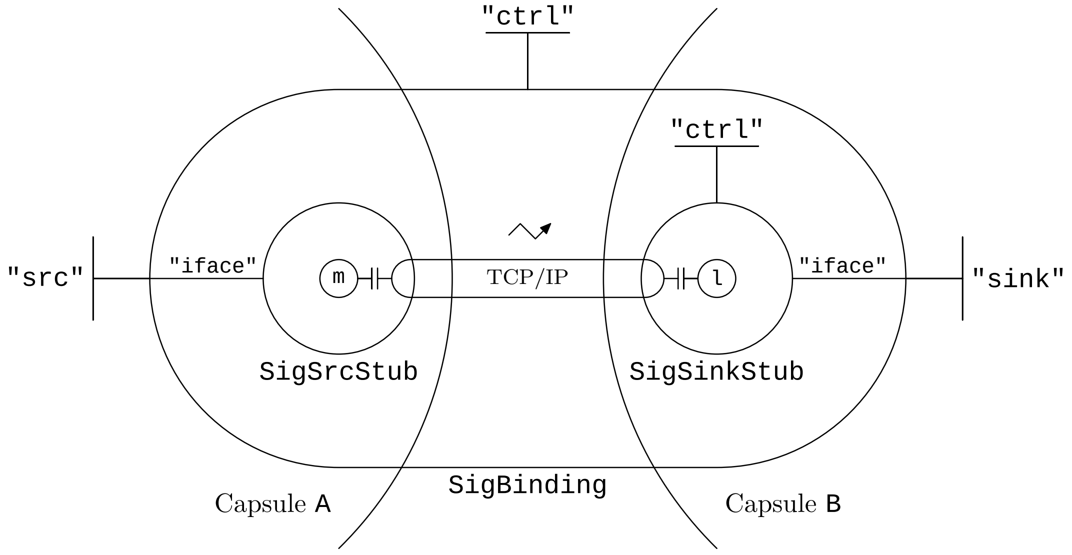
R"""Remote signal bindings

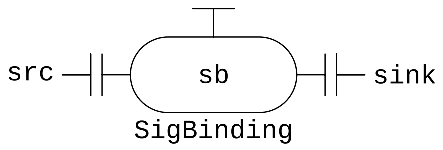
Author : Anders Andersen
Created On : Thu Aug 27 09:55:32 1998
Last Modified By: Anders Andersen
Last Modified On: Fri Apr 7 17:09:57 2000
Status : Unknown, Use with caution!

This module implements the `SigBinding` class and the `sigBind` function. The `SigBinding` class implements a signal binding that forwards signals from a signal source to a signal sink. The signals are sent using TCP/IP.



The figure above shows a signal binding of the `SigBinding` class with the signal source in capsule `A` and the signal sink in capsule `B`. The `sigBind` function can be used to create a signal binding between a given signal source and a given signal sink. The example below creates a signal binding `sb` between a signal source in capsule `A` (represented by the signal source interface reference `src`) and a signal sink in capsule `B` (represented by the signal sink interface reference `sink`):

```
sb = sigBind(src,sink)
```



Note that `sb` returned from the `sigBind` function is *not* an instance of the `SigBinding` class but a reference to the registered signal binding component in the local capsule (the `sigBind` function automatically registers the signal binding in the local capsule).
"""

```
from lbind import *

from component import *

from msg import *

from opbind import *

import capsule
```

```


def _emptyMethod(*args, **kw):
    R"""An empty method

    An empty method accepting any arguments and always returning None. Used as a replacement for a non
    exsisting event method in the signal interface.

    """
    return None


class SigSrcStub(Component):
    R"""A signal source stub

    A stub for the source side of a signal binding.

    """

    def __init__(self, node="", port=0):
        self.msg = EventSrc(node, port)
        Component.__init__(self, {"iface": SigSinkIRef(self)}, self)

    def event(self):
        self.msg.sendreq({"op": "event"})

class SigSinkStub(Stub):
    R"""A signal sink stub

    A stub for the sink side of a signal binding.

    """

    def __init__(self, port=0, cport=0):
        self.serving = 0
        self.listen = EventSink(gethostname(), port)
        self.ctrl = Msg(gethostname(), cport, 1)
        self.reply = 0
        Component.__init__(
            self,
            {"iface": SigSrcIRef(self),
             "ctrl": IRef(self, ["serve", "servethread", "stopserve"], [])},
            self)

    def __del__(self):
        del self.listen
        del self.ctrl


class SigBinding(Composite):

    def __init__(self, srccaps, sinkcaps, serve="servethread"):

        # Capsules
        self.srccaps = srccaps
        self.sinkcaps = sinkcaps

        # Fetch communication ports
        sigport = self.sinkcaps.newPort("sink sig binding")
        ctrlport = self.sinkcaps.newPort("sink sig binding ctrl")

        # Create stubs
        self.sinkstub = self.sinkcaps.mkComponent(
            SigSinkStub, (sigport, ctrlport), {})
        self.srcstub = self.srccaps.mkComponent(
            SigSrcStub, (self.sinkcaps.message.node, sigport), {})

        # Initialize the componet
        Component.__init__(
            self,
            {"src": self.srccaps.getIRef(self.srcstub, "iface"),
             "sink": self.sinkcaps.getIRef(self.sinkstub, "iface"),
             "ctrl": IRef(self, ["servethread", "serve", "stopserve"], [])},
            {"comps": [self.srcstub, self.sinkstub], "ifaces": {}, "edges": []})

        # Start server threads
        if serve in ["servethread", "serve"]:
            self.__serve__(serve)

    def __serve__(self, serve):
        self.sinkcaps.announceMethod(
            self.sinkstub, "ctrl", serve, (), {})

    def serve(self):
        self.__serve__("serve")

    def servethread(self):
        self.__serve__("servethread")

    def stopserve(self):
        debug("SigBinding stopserve")
        self.sinkcaps.announceMethod(
            self.sinkstub, "ctrl", "stopserve", (), {})

def sigBind(src, sink, serve="servethread"):

    # Get access to both capsules
    if type(src.__local__["object"]) is DictType:
        if src.__local__["object"]["capsule"] == capsule.local:
            srccaps = capsule.local
        else:
            srccaps = capsule.CapsuleProxy(
                src.__local__["object"]["capsule"].message.node,
                src.__local__["object"]["capsule"].message.port)
    else:
        srccaps = capsule.local
    if type(sink.__local__["object"]) is DictType:
        if sink.__local__["object"]["capsule"] == capsule.local:
            sinkcaps = capsule.local
        else:
            sinkcaps = capsule.CapsuleProxy(
                sink.__local__["object"]["capsule"].message.node,
                sink.__local__["object"]["capsule"].message.port)
    else:
        sinkcaps = capsule.local

    # Create and register binding
    sb = capsule.local.registerComponent(SigBinding(srccaps, sinkcaps, serve))

    # Bind it to the sig irefs
    localBind(src, capsule.local.getIRef(sb, "src"))
    localBind(capsule.local.getIRef(sb, "sink"), sink)
    return sb
```