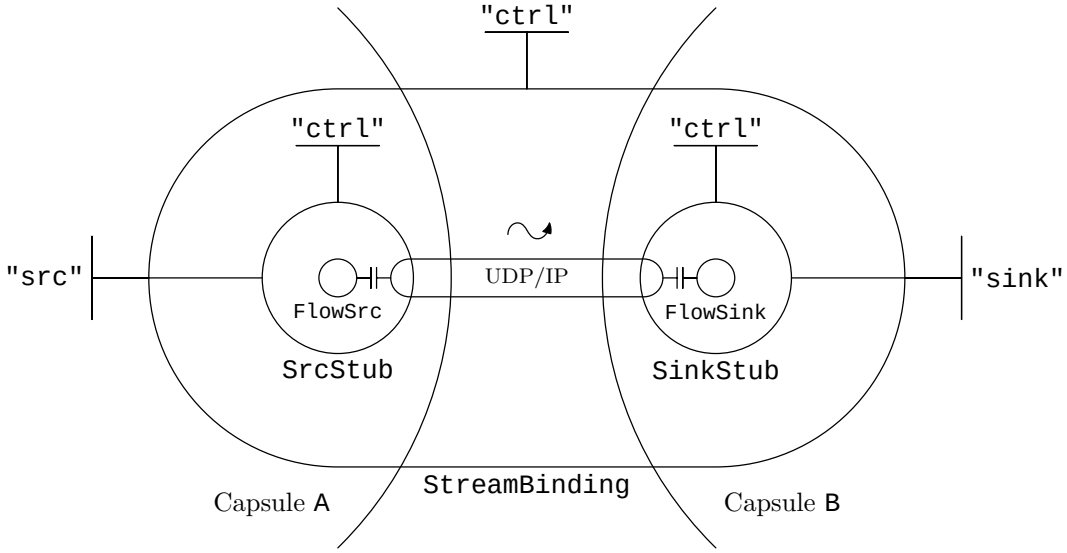
R"""Stream bindings [1]

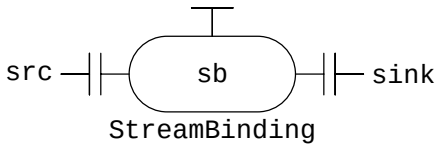
Author : Anders Andersen
Created On : Sat Dec 05 20:12:15 1998
Last Modified By: Anders Andersen
Last Modified On: Fri Apr 7 17:09:56 2000
Status : Unknown, Use with caution!

This module implements the `StreamBinding` class and the `streamBind` function. The `StreamBinding` class implements a simple stream binding that forwards every data frame from the source to the sink. This binding does not do any buffering or timing. Every frame 'pushed' from the source will be 'pushed' to the sink as soon as possible. Sequencing and packet loss are ignored. Every frame is sent as an UDP/IP package over connected UDP/IP.



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

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



This is (almost) equal to:

```
sb = StreamBinding(src_capsule,sink_capsule)
lb1 = localBind(src,sb.interfaces["src"])
lb2 = localBind(sb.interfaces["sink"],sink)
```

One important difference is that `sb` returned from the `streamBind` function is *not* an instance of the `StreamBinding` class but a reference to the registered stream binding component in the local capsule (the `streamBind` function automatically registers the stream binding in the local capsule).

