Using Mozart Functors

What is a functor

The source code must be submitted as an Oz functor. An Oz functor is equivalent to a module in other languages. For a complete description, look at the Application Development section in the Mozart Documentation. It can be found on the following url: `http://www.mozart-oz.org/documentation/apptut/node3.html#chapter.development`

The use of functors makes the execution of full programs simpler. Here is an example of a functor using the **Hello World** code from the QTk document.

```
functor
import
   QTk at 'x-oz://system/wp/QTk.ozf'
   System
   Application
define
   Show = System.show
   Desc = td(button(text:"Show"
                    action:proc {$}
                              {Show 'Hello World'}
                           end)
             button(text:"Close"
                    action:proc {$}
                             {Application.exit 0}
                           end))
in
   {{QTk.build Desc} show}
end
```

All top-level definitions following the `define` are visible throughout the functor body, from **`in`** to **`end`**. Do not write **`declare`** inside a functor. When a functor is imported into another functor it appears as a record with fields. For instance, to access the procedure `Show` from the `System` module, we must write `System.show`, because we are accessing the field `show` from the `System` record.

Compiling a functor

There are two ways of compiling a functor. If the functor is meant for being used by other functors, then, compile it as follows:

```
ozc -c foo.oz
```

That will generate the file **`foo.ozf`** which can be imported by other functors. You can still run this functor with the following command:

```
ozengine foo.ozf
```

But a nicer way to compile executable functors is by given the `-x` option to the oz compiler. Then, the execution can be done as any other unix executable file.

```
ozc -x foo.oz
./foo
```

For submission, send us the source code of the functor, **NOT** the compiled file.