```
// -*- C++ -*-
/*!
 * @file  Pipe.cpp for OpenRTM-aist-0.4.2
 * @brief Pipe component
 * $Date$
 *
 * $Id$
 */
#include "Pipe.h"

// Module specification
// <rtc-template block="module_spec">
static const char* pipe_spec[] =
  {
    "implementation_id", "Pipe",
    "type_name",         "Pipe",
    "description",       "Pipe component",
    "version",           "0.1",
    "vendor",            "AIST",
    "category",          "Generic",
    "activity_type",     "SPORADIC",
    "kind",              "DataFlowComponent",
    "max_instance",      "10",
    "language",          "C++",
    "lang_type",         "compile",
    // Configuration variables
    ""

  };
// </rtc-template>

Pipe::Pipe(RTC::Manager* manager)
    // <rtc-template block="initializer">
  : RTC::DataFlowComponentBase(manager),
    m_inIn("in", m_in),
    m_outOut("out", m_out),
    // </rtc-template>
    m_inout(m_outOut)
{
```

```
// -*- C++ -*-
/*!
 * @file  Pipe.cpp for OpenRTM-aist-1.0.0
 * @brief Pipe component
 * $Date$
 *
 * $Id$
 */
#include "Pipe.h"

// Module specification
// <rtc-template block="module_spec">
static const char* pipe_spec[] =
  {
    "implementation_id", "Pipe",
    "type_name",         "Pipe",
    "description",       "Pipe component",
    "version",           "0.1",
    "vendor",            "NAIST",
    "category",          "Generic",
    "activity_type",     "SPORADIC",
    "kind",              "DataFlowComponent",
    "max_instance",      "10",
    "language",          "C++",
    "lang_type",         "compile",
    // Configuration variables
    ""

  };
// </rtc-template>

Pipe::Pipe(RTC::Manager* manager)
    // <rtc-template block="initializer">
  : RTC::DataFlowComponentBase(manager),
    m_inIn("in", m_in),
    m_outOut("out", m_out),
    // </rtc-template>
    m_inout(m_outOut)
{
}

Pipe::~Pipe()
```

```
  // Registration: InPort/OutPort/Service
  // <rtc-template block="registration">
  // Set InPort buffers
  registerInPort("in", m_inIn);

  // Set OutPort buffer
  registerOutPort("out", m_outOut);

  // Set service provider to Ports

  // Set service consumers to Ports

  // Set CORBA Service Ports

  // </rtc-template>
  m_inIn.setOnWrite(&m_inout);
}

Pipe::~Pipe()
{
}


RTC::ReturnCode_t Pipe::onInitialize()
{
  // <rtc-template block="bind_config">
  // Bind variables and configuration variable

  // </rtc-template>
  return RTC::RTC_OK;
}


/*
RTC::ReturnCode_t Pipe::onFinalize()
```

```
{
}


RTC::ReturnCode_t Pipe::onInitialize()
{
  // Registration: InPort/OutPort/Service
  // <rtc-template block="registration">
  // Set InPort buffers
  addInPort("in", m_inIn);

  // Set OutPort buffer
  addOutPort("out", m_outOut);

  // Set service provider to Ports

  // Set service consumers to Ports

  // Set CORBA Service Ports

  // </rtc-template>
  m_inIn.addConnectorDataListener(ON_BUFFER_WRITE, &m_inout);
```

```
  // <rtc-template block="bind_config">
  // Bind variables and configuration variable

  // </rtc-template>
  return RTC::RTC_OK;
}


/*
RTC::ReturnCode_t Pipe::onFinalize()
```

```
{
  return RTC::RTC_OK;
}
*/
/*
RTC::ReturnCode_t Pipe::onStartup(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/
/*
RTC::ReturnCode_t Pipe::onShutdown(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/
/*
RTC::ReturnCode_t Pipe::onActivated(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/
/*
RTC::ReturnCode_t Pipe::onDeactivated(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/
/*
RTC::ReturnCode_t Pipe::onExecute(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/
/*
RTC::ReturnCode_t Pipe::onAborting(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/
/*
```


```
{
  return RTC::RTC_OK;
}
*/
/*
RTC::ReturnCode_t Pipe::onStartup(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/
/*
RTC::ReturnCode_t Pipe::onShutdown(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/
/*
RTC::ReturnCode_t Pipe::onActivated(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/
/*
RTC::ReturnCode_t Pipe::onDeactivated(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/
/*
RTC::ReturnCode_t Pipe::onExecute(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/
/*
RTC::ReturnCode_t Pipe::onAborting(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/
/*
```

```
RTC::ReturnCode_t Pipe::onError(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/
/*
RTC::ReturnCode_t Pipe::onReset(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/
/*
RTC::ReturnCode_t Pipe::onStateUpdate(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/
/*
RTC::ReturnCode_t Pipe::onRateChanged(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/


extern "C"
{

  void PipeInit(RTC::Manager* manager)
  {
    RTC::Properties profile(pipe_spec);
    manager->registerFactory(profile,
                             RTC::Create<Pipe>,
                             RTC::Delete<Pipe>);
  }

};
```

```
RTC::ReturnCode_t Pipe::onError(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/
/*
RTC::ReturnCode_t Pipe::onReset(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/
/*
RTC::ReturnCode_t Pipe::onStateUpdate(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/
/*
RTC::ReturnCode_t Pipe::onRateChanged(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/


extern "C"
{

  void PipeInit(RTC::Manager* manager)
  {
    coil::Properties profile(pipe_spec);
    manager->registerFactory(profile,
                             RTC::Create<Pipe>,
                             RTC::Delete<Pipe>);
  }

};
```