"""

```
# We need to check the type of some attributes
from types import *

# For low level communication
from socket import *

# Listen on different connections at the same time
import select

# Need sys to access information about an exception
import sys

# Misc values for the Open-ORB core
from misc import *

# Local bindings (and interface references)
from lbind import *

# Components
from component import *

# Composite components
from composite import *

# Use the message objects
from msg import *

# Get the capsule
import capsule


class SrcStub(Component):
    R"""A source stub

    A source stub in a stream binding.
    """

    def __init__(self, node="", port=0, cport=0):
        R"""Initialize the stub

        Create the a flow and a message object (for out-going stream and control messages) and create
        forwarding methods (using the out-going message object) for the differen method calls. Also initialize
        the component part (interfaces) of the stub.
        """

        # Initialize the message objects
        self.flow = FlowSrc(node, port)
        self.ctrl = Msg(node, cport)

        # Insert forwarding methods
        fref = IRef(None, ["put"], [])
        fref.__local__["iobj"] = IObj()
        fref.__local__["iobj"].__dict__["put"] = self.flow.put
        cref = IRef(self, ["start", "stop"], [])

        # Initialize the stub as a component
        Component.__init__(self, {"src": fref, "ctrl": cref}, self)
```

```
    def __del__(self):
        try:
            self.stop()
        except:
            pass
        del self.flow
        del self.ctrl

    def start(self):
        self.ctrl.announce({"op": "start"})
        self.flow.start()

    def stop(self):
        sefl.flow.stop()
        self.ctrl.announce({"op": "stop"})


class SinkStub(Component):

    def __init__(self, port=0, cport=0, serve="servethread"):
        self.serving = 0
        self.flow = FlowSink(port)
        self.ctrl = Msg("", cport, 1)
        fref = IRef(None, [], ["put"])
        cref = IRef(
            self, ["serve", "servethread", "start", "stop", "stopserve"], [])
        Component.__init__(self, {"sink": fref, "ctrl": cref}, self)
        if serve in ["serve", "servethread"]:
            self.__serve__(serve)

    def __del__(self):
        self.stopserve()
        del self.flow
        del self.ctrl

    def __serveloop__(self):

        # Start serving
        debug("SinkStub serve: ready (%d, %d)" % (self.flow.port,
                                                  self.ctrl.port))
        while 1:

            # Wait for a request (either server or control)
            inreqlist, o, e = select.select(
                [self.flow.flowsocket, self.ctrl.listensocket], [], [])

            # Traverse the requests
            for inreq in inreqlist:
                if inreq == self.ctrl.listensocket:
                    connection, requests = self.ctrl.recvreq()
                    for req in requests:
                        debug("SinkStub ctrl request: %s" % (`req`,))
                        if req["op"] == "start":
                            debug("SinkStub start: %d (%d)" % (
                                self.flow.port, self.ctrl.port))
                            self.start()
                        elif req["op"] == "stop":
                            debug("SinkStub stop: %d (%d)" % (
```

```
                            self.flow.port, self.ctrl.port))
                        self.stop()
                    elif req["op"] == "stopserve":
                        debug("SinkStub stopserve: %d (%d)" % (
                            self.flow.port, self.ctrl.port))
                        self.serving = 0
                        self.stop()
                        return
                    continue

                # Forward put
                dataset = self.flow.get()
                for data in dataset:
                    self.interfaces["sink"].put(data)

    def __serve__(self, serve="servethread"):
        if not self.serving:
            self.serving = 1
            if serve == "servethread":
                import thread
                thread.start_new_thread(self.__serveloop__, (), {})
            else:
                self.__serveloop__()

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

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

    def start(self):
        self.flow.start()

    def stop(self):
        self.flow.stop()

    def stopserve(self):
        if self.serving:
            self.ctrl.announce({"op": "stopserve"})


class StreamBinding(Composite):

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

        # Fetch communication ports
        self.srccaps=srccaps
        self.sinkcaps=sinkcaps
        port = self.sinkcaps.newPort("stream binding")
        cport = self.sinkcaps.newPort("stream binding ctrl")

        # Create stubs
        self.srcstub = self.srccaps.mkComponent(
            SrcStub, (self.sinkcaps.message.node, port, cport), {})
        self.sinkstub = self.sinkcaps.mkComponent(
            SinkStub, (port, cport, serve), {})

        # Initialize component
```

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

        # Start flow
        self.start()

    def __sinkctrl__(self, method):
        self.sinkcaps.announceMethod(self.sinkstub, "ctrl", method, (), {})

    def __srcctrl__(self, method):
        self.srccaps.announceMethod(self.srcstub, "ctrl", method, (), {})

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

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

    def start(self):
        self.__srcctrl__("start")

    def stop(self):
        self.__srcctrl__("stop")

    def stopserve(self):
        self.__sinkctrl__("stopserve")


def streamBind(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 bind stream binding
    rsb = capsule.local.registerComponent(
        StreamBinding(srccaps, sinkcaps, serve))
    localBind(src, capsule.local.getIRef(rsb, "src"))
```

```
    localBind(capsule.local.getIRef(rsb, "sink"), sink)
    return rsb
```