common-lisp-types.lisp

Erik Winkels

August 13, 2009

Contents

1 Type Handlers

1.1 Type Handler Keywords

1.1.1 :after

Code that comes on the line after the call to the cfun.

1.1.2 :arg-type

The argument type it should take in the defcfun.

1.1.3 :before

Code that precedes the line of the call to the cfun.

1.1.4 :return-arg

Ogre returns Vector3, Quaternions and other objects. I have chosen to pass a CFFI allocated array by reference to the wrapper functions and have that array filled with the relevant values of the returned object. If you check the CL-ARRAY3 function below you'll see that the :below line allocates the array and :return-arg makes sure that what was actually a returned object by Ogre is now the name of the array reference.

1.1.5 :return-type

This is either a string or a cons. If it is a string just signifies what the return type of the defcfun should be. If it is a cons then its car signifies the return type of the defcfun (generally :bool or :void) and its cdr signifies that we should also generate an argument for to the C wrapper function. This is usually to pass an array by reference instead of returning a pointer. To take CL-ARRAY3 as an example again, this is often used for Ogre functions that return a Vector3. Instead of returning a pointer to a Vector3 object we allocate an array that can hold 3 Ogre::Reals, pass that to the C-wrapper which fills it with the x, y and z values of the Vector3 object and a CFFI translator returns it as a (vector x y z). Oh, and the array gets deallocated again by CFFI.

