

NAME

encode()/decode() – data stream manipulation

DESCRIPTION

The `encode()` and `decode()` built-in functions convert between `spec` data objects and standard data-interchanges formats. Such a capability may be useful, for example, to pass data over sockets between `spec` and other processes. Currently, the JSON (JavaScript Object Notation) format is supported.

The `spec` data objects are numbers, strings, associative arrays and data arrays. The `encode()` function takes as argument one or more `spec` data objects and returns a string conforming to the data-interchange format specifications. The `decode()` functions takes a string argument which contains encoded data and returns a `spec` data object appropriate to the data. Since `spec` does not have composite data types, the `decode()` function only accepts encoded representations of one kind of value, but that value can be an associative array structure or a data array containing multiple items.

BUILT-IN FUNCTIONS

In the `encode()` and `decode()` functions, the first argument is the name of the format. Currently, "json" is the only recognized format. The argument is case insensitive.

`encode(format, obj [, ...])` – Returns a string representation of the `spec` data objects in the specified `format`.

`decode(format, str)` – Returns a `spec` data object obtained from the string representation in `format`.

FORMATS

The JSON format is documented at json.org. In general, a JSON value can be a string, a number, an object or an array. Arbitrary white space is allowed between the various elements.

Strings are always delimited by double quotes. Within strings, octal string escape sequences are not allowed. Instead, `\u` followed by *four-hex-digits* is used. Since `spec` only supports eight-bit characters, the two high-order hex digits are ignored when decoding such sequences.

Numbers can only be in decimal format (no octal or hexadecimal), and leading zeroes are not allowed.

A JSON object is a comma delimited list of name/value pairs (separated by a colon), enclosed in curly brackets. The name is always a string. `spec` only supports values that are numbers or strings. The object type is used for encoding and decoding associative arrays.

The array object is a comma delimited list of numbers or strings enclosed in square brackets. `spec` supports one- or two-dimensional number arrays and one-dimensional string arrays. `spec` can encode an array of mixed values, but can decode arrays that have only one value type.

EXAMPLES

The following examples illustrate JSON encoding.

`spec` associative arrays are encoded as a JSON `object`, which is a set of name/value pairs, enclosed in curly brackets. The name is always a string. `spec`'s encoded objects will only include string and number value types. To encode a `spec` associative array:

```
1.SPEC> s = encode("json", A)

2.SPEC> print s
{"0": 120, "1": 60, "2": -35.2645, "3": -45}
```

spec number data arrays can contain signed or unsigned 8-, 16- or 32-bit integers, 32-bit floats or 64-bit doubles and can have one or two dimensions. To encode a data array:

```
3.SPEC> ulong array data[10], data[2][5]
```

```
4.SPEC> array_op("fill", data)
```

```
5.SPEC> s = encode("json", data)
```

```
6.SPEC> print s
[0,1,2,3,4,5,6,7,8,9]
```

```
7.SPEC> array_op("fill", data2)
```

```
8.SPEC> print encode("json", data2)
[[0,0,0,0,0],[1,1,1,1,1],[2,2,2,2,2]]
```

A spec string data array is similar to an 8-bit array with respect to storage, but the contents will be encoded as JSON strings. A one-dimensional string array will be encoded as a single string. A two-dimensional string array will be encoded as one-dimensional array of strings.

```
9.SPEC> string array sdata[20], sdata2[3][20]
```

```
10.SPEC> sdata = "\033[7mtest\033[0m"
```

```
11.SPEC> sdata2[0] = "first"
```

```
12.SPEC> sdata2[1] = "middle"
```

```
13.SPEC> sdata2[2] = "last"
```

```
14.SPEC> print encode("json", sdata)
"\u001B[7mtest\u001B[0m"
```

```
15.SPEC> print encode("json", sdata2)
["first","middle","last"]
```

Note the octal escapes in the first example have been encoded using `\u` followed by four hexadecimal digits.

A list of mixed data objects can also be encoded:

```
16.SPEC> s = encode("json", PI, "hello world", data, A)
```

```
17.SPEC> print s
[3.14159265358979, "hello world", [0,1,2,3,4,5,6,7,8,9],
{"0": 120, "1": 60, "2": -35.2645, "3": -45}]
```

Here, the list is encoded as an array of values consisting of a number, a string, an array and an object.

The `decode()` function will only accept JSON encoded strings that can be decoded to one of the recognized spec data types. Mixed data types, such as combinations of scalars and arrays in the last example above, are not allowed.