Module

Marshal

The marshaling library converts collections of Ruby objects into a byte stream, allowing them to be stored outside the currently active script. This data may subsequently be read and the original objects reconstituted. Marshaling is described starting on page 431. Also see the YAML library on page 831.

Marshaled data has major and minor version numbers stored along with the object information. In normal use, marshaling can load only data written with the same major version number and an equal or lower minor version number. If Ruby's "verbose" flag is set (normally using -d, -v, -w, or --verbose), the major and minor numbers must match exactly. Marshal versioning is independent of Ruby's version numbers. You can extract the version by reading the first two bytes of marshaled data.

```
RUBY_VERSION                                          # =>   "1.9.1"
[ Marshal::MAJOR_VERSION, Marshal::MINOR_VERSION ]    # =>   [4, 8]
str = Marshal.dump("thing")
str.bytes.first(2)                                    # =>   [4, 8]
```

Some objects cannot be dumped: if the objects to be dumped include bindings, procedure or method objects, instances of class IO, or singleton objects, or if you try to dump anonymous classes or modules, a TypeError will be raised.

If your class has special serialization needs (for example, if you want to serialize in some specific format) or if it contains objects that would otherwise not be serializable, you can implement your own serialization strategy using the instance methods marshal_dump and marshal_load: If an object to be marshaled responds to marshal_dump, that method is called instead of _dump. marshal_dump can return an object of any class (not just a String). A class that implements marshal_dump must also implement marshal_load, which is called as an instance method of a newly allocated object and passed the object originally created by marshal_dump.

The following code uses this to store a Time object in the serialized version of an object. When loaded, this object is passed to marshal_load, which converts this time to a printable form, storing the result in an instance variable.

```
class TimedDump
  attr_reader :when_dumped
  attr_accessor :other_data
  def marshal_dump
    [ Time.now, @other_data ]
  end
  def marshal_load(marshal_data)
    @when_dumped = marshal_data[0].strftime("%I:%M%p")
    @other_data  = marshal_data[1]
  end
end


t = TimedDump.new
t.other_data = "wibble"
t.when_dumped      # =>   nil

str = Marshal.dump(t)

newt = Marshal.load(str)
```

Module constants

MAJOR_VERSION	Major part of marshal format version number.
MINOR_VERSION	Minor part of marshal format version number.

Module methods

dump

dump(*obj* ⟨ , *io* ⟩ , *limit*=–1) → *io*

Serializes *obj* and all descendent objects. If *io* is specified, the serialized data will be written to it; otherwise, the data will be returned as a String. If *limit* is specified, the traversal of subobjects will be limited to that depth. If *limit* is negative, no checking of depth will be performed.

```
class Klass
  def initialize(str)
    @str = str
  end
  def say_hello
    @str
  end
end

o = Klass.new("hello\n")
data = Marshal.dump(o)
obj = Marshal.load(data)
obj.say_hello   # =>   "hello\n"
```

load

load(*from* ⟨ , *proc* ⟩) → *obj*

Returns the result of converting the serialized data in *from* into a Ruby object (possibly with associated subordinate objects). *from* may be either an instance of IO or an object that responds to to_str. If *proc* is specified, it will be passed each object as it is deserialized.

restore

restore(*from* ⟨ , *proc* ⟩) → *obj*

A synonym for Marshal.load.