```
// -*- C++ -*-
/*!
 * @file  Pipe.h for OpenRTM-aist-0.4.2
 * @brief Pipe component
 * @date  $Date$
 *
 * $Id$
 */
#ifndef PIPE_H
#define PIPE_H

#include <rtm/idl/BasicDataTypeSkel.h>
#include <rtm/Manager.h>
#include <rtm/DataFlowComponentBase.h>
#include <rtm/CorbaPort.h>
#include <rtm/DataInPort.h>
#include <rtm/DataOutPort.h>

// Service implementation headers
// <rtc-template block="service_impl_h">

// </rtc-template>

// Service Consumer stub headers
// <rtc-template block="consumer_stub_h">

// </rtc-template>

using namespace RTC;

template <class DataType, template <class DataType> class BufferT
class DirectInOut
  : public RTC::OnWrite<DataType>
{
  OutPort<DataType, BufferType>& m_out;
public:
  DirectInOut(OutPort<DataType, BufferType>& out) : m_out(out) {}
  virtual void operator()(const DataType& value)
  {
    std::cout << "callback DirectInOut called" << std::endl;
```

```
// -*- C++ -*-
/*!
 * @file  Pipe.h for OpenRTM-aist-1.0.0
 * @brief Pipe component
 * @date  $Date$
 *
 * $Id$
 */
#ifndef PIPE_H
#define PIPE_H

#include <rtm/idl/BasicDataTypeSkel.h>
#include <rtm/Manager.h>
#include <rtm/DataFlowComponentBase.h>
#include <rtm/CorbaPort.h>
#include <rtm/DataInPort.h>
#include <rtm/DataOutPort.h>

// Service implementation headers
// <rtc-template block="service_impl_h">

// </rtc-template>

// Service Consumer stub headers
// <rtc-template block="consumer_stub_h">

// </rtc-template>

using namespace RTC;

template <class DataType>
class DirectInOut
  : public ConnectorDataListenerT<DataType>
{
  OutPort<DataType>& m_out;
public:
  DirectInOut(OutPort<DataType>& out) : m_out(out) {}
  virtual void operator()(const ConnectorInfo& info,
                          const DataType& value)
  {
    std::cout << "callback DirectInOut called" << std::endl;
```

```
    m_out.write(value);
  }
};


class Pipe  : public RTC::DataFlowComponentBase
{
 public:
  Pipe(RTC::Manager* manager);
  ~Pipe();

  // The initialize action (on CREATED->ALIVE transition)
  // formaer rtc_init_entry()
 virtual RTC::ReturnCode_t onInitialize();

  // The finalize action (on ALIVE->END transition)
  // formaer rtc_exiting_entry()
  // virtual RTC::ReturnCode_t onFinalize();

  // The startup action when ExecutionContext startup
  // former rtc_starting_entry()
  // virtual RTC::ReturnCode_t onStartup(RTC::UniqueId ec_id);

  // The shutdown action when ExecutionContext stop
  // former rtc_stopping_entry()
  // virtual RTC::ReturnCode_t onShutdown(RTC::UniqueId ec_id);

  // The activated action (Active state entry action)
  // former rtc_active_entry()
  // virtual RTC::ReturnCode_t onActivated(RTC::UniqueId ec_id);

  // The deactivated action (Active state exit action)
  // former rtc_active_exit()
  // virtual RTC::ReturnCode_t onDeactivated(RTC::UniqueId ec_id)

  // The execution action that is invoked periodically
  // former rtc_active_do()
  // virtual RTC::ReturnCode_t onExecute(RTC::UniqueId ec_id);

  // The aborting action when main logic error occurred.
  // former rtc_aborting_entry()
```

```
    m_out.write(const_cast<DataType&>(value));
  }
};


class Pipe  : public RTC::DataFlowComponentBase
{
 public:
  Pipe(RTC::Manager* manager);
  ~Pipe();

  // The initialize action (on CREATED->ALIVE transition)
  // formaer rtc_init_entry()
 virtual RTC::ReturnCode_t onInitialize();

  // The finalize action (on ALIVE->END transition)
  // formaer rtc_exiting_entry()
  // virtual RTC::ReturnCode_t onFinalize();

  // The startup action when ExecutionContext startup
  // former rtc_starting_entry()
  // virtual RTC::ReturnCode_t onStartup(RTC::UniqueId ec_id);

  // The shutdown action when ExecutionContext stop
  // former rtc_stopping_entry()
  // virtual RTC::ReturnCode_t onShutdown(RTC::UniqueId ec_id);

  // The activated action (Active state entry action)
  // former rtc_active_entry()
  // virtual RTC::ReturnCode_t onActivated(RTC::UniqueId ec_id);

  // The deactivated action (Active state exit action)
  // former rtc_active_exit()
  // virtual RTC::ReturnCode_t onDeactivated(RTC::UniqueId ec_id)

  // The execution action that is invoked periodically
  // former rtc_active_do()
  // virtual RTC::ReturnCode_t onExecute(RTC::UniqueId ec_id);

  // The aborting action when main logic error occurred.
