Preliminary notes/documentation for the calldll module, version 0.2.

Jack Jansen, jack@cwi.nl, 23-May-97.
日本語翻訳：山本　昇、高エネルギー加速器研究機構、加速器研究施設

May 31, 2001

Abstract

calldll モジュールは C コードを一行も書くことなく、Python から任意の C 関数を呼び出すことを可能にする。calldll の主な目的は、Python ラッパモジュールがまだ用意されていない MacOS ツールボックスの関数を呼び出すことにある。calldll は不完全でもある。すなわち現在の所極めて限られた引数の型だけがサポートされている。どうかあなたの必要としている引数の型を教えて欲しい。あるいは、どんな型でも引き渡すことが出来るような一般的な"逃げ道"についての良いアイデアがあるなら私に教えて欲しい。
Calldll allows you to call random C functions from python without writing any C code. It is mainly meant to call MacOS toolbox routines for which no Python wrapper module is available. It is also incomplete, in that only a few argument types are currently supported. Please let me know which other argument types you need, and/or whether you have any ideas on a general "escape" allowing people to pass anything.

1

calldll モジュールは現在のところ、*PowerPC* **でのみ**動作する。calldll は完全なソースコードとプロジェクトファイルが配付されている、だから CFM68K の環境に移植しようと望む方は、そうすることを歓迎します。クラッシック 68K 環境への移植は不可能である、と思っている。(だから、どなたか私が間違っていることを証明してください:-) .

The module works *only* on PowerPC currently. It is distributed complete with source and project files, so that people willing to look at a CFM68K port are welcome to do so. A classic 68K implementation is impossible, I think (so prove me wrong, please:-).

モジュールは三つの関数を提供する：The module exports three functions:

symtable = getlibrary(libraryname) ライブラリ "libraryname" をインポートするためのリファレンスを取得する。多くのツールボックス関数を含んでいる "InterfaceLib" はもっともよく使われるものである。戻り値であるシンボルテーブルは newcall に渡される関数を検索することに使われる。"symtable.WaitNextEvent" は WaitNextEvent 関数のアドレスを返す。 "symtable['WaitNextEvent']"としても同じである。戻り値の symtable は Python のマッピング型データである。したがって、中身を調べるために、Keys() や len(...) を使える。

symtable = getdiskfragment(fss, file) (fss, file) の組で指定されるファイル[1]をロードして、そのシンボルテーブルへのリファレンスを返す。

callable = newcall(routine, returntype, [argtype, ...)] 呼び出し可能な Python オブジェクトを返す。(上で説明されたように、) 呼び出したい C-関数、C-関数の戻り値の型、そして引数の型を引数として指定する。値として返されたオブジェクトは通常の方法で Python コードの中で使用することが出来る。関数オブジェクトの実行時には引数の型がチェックされる。(もちろん newcall 呼び出し時に指定した引数の型が誤っているなら、Mac をクラッシュさせる可能性はある。) 呼び出し可能オブジェクトを印刷すると、C の呼び出し規約の記述が印刷される。

[1] オリジナルのドキュメントは引数が一つであるが、サンプルコードでは fss,filename) が引数となっている。実行してみても引数が一つではエラーになる。

- symtable = getlibrary(libraryname) Get a reference to import library libraryname. ”InterfaceLib” is the most commonly used one, containing most toolbox routines. The symbol table can be used to lookup routines to be passed to newcall: ”symtable.WaitNextEvent” will return the address of routine WaitNextEvent. and so will ”symtable['WaitNextEvent']”. The symtable is a mapping, so you can use keys() and len(...) to inspect it. - symtable = getdiskfragment(file) Load the specified file (given by fsspec or filename) and return a reference to its symboltable. - callable = newcall(routine, returntype, [argtype, ...]) Return a callable object. You specify the C routine to be called (as explained above), the type of the return value and the argument types. The resulting object can be called from Python code in the normal way, and typechecking on arguments is performed (but, of course, if you specify incorrect argument types in this call you may well crash your machine). Printing a callable will give you a description of the (C-) calling sequence.

C 関数の返す値は、'None', 'Byte', 'Short', 'Long', 'Pstring'（アドレスとして返されたパスカル文字列、Python 文字列にコピーされる), 'Cobject'(void ポインタのラッパー),'Handle' (新たなハンドル,Res.Resource object として返される。) あるいは 'OSErr' (ゼロで無ければ、MacOS.Error 割り込みを発生する) のうちいずれかでなければならない。

The C return value can be one of 'None', 'Byte', 'Short', 'Long', 'Pstring' (a pascal string returned by address, copied to a Python string), 'Cobject' (a wrapper around a void pointer), 'Handle' (a new handle, returned as a Res.Resource object) or 'OSErr' (which raises MacOS.Error if non-zero).

newcall で指定するの引数型は、'InByte', 'InShort', 'InLong', 'InString' (a python 文字列, C 関数にはデータへのポインタが渡される。使用に当たっては、慎重な注意が必要である。), 'InPstring' (Python 文字列が Str255 にコピーされ、アドレスが実際の C 関数に渡される）、'InCobject', 'InHandle', 'OutByte' (一バイト分の記憶領域が割り当てられ、そのアドレスが C 関数に渡される。結果の値は、Python に返される。), 'OutShort', 'OutLong', 'OutPstring' (再び記憶領域が事前に割り当てられ、そのアドレスがC関数の渡される。), 'OutCobject' ((void*) 型の記憶領域が割り当てられ、(void **) が C に渡される。戻り値の (void *) は Cobject 型に埋め込まれて Python に返される。) あるいは 'OutHandle' (承前。つまり、この型は普通使われない。通常”InHandle”が使われるはずである。なぜなら、ほとんどのツールボックスは事前にユーザがハンドルを割り当てていると期待しているからである。）のいずれであっても良い。

Arguments can be any of 'InByte', 'InShort', 'InLong', 'InString' (a python string, with the address of the data passed to the C routine, so be careful!), 'InPstring' (a python string copied to a Str255 and passed by address), 'InCobject', 'InHandle', 'OutByte' (storage is allocated for a single byte, the address passed to C and the resulting value returned to Python), 'OutShort', 'OutLong', 'OutPstring' (again: storage pre-allocated and the address passed to C), 'OutCobject' (storage for a void * is allocated, this void ** is passed to C and the resulting void * is encapsulated in the Cobject returned) or 'OutHandle' (ditto, which means that this is usually *not* what you use, you normally use 'InHandle' because most toolbox calls expect you to preallocate the handle).

すべての戻り値 (関数の戻り値および Out 引数で返される値) は集められる。もし一つも返す値が無ければ None がかえされる。もし戻り値が一つであれば、その値が返される。もし一つ以上の戻り値があれば Python の tuple が返される。All values to be returned (from the return value and the Out arguments) are collected. If there aren't any None is returned, if there is one value this value is returned, if there are multiple values a tuple is returned.

testcalldll.py にテストコードが、samplecalldll.py には最小限の例が収められている。There is test code in testcalldll.py, and a minimal example in samplecalldll.py.

楽しんでくれたまえ、そしてこれらのデザインについて pythonmac-sig で議論しよう。

Have fun, and let's discuss the design of this thingy on pythonmac-sig,