Using C++ from Lisp using smoke

David Lichteblau

December 15, 2009
How not to interface with C++

- FFI usually for C, not C++:
  - extern C { ... } wrapper code not satisfactory
How to interface with C++

- Want to use (almost) any class, call (almost) any method
- Want useful introspection for the bindings
- Want to make “subclasses”, i.e. override methods in Lisp
Smoke

- Part of the KDE project (kdebindings)
- Portable C++ header parser written in C++
- Generates C++ glue code — only minimal extern C {}
- msvc and gcc (Window, Linux, Mac, ...)
- Focus: Libraries from the Qt and KDE world

Basis of Qt/KDE bindings for Ruby, C#, Common Lisp. (Also, PHP, alternative Python bindings, QtScript stuff, ... )
class x_QPushButton : public QPushButton {
    SmokeBinding* _binding;

public:
    // provide every constructor
    void x_0(Smoke::Stack x) { _binding = (SmokeBinding*)x[1].s_class; }
    ...

    // override every method
    virtual void actionEvent(QActionEvent* x1) { ...
        if (this->_binding->callMethod(22074, (void*)this, x)) return;
        this->QWidget::actionEvent(x1);
    }
    ...

    // destructor
    ~x_QPushButton() { this->_binding->deleted(374, (void*)this); }
};

Allows the user-provided _binding object to intercept any method call for instances of x_QPushButton, i.e. for an object that it instantiated itself.
void xcall_QPushButton(Smoke::Index xi, void *obj, Smoke::Stack args) {
    x_QPushButton *xself = (x_QPushButton*)obj;
    switch(xi) {
        case 0: xself->x_0(args); break;
        ...
        case 3: x_QPushButton::x_3(args); break;
        ...
        case 28: x_QPushButton::x_28(args); break;
        ...
        case 34: delete (QPushButton*)xself; break;
    }
}

- Allows callers to call any method based on a single function pointer to xcall_QPushButton.
- Name mangling not an issue, because function pointer provided by introspection.
- Can use this for QPushButtons that aren't x_QPushButtons.
Smoke data introspection

Metadata available in each Library libsmokeqt.so, libsmokeqtw.webkit.so, ...:

class Smoke {
    struct Class {
        const char *className; // Name of the class
        ...
        ClassFn classFn; // Calls any method in the class
    }
    struct Method { ... }
    struct Type { ... }

    /* Tables of stuff: */
    
    Class *classes;
    Index numClasses;

    Method *methods;
    Index numMethods;

    Type *types;
    Index numTypes;
}

Tables sorted by name for binary searchability.
(Once found, cache the index.)
C++ from Lisp

- CommonQt (David Lichteblau), released at ILC 2009
- cl-smoke (Tobias Rautenkranz), released a few days later

Take your pick:

- CommonQt: No CLOS, C++ names, no load time overhead.
- cl-smoke: Fancy CLOS stuff, Lisp symbols, load time overhead.

cl-smoke probably way better at this point, but not the technical direction that I’m interested in...
C++ from Lisp

- CommonQt (David Lichteblau), released at ILC 2009
- cl-smoke (Tobias Rautenkranz), released a few days later

Take your pick:

- CommonQt: No CLOS, C++ names, no load time overhead.
- cl-smoke: Fancy CLOS stuff, Lisp symbols, load time overhead.

cl-smoke probably way better at this point, but not the technical direction that I’m interested in...
QT> (qapropos "qpushbutton")
Class QPushButton
Method QPushButton::QPushButton [14720]
Method QPushButton::QPushButton [14721]
Method QPushButton::QPushButton [14722]
Method QPushButton::QPushButton [14742]
Method QPushButton::QPushButton [14743]
Method QPushButton::QPushButton [14744]
Method QPushButton::"QPushButton" [14746]
NIL
QT> (find-qclass "QPushButton")
23936
QT> (qdescribe 23936)
23936 <374,0,0> is a smoke class

    name: QPushButton
    flags: #x5 (VIRTUAL, CONSTRUCTOR)

...
C++ from Lisp

QT> (qdescribe 23936)
23936 <374,0,0> is a smoke class

name: QPushButton
flags: #x5 (VIRTUAL, CONSTRUCTOR)

Superclasses:
  QAbstractButton
  QWidget
  QObject
  QPaintDevice

Methods:
  QPushButton#$ QPushButton::QPushButton [14744]
  QPushButton#$# QPushButton::QPushButton [14722]
  QPushButton$ QPushButton::QPushButton [14743]
  QPushButton$# QPushButton::QPushButton [14721]
  autoDefault QPushButton::autoDefault [14725]
  event# QPushButton::event [14734]
  focusInEvent# QPushButton::focusInEvent [14737]
  focusOutEvent# QPushButton::focusOutEvent [14738]
  initStyleOption# QPushButton::initStyleOption [14739]
  isDefault QPushButton::isDefault [14727]
  isFlat QPushButton::isFlat [14732]
  keyPressEvent# QPushButton::keyPressEvent [14736]
  menu QPushButton::menu [14730]
  metaObject QPushButton::metaObject [14713]
  minimumSizeHint QPushButton::minimumSizeHint [14724]
  paintEvent# QPushButton::paintEvent [14735]
  qt_metacall$$? QPushButton::qt_metacall [14719]
  qt_metacast$ QPushButton::qt_metacast [14714]
  setAutoDefault$ QPushButton::setAutoDefault [14726]
  setDefault$ QPushButton::setDefault [14728]
  setFlat$ QPushButton::setFlat [14731]
  setMenu# QPushButton::setMenu [14729]

Use (QDESCRIBE "QPushButton" T) to see inherited methods.

Properties:
  bool autoDefault
  bool default
  bool flat

Use (QDESCRIBE "QPushButton" T) to see inherited properties.

NIL

QT>
We use the following encoding scheme to represent references into meta data as a 22 bit integer:

```
000000000000001000100 = (class number 1 in the second module)
```

<table>
<thead>
<tr>
<th>16 bit index</th>
<th>4 bit module index</th>
<th>2 bit type</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Properties:

- no CLOS object caching, no memory overhead
- can just compare references using EQL
- fits into a fixnum
- Index ordering within a module and type is preserved, so that binary search in the tables works for references as well as indexes.