  // former rtc_aborting_entry()
```

```
 // virtual RTC::ReturnCode_t onAborting(RTC::UniqueId ec_id);

 // The error action in ERROR state
 // former rtc_error_do()
 // virtual RTC::ReturnCode_t onError(RTC::UniqueId ec_id);

 // The reset action that is invoked resetting
 // This is same but different the former rtc_init_entry()
 // virtual RTC::ReturnCode_t onReset(RTC::UniqueId ec_id);

 // The state update action that is invoked after onExecute() ac
 // no corresponding operation exists in OpenRTm-aist-0.2.0
 // virtual RTC::ReturnCode_t onStateUpdate(RTC::UniqueId ec_id)

 // The action that is invoked when execution context's rate is
 // no corresponding operation exists in OpenRTm-aist-0.2.0
 // virtual RTC::ReturnCode_t onRateChanged(RTC::UniqueId ec_id)


protected:
 // Configuration variable declaration
 // <rtc-template block="config_declare">

 // </rtc-template>

 // DataInPort declaration
 // <rtc-template block="inport_declare">
 TimedLong m_in;
 InPort<TimedLong> m_inIn;

 // </rtc-template>

 // DataOutPort declaration
 // <rtc-template block="outport_declare">
 TimedLong m_out;
 OutPort<TimedLong, NullBuffer> m_outOut;

 // </rtc-template>

 // CORBA Port declaration
 // <rtc-template block="corbaport_declare">
```

```
 // virtual RTC::ReturnCode_t onAborting(RTC::UniqueId ec_id);

 // The error action in ERROR state
 // former rtc_error_do()
 // virtual RTC::ReturnCode_t onError(RTC::UniqueId ec_id);

 // The reset action that is invoked resetting
 // This is same but different the former rtc_init_entry()
 // virtual RTC::ReturnCode_t onReset(RTC::UniqueId ec_id);

 // The state update action that is invoked after onExecute() ac
 // no corresponding operation exists in OpenRTm-aist-0.2.0
 // virtual RTC::ReturnCode_t onStateUpdate(RTC::UniqueId ec_id)

 // The action that is invoked when execution context's rate is
 // no corresponding operation exists in OpenRTm-aist-0.2.0
 // virtual RTC::ReturnCode_t onRateChanged(RTC::UniqueId ec_id)


protected:
 // Configuration variable declaration
 // <rtc-template block="config_declare">

 // </rtc-template>

 // DataInPort declaration
 // <rtc-template block="inport_declare">
 TimedLong m_in;
 InPort<TimedLong> m_inIn;

 // </rtc-template>

 // DataOutPort declaration
 // <rtc-template block="outport_declare">
 TimedLong m_out;
 OutPort<TimedLong> m_outOut;

 // </rtc-template>

 // CORBA Port declaration
 // <rtc-template block="corbaport_declare">
```

```
  // </rtc-template>

  // Service declaration
  // <rtc-template block="service_declare">

  // </rtc-template>

  // Consumer declaration
  // <rtc-template block="consumer_declare">

  // </rtc-template>

  DirectInOut<TimedLong, NullBuffer> m_inout;

 private:

};


extern "C"
{
  void PipeInit(RTC::Manager* manager);
};

#endif // PIPE_H
```

```
  // </rtc-template>

  // Service declaration
  // <rtc-template block="service_declare">

  // </rtc-template>

  // Consumer declaration
  // <rtc-template block="consumer_declare">

  // </rtc-template>

  DirectInOut<TimedLong> m_inout;

 private:

};


extern "C"
{
  void PipeInit(RTC::Manager* manager);
};

#endif // PIPE_H
```