About a way of working on desktop Forth systems with (external) functions found in dynlibs and such.

Rationale for using functionality(!) as given by **C(** and **CALL**

Flexibility of usage because of splitting the dynlib interface in two:
CALL and the **C(...)C-** chain
Respectively produce an external's address and preparing for executing it.
Doesn't interfere with other dynlib interfaces.
Usage in parallel no problem.
No buried tools.
Easy to use as building blocks for more elaborate and fancy constructions.
Easy to add new types.
Works in 32 as well 64 bit.

NoTE: the **C(** _etc **)C-** words are by Marcel Hendrix, the author of iForth.
The **CALL** and **C:** words plus their derivatives are by me, unless noted otherwise.

Help for **C(** and family as used in iForth

C((-- mask)
Prepares system for compiling an external function.
Creates an initial parameter mask ud for 32bit or u for 64bit iForth versions.
It leaves the system in interpretive mode (if not there already).
Normally **C(** is used together with one of the **)C-** words to compile an external function call.
C(is **IMMEDIATE** but not(!) Compile Only.

The mask represents the type and order of the parameters passed to the external function.
Words following **C(** can add or change the mask.
This is typically done by words like:
_int _float _void _double
who add their type to the mask. See miscutil.frt for what's defined.
The mask building is finished and used by any of the **)C-** words.

You can write your own version of **C(** as long as it complies with the above:
: myC((-- mask) postpone c( ] _int _int postpone [; immediate
Here the mask will start with two ints added.

)C-int (imp mask --)
Compiles a call to an external function imp using the mask to compile the parameter passing to imp.
Imp is a pointer to an external function as found in an imported library.
The call returns an int as implicated by -int (... -- n)

)C-float (imp mask --)
Compiles a call to an external function imp using the mask to compile the parameter passing to imp.
Imp is a pointer to the external function as found in an imported library.
The call returns a sfloat as implicated by -float (... -- ...) (F: ... -- ... float)

)C-long (imp mask --)
Compiles a call to an external function imp using the mask to compile the parameter passing to imp.
Imp is a pointer to the external function as found in an imported library.
The call returns a double as implicated by -long (... -- ud)

)C-void (imp mask --)
Compiles a call to an external function imp using the mask to compile the parameter passing to imp.
Imp is a pointer to an external function as found in an imported library.
The call returns nothing as implicated by -void (... -- ...)

)C-double (imp mask --)
Compiles a call to an external function imp using the mask to compile the parameter passing to imp.
Imp is a pointer to the external function as found in an imported library.
The call returns a double as implicated by -double (... -- ...) (F: ... -- ... double)

)C-sxint (imp mask --)
Compiles a call to an external function imp using the mask to compile the parameter passing to imp.
Imp is a pointer to the external function as found in an imported library.
The call returns a sign extended 32bit->64bit int as implicated by -sxint
(... -- n)

Caution: **C(...)C-** does not necessarily depicts the same as the Forth data stack picture!
C(_int _int)C-int is the same as (n1 n2 -- n3)
C(_int _int _float)C-int is the same as (n1 n2 -- n3) (F: r --)
C(_int _int)C-float is the same as (n1 n2 --) (F: -- r)
etc.

Help for **CALL** and **C:** as implemented for iForth and VFX

CALL (spaces<name> --) (Runtime: -- addr)
Get pointer to name's entree in the current open foreign library.
Compile code to push pointer (imp) on the data stack in current definition.
Info about the pointer and where it's declared is added to the **EXTERNALS** chain for **/EXTERNALS**. This ensures relinking at booting.
NoTE name is _not_ added to the Forth dictionary.

C: (spaces<name> --) (Runtime: -- addr)
Create a word 'name' while doing a **CALL** on name.
Now name is added to the Forth dictionary.

Usage

```
C: abs C( _int )C-int ;
is the same as
: abs CALL abs C( _int )C-int ;

useful for synonyms
: _abs CALL abs C( _int )C-int ;

no problem adding Forth stack notation
could be handy for some ;-)
C: abs ( n -- |n| )  c( _int )c-int ;

for variable input, varargs functions, the C( ... )C- chain later decides
the parameters passing and running the function
C: objc_msgSend ( -- function-address ) ;
objc_msgSend only leaves its function pointer when run

also used for global exported data, use @ on those
C: NSApp ( -- global-address ) ;
```

flexibility of usage because of **CALL**
: myrandom (-- |n|) **random CALL** abs (n -- |n|) **c(_int)c-int ;**

Extensions

FUNCTION: (spaces<name> spaces(stack picture> --)
SwiftForth's **FUNCTION:** and **GLOBAL:** can be created with **CALL** and the **C(...)C-** chain
Done in iForth, though not used. Also done in VFX (without the **C(...)C-** chain), in use!

EXTERN: ("text" --)
iForth's **EXTERN:** is made using the **C(...)C-** chain by Hanno Schwalm.

FUNCTION: abs (n1 -- n2)
is the same as
EXTERN: int abs(int i);
but also the same as
C: abs **c(_int)c-int ;**
and
: abs **CALL** abs **c(_int)c-int ;**
whatever you prefer!

Variants

the following words are used in an interface to Objective-C

COCOA((-- mask)
Similar to **C(** but two _ints added to initial mask, referring the object and selector.

CALLCOCOA (spaces<name> --)
Similar to **CALL** but retrieve the selector from method name and compile code to pass it as parameter and compile **objc_msgSend**

COCOA: (spaces<name> --)
Create a word 'name' while doing a **CALLCOCOA** on name.
Now name is added to the Forth dictionary (and the message/selector sending mechanism is now similar to calling an external function).

Example:
cocoa: @processInfo (-- NSProcessInfo:info) **cocoa(_void)c-int ;**

Notes about the implementation

The **)C-** words.
The real workhorse. The gory details about the parameters wrt alignment, order, passing via registers or stack etc. is dealt with here. Also setting up the prolog and epilog i.e. preparing executing the external function is done here.
The **)C-** words contain words from the **FOREIGN** family. These are the iForth versions for the **varXcall** (VFX) and **EXTCALL** (SwiftForth) group of words.

The **C(_int _float _etc)C-** words are non parsing. They're all colon definitions, using the stack for their parameters. It's the Forth interpreter's job to deal with these words. Easy to maintain!

CALL obviously makes use of library-find like words, retrieving a function pointer.