are transmitted as function return values instead. If more than one value must be returned, the return value is a tuple. If the C function has both a regular return value (that is not omitted because of the previous rule) and an output argument, the return value comes first in the tuple. Examples: the C call

  ```
  getmcolor(i, &red, &green, &blue)
  ```

  is translated to Python as

  ```
  red, green, blue = getmcolor(i)
  ```

The following functions are non-standard or have special argument conventions:

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

Here is a tiny but complete example GL program in Python:

```
import gl, GL, time

def main():
     gl.foreground()
     gl.prefposition(500, 900, 500, 900)
     w = gl.winopen('CrissCross')
     gl.ortho2(0.0, 400.0, 0.0, 400.0)
     gl.color(GL.WHITE)
     gl.clear()
     gl.color(GL.RED)
     gl.bgnline()
     gl.v2f(0.0, 0.0)
     gl.v2f(400.0, 400.0)
     gl.endline()
     gl.bgnline()
     gl.v2f(400.0, 0.0)
     gl.v2f(0.0, 400.0)
     gl.endline()
     time.sleep(5)

main()
```

16.9 Standard Modules GL and DEVICE

GL and DEVICE GL DEVICE

These modules define the constants used by the Silicon Graphics *Graphics Library* that C programmers find in the header files <gl/gl.h> and <gl/device.h>. Read the module source files for details.

16.10 Built-in Module imgfile

imgfile

The imgfile module allows python programs to access SGI imglib image files (also known as .rgb files). The module is far from complete, but is provided anyway since the functionality that there is is enough in some cases. Currently, colormap files are not supported.

The module defines the following variables and functions:

[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]
[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

Chapter 17

SunOS Specific Services

The modules described in this chapter provide interfaces to features that are unique to the SunOS operating system (versions 4 and 5; the latter is also known as Solaris version 2).

17.1 Built-in Module sunaudiodev

sunaudiodev

This module allows you to access the sun audio interface. The sun audio hardware is capable of recording and playing back audio data in U-LAW format with a sample rate of 8K per second. A full description can be gotten with man audio.

The module defines the following variables and functions:

[Sorry. Ignored \begin{excdesc} ... \end{excdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

17.1.1 Audio Device Objects

The audio device objects are returned by open define the following methods (except control objects which only provide getinfo, setinfo and drain):

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

[Sorry. Ignored \begin{funcdesc} ... \end{funcdesc}]

There is a companion module, SUNAUDIODEV, which defines useful symbolic constants like MIN\s\do5(G)AIN, MAX\s\do5(G)AIN, SPEAKER, etc. The names of the constants are the same names as used in the C include file <sun/audioio.h>, with the leading string AUDIO\s\do5()stripped.

Useability of the control device is limited at the moment, since there is no way to use the “wait for something to happen” feature the device provides.