packedobjects egg

Cross-platform data encoding based on Packed Encoding Rules
Extension for Chicken Scheme
Version 0.5

John P. T. Moore

Table of Contents

1 About this egg

1.1 Version history

`0.5`	Alpha release

1.2 Usage

Load this egg like so:

```
(require-extension packedobjects)
```

2 Documentation

packedobjects is a highly portable, cross platform, data encoding tool. The project is based on the telecommunications standard Packed Encoding Rules (PER). An abstract syntax language is used to define a protocol specification. packedobjects uses the Scheme programming language to represent the protocol specification within a symbolic expression (s-expression). Using an s-expression provides a more dynamic approach over the traditional method of parsing the specification and producing high level language code. The output of applying packedobjects' encoding rules to a protocol specification is a concisely encoded bit stream suitable for application domains such as network games development and mobile application development. The project has been released under the terms of the BSD license.

For a more detailed introduction refer to the following short paper.

`packedobjects` [procedure]

```
(packedobjects <protocol> #!key <pdusize> <strsize>)
```

Creates a packedobject using the supplied protocol specification. Optional values for the PDU and string buffer sizes can be specified. They default to 5000 and 1000 respectively. The PDU buffer can grow dynamically, however, the string buffer is fixed in size and therefore must handle the largest possible string. The PDU buffer must be freed manually whereas the string buffer will be automatically garabage collected.

`<packedobject>` [procedure]

```
(<packedobject> 'pack <values>)
```

Encodes a list of values and returns the number of bytes used.

`<packedobject>` [procedure]

```
(<packedobject> 'unpack)
```

Returns a list of values.

`<packedobject>` [procedure]

```
(<packedobject> 'read <fd> <size>)
```

Reads a number of bytes from a file descriptor into the PDU buffer.

`<packedobject>` [procedure]

```
(<packedobject> 'write <fd> <size>)
```

Writes a number of bytes from the PDU buffer to a file descriptor.

`<packedobject>` [procedure]

```
(<packedobject> 'dump-buffer <filename> <size>)
```

Dumps a number of bytes from the PDU buffer to a file.

`<packedobject>` [procedure]

```
(<packedobject> 'free)
```

Frees the PDU buffer.

3 Data types

string

`desc:` 7 bit string limited to 10 characters in length.

`spec:` (foobar string (size 1 10))

`values:` (foobar "foobar")

octet-string

`desc:` 8 bit string with no length restriction.

`spec:` (foobar string ())

`values:` (foobar "foobar")

bit-string

`desc:` Bitstring fixed in length.

`spec:` (foobar bit-string (size 8))

`values:` (foobar "10101010")

hex-string

`desc:` String of hexadecimal characters with no length restriction.

`spec:` (foobar hex-string (size 1 max))

`values:` (foobar "foobar")

integer

`desc:` Integer with a minimum allowed value of 1 and maximum allowed value of 10.

`spec:` (foobar integer (range 1 10))

`values:` (foobar 5)

boolean

`desc:` Boolean value.

`spec:` (foobar boolean)

`values:` (foobar #t)

enumerated

`desc:` List of alternative symbols.

`spec:` (foobar enumerated (mon tues wed thurs fri))

`values:` (foobar fri)

sequence

`desc:` Ordered sequence of types.

`spec:` (foobar sequence (foo string (size 1 10)) (bar string (size 1 10))) `values:` (foobar (foo "foo") (bar "bar"))

set

`desc:` Unordered sequence of types. Each element of a set is also optional.

`spec:` (foobar set (foo string (size 1 10)) (bar string (size 1 10)))

`values:` (foobar (bar "bar") (foo "foo"))

sequence-of

`desc:` Ordered sequence of types where the sequence may repeat.

`spec:` (foobar sequence-of (foo string (size 1 10)) (bar string (size 1 10)))

`values:` (foobar ((foo "foo") (bar "bar")) ((foo "anotherfoo") (bar "anotherbar")))

choice

`desc:` Single item from a series of types.

`spec:` (foobar choice (foo integer ()) (bar integer ()))

`values:` (foobar (bar 100))

4 Examples

```
(require-extension packedobjects)

(define bbcard
  '(bbcard sequence
           (name string (size 1 60))
           (team string (size 1 60))
           (age integer (range 1 100))
           (position string (size 1 60))
           (handedness enumerated (left-handed right-handed ambidextrous))
           (batting-average sequence
                            (mantissa integer ())
                            (base enumerated (2 10))
                            (exponent integer ())))))

(define bbcard-values
  '(bbcard
    (name "Casey")
    (team "Mudville Nine")
    (age 32)
    (position "left field")
    (handedness ambidextrous)
    (batting-average
     (mantissa 250)
     (base 10)
     (exponent -3))))

(define po (packedobjects bbcard))
(print* "encoded in " (po 'pack bbcard-values) " bytes.\n")
(print (po 'unpack))
(po 'free)
```

5 Extended Example

The following example retrieves random numbers from random.org.

The client talks to a server process which in turn obtains the numbers using HTTP. The data transferred between client and server is significantly more efficient, in terms of bits on the wire, than if the client talked directly to random.org using HTTP.

Download the source files:

- `http://www.call-with-current-continuation.org/eggs/tcpclient.scm`
- `http://www.call-with-current-continuation.org/eggs/tcpserver.scm`

6 Limits

- The length of strings, the number of elements in a set and the number of times a sequence-of may repeat is restricted to 2^30-1.
- The maximum integer (unsigned), the number of choices and the number of enumerations is restricted to 2^32-1.

7 To do

- Performance enhancements.
- Improvements to error/exception handling.
- Further testing.

8 License

Index