1.2 Type Handler Functions

1.2.1 cl-arrayX

These should ofcourse be done programatically instead if enumerating them like we do now.

```
(defun cl-array2 (type name)
  (declare (ignore name))
  (case type
    (:after (mkfstr "(vector (mem-aref array 'okra-real 0) "
                            "(mem-aref array 'okra-real 1))"))
    (:arg-type "okra-array2")
    (:before "(with-foreign-object (array 'okra-real 2)")
    (:overloaded-type "'(simple-vector 2)")
    (:return-arg "array")
    (:return-type '(":void" . "(array2 :pointer)"))
    (otherwise nil)))


(defun cl-array3 (type name)
  (declare (ignore name))
  (case type
    (:after (mkfstr "(vector (mem-aref array 'okra-real 0) "
                            "(mem-aref array 'okra-real 1)~%"
                "            (mem-aref array 'okra-real 2))"))
    (:arg-type "okra-array3")
    (:before "(with-foreign-object (array 'okra-real 3)")
    (:overloaded-type "'(simple-vector 3)")
    (:return-arg "array")
    (:return-type '(":void" . "(array3 :pointer)"))
    (otherwise nil)))


(defun cl-array4 (type name)
  (declare (ignore name))
  (case type
    (:after (mkfstr "(vector (mem-aref array 'okra-real 0) "
```

```
                            "(mem-aref array 'okra-real 1)~%"
                "            (mem-aref array 'okra-real 2) "
                            "(mem-aref array 'okra-real 3))"))
    (:arg-type "okra-array4")
    (:before "(with-foreign-object (array 'okra-real 4)")
    (:overloaded-type "'(simple-vector 4)")
    (:return-arg "array")
    (:return-type '(":void" . "(array4 :pointer)"))
    (otherwise nil)))


(defun cl-array6 (type name)
  (declare (ignore name))
  (case type
    (:after (mkfstr "(vector (mem-aref array 'okra-real 0) "
                            "(mem-aref array 'okra-real 1)~%"
                "            (mem-aref array 'okra-real 2) "
                            "(mem-aref array 'okra-real 3)~%"
                "            (mem-aref array 'okra-real 4) "
                            "(mem-aref array 'okra-real 5))"))
    (:arg-type "okra-array6")
    (:before "(with-foreign-object (array 'okra-real 6)")
    (:overloaded-type "'(simple-vector 6)")
    (:return-arg "array")
    (:return-type '(":void" . "(array6 :pointer)"))
    (otherwise nil)))


(defun cl-array9 (type name)
  (declare (ignore name))
  (case type
    (:after (mkfstr "(vector (mem-aref array 'okra-real  0) "
                            "(mem-aref array 'okra-real  1)~%"
                "            (mem-aref array 'okra-real  2) "
                            "(mem-aref array 'okra-real  3)~%"
                "            (mem-aref array 'okra-real  4) "
                            "(mem-aref array 'okra-real  5)~%"
                "            (mem-aref array 'okra-real  6) "
                            "(mem-aref array 'okra-real  7)~%"
                "            (mem-aref array 'okra-real  8))"))
    (:arg-type "okra-array9")
    (:before "(with-foreign-object (array 'okra-real 9)")
    (:overloaded-type "'(simple-vector 9)")
    (:return-arg "array")
    (:return-type '(":void" . "(array9 :pointer)"))
    (otherwise nil)))


(defun cl-array16 (type name)
  (declare (ignore name))
```

```
  (case type
    (:after (mkfstr "(vector (mem-aref array 'okra-real  0) "
                            "(mem-aref array 'okra-real  1)~%"
                "            (mem-aref array 'okra-real  2) "
                            "(mem-aref array 'okra-real  3)~%"
                "            (mem-aref array 'okra-real  4) "
                            "(mem-aref array 'okra-real  5)~%"
                "            (mem-aref array 'okra-real  6) "
                            "(mem-aref array 'okra-real  7)~%"
                "            (mem-aref array 'okra-real  8) "
                            "(mem-aref array 'okra-real  9)~%"
                "            (mem-aref array 'okra-real 10) "
                            "(mem-aref array 'okra-real 11)~%"
                "            (mem-aref array 'okra-real 12) "
                            "(mem-aref array 'okra-real 13)~%"
                "            (mem-aref array 'okra-real 14) "
                            "(mem-aref array 'okra-real 15))"))
    (:arg-type "okra-array16")
    (:before "(with-foreign-object (array 'okra-real 16)")
    (:overloaded-type "'(simple-vector 16)")
    (:return-arg "array")
    (:return-type '(":void" . "(array16 :pointer)"))
    (otherwise nil)))


(defun cl-boolean (type name)
  (declare (ignore name))
  (case type
    (:arg-type ":boolean")
    (:overloaded-type "'boolean")
    (:return-type ":boolean")
    (otherwise nil)))


(defun cl-light-types (type name)
  (declare (ignore name))
  (case type
    (:arg-type "light-types")
    (:return-type "light-types")
    (otherwise nil)))


(defun cl-long (type name)
  (declare (ignore name))
  (case type
    (:arg-type ":long")
    (:return-type ":long")
    (otherwise nil)))
```

```
(defun cl-okra-real (type name)
  (declare (ignore name))
  (case type
    (:arg-type "okra-real")
    (:overloaded-type "'real")
    (:return-type "okra-real")
    (otherwise nil)))


(defun cl-operation-type (type name)
  (declare (ignore name))
  (case type
    (:arg-type "operation-type")
    (:return-type "operation-type")
    (otherwise nil)))


(defun cl-plane (type name)
  (declare (ignore name))
  (case type
    (:arg-type "okra-array4")
    (:overloaded-type "'cffi:foreign-pointer")
    (:return-type ":pointer")
    (otherwise nil)))


(defun cl-pointer (type name)
  (declare (ignore name))
  (case type
    (:arg-type ":pointer")
    (:overloaded-type "'cffi:foreign-pointer")
    (:return-type ":pointer")
    (otherwise nil)))


;; XXX: should be moved to common-lisp-config.lisp
(defparameter *simple-types* '("uint16" "unsigned short"))

(defun cl-simple-type (type name)
  (case type
    (:arg-type (mkstr ":" (lisp-name name)))
    (:overloaded-type (cond ((member name *simple-types* :test #'equal)
                             "'integer")
                            (t nil)))
    (:return-type (mkstr ":" (lisp-name name)))
    (otherwise nil)))


(defun cl-string (type name)
  (declare (ignore name))
```

```
  (case type
    (:arg-type ":string")
    (:overloaded-type "'string")
    (:return-type ":string")
    (otherwise nil)))


(defun cl-size-t (type name)
  (declare (ignore name))
  (case type
    (:arg-type ":unsigned-int")
    (:overloaded-type "'integer")
    (:return-type ":unsigned-int")
    (otherwise nil)))
```

First try to return a CPP vector as a dynamically sized array. Leaks like a sieve in combination with c-render-system-list.

"(foreign-free rs-list)" possibly doesn't work and even if it does it'll only free the rs-list memory and not that which all of its pointers are pointing to.

```
(defun cl-render-system-list (type name)
  (declare (ignore name))
  (case type
    (:after
      (mkfstr "for i from 1 to (parse-integer (mem-aref rs-list :string 0))~%"
              "        collect (mem-aref rs-list :string i)~%"
              "        finally (foreign-free rs-list)"))
    (:before "(loop with rs-list =")  ; should be :before-call like in cpp
    (:return-type ":pointer")
    (otherwise nil)))


(defun cl-unsigned-int (type name)
  (declare (ignore name))
  (case type
    (:arg-type ":unsigned-int")
    (:overloaded-type "'integer")
    (:return-type ":unsigned-int")
    (otherwise nil)))


(defun cl-void (type name)
  (declare (ignore name))
  (case type
    (:arg-type "")
    (:return-type ":void")
    (otherwise nil)))
```