NAME

TclList – Python interface to Tcl list objects

SYNOPSIS

Python Usage:

import SoftwareAlchemy.Common.TclList

tl **= TclList(***tcl_string***='',** *make_numeric***=False,** *from_list***=None)**
*tl***.make_numeric()**
*tl***.from_list(***iterable***)**
string **=** *tl***.to_tcl()**

Go Usage:

func ParseTclList(*s* **string) ([]string, error)**
func ToTclString(*listval* **[]string) (string, error)**

DESCRIPTION

Some of the older elements of GMA (which used to be entirely written in the Tcl language, after it was ported from the even older C++ code) use Tcl list objects as their data representation. Notably the biggest example is the **mapper**(6) tool (which is still written in Tcl itself), whose map file format and TCP/IP communications protocol include marshalled data structures represented as Tcl lists.

While this is obviously convenient for Tcl programs in that they can take such strings and natively use them as lists of values, it is also useful generally in that it is a simple string representation of a simple data structure. The actual definition of this string format is included below.

The **TclList** Python object provides an easy Python interface to manipulate Tcl lists as Python lists. (Courtesy of the fact that the Python's **tkinter** standard library module includes an embedded Tcl interpreter, this is done by calling into the Tcl interpreter, so this should be very accurate in terms of being genuine Tcl lists and not an approximate emulation.)

To create a **TclList** object from a string representation of a Tcl list (such as a value read from a map file), simply call the constructor

tl **= TclList(***tcl_string***)**

Now you can access *tl* as if it were a Python **list** object. For example, after executing

tl = TclList('1 2 {Hello world} {3 4 {5 6}}')

then the contents of **tl** will be **['1', '2', 'Hello world', '3 4 {5 6}']**.

A couple of things to note here. First, since no type information is provided in the Tcl list representation (and in fact Tcl uses dynamic weak type bindings anyway), all values are assumed to be strings. Second, there is no way to distinguish a string which contains spaces from a sub-list, so "{Hello world}" may be a string of two words or a list with two elements. You are expected to simply use each appropriately in each case. If we wanted to consider element 3 of **tl** in this example as a list, we can construct a new **TclList** for it:

tl2 = TclList(tl[3])

which gives us **tl2=['3', '4', '5 6']**.

If the constructor's optional *make_numeric* parameter is **True**, it will convert every element which appears to be a numeric value into a Python **int** or **float** object:

tl = TclList('1 2 {Hello world} {3 4 {5 6}}', make_numeric=True)

in this case, the contents of **tl** will be **[1, 2, 'Hello world', '3 4 {5 6}']**.

You can also invoke the **tl.make_numeric()** method later to convert numeric elements at that time.

Alternatively, if you have a Python iterable object such as a list or tuple, you can call the constructor as

tl **= TclList(from_list=***iterable***)**

in which case, the **TclList** object will be created from the *iterable*. If the *from_list* parameter is given, then *tcl_string* is ignored. The *make_numeric* parameter is still respected and will force elements to numeric where possible if it is **True**.

If you already have a **TclList** object *tl*, you may call the *tl***.from_list()** method with an *iterable* parameter. This will replace the existing elements of *tl* with those from the iterable. For example,

tl = TclList()
*tl***.from_list([1, 2, [3, ’a’, ’b’], 4, 5], ’spam spam eggs spam’])**

or

tl = TclList(from_list=[1, 2, [3, ’a’, ’b’], 4, 5], ’spam spam eggs spam’])

both yield values in **tl** of

[1, 2, ’3 {a b} 4 5’, ’spam spam eggs spam’]

A **TclList** object may be converted to a valid Tcl string representation by calling the *tl***.to_tcl()** method. For example, given the *tl* object from the previous example, calling

tlist **=** *tl***.to_tcl()**

will assign the string "**1 2 {3 {a b} 4 5} {spam spam eggs spam}**" to *tlist*.

Go API Variant

In the Go version, two functions are provided to simply convert between a discrete list of string values (as a slice of strings) and a TCL-formatted string representation. No attempt is made to convert numeric values in order to preserve the strong typing of the result value.

ToTclString(*listval***)** takes a slice of strings as input, returning a single string result. **ParseTclList(***s***)** parses an input string *s*, returning a list of strings representing that list's elements. Both functions return an error in case they were unable to properly deal with their input values.

TCL LIST FORMAT

In a nutshell, a Tcl list (as a string representation) is a space-delimited list of values. Any value which includes spaces is enclosed in curly braces. An empty string (empty list) as an element in a list is represented as "**{}**". (E.g., "**1 {} 2**" is a list of three elements, the middle of which is an empty string.) An entirely empty Tcl list is represented as an empty string "".

A list value must have balanced braces. A balanced pair of braces that happen to be inside a larger string value may be left as-is, since a string that happens to contain spaces or braces is only distinguished from a deeply-nested list value when you attempt to interpret it as one or another in the code. Thus, the list

"**a b {this {is a} string}**"

has three elements: "**a**", "**b**", and "**this {is a} string**". Otherwise, a lone brace that's part of a string value should be escaped with a backslash:

"**a b {this \{ too}**"

Literal backslashes may be escaped with a backslash as well.

While extra spaces are ignored when parsing lists into elements, a properly formed string representation of a list will have the miminum number of spaces and braces needed to describe the list structure.

DIAGNOSTICS

These functions may raise the following exceptions in addition to the normal set that may occur otherwise:

TclListFormatError
The string does not represent a valid Tcl list.

SEE ALSO

AUTHOR

Steve Willoughby / steve@alchemy.com.

HISTORY

The Go port appeared in November, 2020, in GMA version 4.2.2. The original Python version calls directly into the Python interpreter's embedded Tcl interpreter to perform the required translations. The Go version implements the conversion operations directly, including a minimal amount of code ported from the open-source Tcl interpreter code.

BUGS

COPYRGHT