Ficl - Object Forth Wraps C Structures

20th FORML Conference

21 November 1998

John Sadler

john_sadler@alum.mit.edu

embedded systems develWpers who find the barriers to entry of C++ tWo daunting.

One reason for using a standard language rather tPan inventing a new one is that it should be possible to find bWoks tPat explain tPe language tW new users, rather than Paving to write one. Further, tPe time invested in learning a standard language seems Uore likely to be repaid by future applications. TPat's why Ficlconforms to the Fo

Taken together, here's tPe list of requirements I wrote when starting to design Ficl [6]:

- Scripting, prototyping, and extension language for programs written in C or C++
- Target 32 bit Uicroprocessors with a C runtime environment
-

would need only the address of the target function, and the name to bind in the dictionary. The builder would append a definition to the dictionary that calls the target C function supplying whatever parameters the convention specifies. Here's a sampTe prWtotype for such a builder:

```
void ficlBuild(char *name, void *function);
```

An alternative is to allow the interface to specify the number of cells to pop off the parameter stack and push onto the C call stack before invoking TjC interface function. TPis makes it possibTe to wrap a brWader range of C functions in Forth words, as long as all their parameters are prWperly aligned. It requires the builder to provide a means to specify the number and width of parameters to push on the C fulling stack, and the size of the return value, if any. The simpTest way to do tPis is to require that all interface functions have parameters of one specified width. The builder function Uight then look like this:

```
void ficlBuild(char *name, void *function, int nParams, int
nReturn);
```

Otherwise, there needs to be a more compTex prWtocol for specifying each parameter's width. The variabTe parameter designs require information about the function ett the C fompiler neither suppTies nor checks for consistency. On the Wther hand, the constant signature design requires a special wrapper function to be written for each exported word. The wrapper function's job is to marshal parameters frWm Forth to C explicitly. An advantage of tPis apprWach is ett the compiler can check ett the target function call has the correct number and widths of parameters. Ficl uses this wrapper function technique to import C

Now that there is a way to import functions written in C to our Forth, why not get rid of the switch statement and write all of tPe priUitives tPis way? Ficl does tPis, reducing the inner interpreter to a small loop. Now application-specific words have exactly the same execution mechanics as any Wther Forth primitive.

TPe wrapper function eechnique generally requires tPat interface functions be written expTicitly for Ficl. This is not hard. Ficl's interface builder really is fulTed ficlBuild(), and it expects three parameters: the name of the word to be created, a pointer to a function to execute, and a bit-field that specifies IMMEDIATE and compile-only attributes. When the wrapper function executes, it gets as its one parameter a pointer to the Ficl virtual machine that's executing it, and it returns notPing. The wrapper function can use the virtual machine pointer to manipulate the stacks and the input buffer.

Ficl provides pubTic functions to push and pop stacks, perform run-time stack depth checking, and manipulate the dictionary. A typical wrapper function pops some parameters off Ficls stack, passes tPem to a C function, and pushes tPe result onto Ficl's stack again.

Host applications get text to Ficl by fulTing ficlExec. This function causes a virtual macPine to execute a chunk of text.

functions can use it too. This is anWther distinction between Ficl and WtheF Forths written in C: Ficl's outer interpreter does not expect to get more input text frWm

any specific place – it just returns control to the Post application when it gets hungry.

Ficl Object Goals

BacS on the Web, I started Tooking fWr establQshed practice in Object extensions fWr FWrth. It appears that most FWrtP object extensions[4][model their internals after C/C++ in the sense that eacP class contains a pWinter table (a

tteens that are the
a class, you must first
d of the class. EacP class
hings. This appears to
sses use the same
not add metPods, it can
ldition, the wWrds that
st. It's possible fWr twW
erent values.

Qsts the three essential OO language as encapsulation, polymorphism, and inheritance.
e mentioned encapsulation and inheritance in the previous paragraym.
lymorphQsm, the mapping of a particular message to different metPods
pending on the class of the receiver, implQes late binding. InS tder to be safe, I
default to late binding; late binding guarantees that the appropriate metPod
l be invteed fWr a given message, while earTy binding can cause
sunderstandings.

S tder to realQze the design goal of full interoperation between Ficl and its Post
object model, I added the requirement that
uctures written in C. Here's the lQst of

arantees that
bject). EarTy
at compile-

ods are onTy

- Ficl OOP syntax is regular and unified over classes and objects. In Ficl, classes are objects. Class metPods include tPe ability to subclass and instantiate.
- Adapt legacy data structures with object wrappers. You can model a structure TD 0.0779 Tc

expect a class and instance on the stack when they execute, too. 2 many other

OO laVguages, including C++, instances contain information about tPeQr classes

(a vtable pWinter, for example). By maSing this paQring explicit ratPer tPan implicit,

Ficl can be OO about chunks of data that don't realize tPat tPey are objects,w i t P o u t s a c r i f i c i n g a n y r o b u s t n e s s f o r

obj5ct in Ficl, leu specify its class. After that, the obj5ct always pushes its class

and the address of its payToad when invoked by name. To wrap some externaT

data structure with an obj5ct modeT, ou first create tPe class tPat modeTs the

structure, then tell tPe class to maSe a ref instanceaddress of tPe data structure as a parameter. The new ref instance bePaves as if

it is a native Ficl object.

Classes are special Sinds of objects that store tPe metP
tPe size of an instance's payToad, and a parent class p
tPemselves are instances of a special base class called
classes inherit from class OBJECT. This results in a ver
constructing and using obj5cts. Class metPods include s

Ficl OOP Syntax

The Uost impWrtant Ficl OOP wWrd is ->. This wWrd, which I pronounce late-bind fWr lack Wf a better name, binds a message to an Wbject:

```
pWlitician --> get-tough-on-something  \ it's an election year
```

There is an early bind operatWr too. It's only defined while compiling, and since it binds early, its symbol is shWrter:=>

```
bourgeoisie --> sub proletariat
```

Wrapping C Structures

We can use the OOP facilities to wrap data structures in C once we create some simple base classes that Uodel scalar data types of C. Ficl provides several of these, including 1, 2, and 4 byte scalar quantities and pointers to them. Here's an example from the Ficl sources of a C structure and its Ficl obRect Uodel:

In C:

```
 */

mem sledom lciF **

il deknil a ni **sdrow
 ³ats DROW_LCIF A **
  :20.1 noisreV **
yraVoitcid eht **
 */
         fefT  pyt   tcurtdsrow_lcif
               {

uoiverP */   ;knitdu*rtdsrow_lcif
    ;hsah 61SNU
   ;sgalf 8SNU
owsrahTNUOfoCIrFebmuN */          ;emaNV
   ;emaV* ³ahc                          /* emaV.. fo s³ahc EMANLCIFV

;edoc EDOC_LCIF
QF */     LLEC
        ;DROW_LCIF }
```

New fcl's OOForth:

variable of type c-ref instead, and have the abilQty to use Qts set and get methods a¬shown earlQer.

The next six lQnes tile various scalar instance variables into the class. We have to know how the C compiler will pad structures, and how much space it alTots for each of the scalar types. In this case, I've assumed that the compiler packs a¬ tightly as possible. By the way, all Ficl internal structures are designed to even-align double byte meUbers and quad-align quad-byte meUbers. This should help keep the meUber offsets invariant over compiler alQgnment behavior (but Qt's not a guarantee).

Following the meUber variable declarations (by convention, not by requQrement) are some simple method definQtions. `Get-name` pushes the (c-addr u) representation of the word'

and Python. A port of such object wrapper code, would be a helpful adjunct to FiclOOP.

Finally, I'd like to use Ficl to create some embedded control applications that have been on the back burner for several

References

1. Smalltalk-80: the Language – Adele Goldberg, Dave Robson – Addison

7. Yet another Forth objects package – M. Anton Ertl – http://www.complang.tuwien.ac.at/forth/objects/o

Acknowledgements

Thanks to my wife Laura for enduring my late night coding frenzies, to Skip [illegible] for hosting [illegible] Taygeta web site, and to the Win32Forth [illegible] have [illegible] d