NORDUGRID

NORDUGRID-TECH-23

1/10/2008

ARC Data Manager Component (DMC)

Implementation Guide

Mattias Ellert[*]

[*]mattias.ellert@fysast.uu.se

Contents

Chapter 1

Introduction

The libarcdata2 library in the new Advanced Resource Connector (ARC) is based on libarcdata library in the old ARC. Unlike its predecessor the libarcdata2 library takes advantage of the plugin mechanism of the ARC Loader. The core libarcdata2 library does not introduce any additional external dependencies. Plugins for specific data access protocols can however have various external dependencies. This separation of external dependencies from the core library helps reduce the minimum set of requirements for ARC while allowing the support for additional access protocols requiring special dependencies to be installed by those who need it.

The plugins used by the libarcdata2 library are called Data Manager Components (DMCs). Each DMC should provide a class that inherits from the DMC base class. Like other ARC Loader plugins the DMCs should provide a descriptor that maps the name of the plugin to an function that creates an instance of the DMC. In order for the default DMC client configuration to automatically find the DMC the name of the DMC in the descriptor and the name of the loadable module containing the DMC should match. A DMC called X should reside in a module called libdmcX.

The constructor and destructor of a DMC should call the Register and Unregister method of respectively.

The most central class in the arcdata2 library is the DataPoint class. This class describes a source or destination of a file transfer.

A DMC should provide an iGetDataPoint method that returns a pointer to an instance of a specialisation of the DataPoint class when called with a URL. If the DMC does not support the provided URL a NULL pointer should be returned instead.

The libarcdata2 library contains a wrapper class for the DataPoint class called DataHandle. Calling the constructor of the DataHandle class with a URL will trigger the iGetDataPoint class of all registered DMCs to be called until a non-NULL pointer is returned. By dereferencing the DataHandle object the underlying DataPoint can be accessed.

Two partial specialisations of the DataPoint class are available in the libarcdata2 library. These are the DataPointIndex and DataPointDirect classes. The DataPointIndex class implements common functionality for DMCs that support file catalogues, while the DataPointDirect implement common functionality for DMCs that support direct file access. The specialisation of the DataPoint class used by a DMC should normally inherit from one of these classes rather than from the DataPoint base class directly.

Chapter 2

The DMC class

The DMC class describes a loadable module for arcdata2 library. A typical specialization of this class for a custom DMC plugin would look like this:

```
#ifndef __ARC_DMCCUSTOM_H__
#define __ARC_DMCCUSTOM_H__

#include <arc/data/DMC.h>

namespace Arc {

  class DMCCustom : public DMC {
   public:
    DMCCustom(Config *cfg);
    ~DMCCustom();
    static DMC* Instance(Config *cfg, ChainContext *ctx);
    DataPoint* iGetDataPoint(const URL& url);
   protected:
    static Logger logger;
  };

} // namespace Arc

#endif // __ARC_DMCCUSTOM_H__
```

The corresponding implementation should look like this:

```
#include <arc/Logger.h>
#include <arc/URL.h>
#include <arc/loader/DMCLoader.h>

#include "DMCCustom.h"
#include "DataPointCustom.h"

namespace Arc {

  Logger DMCCustom::logger(DMC::logger, "Custom");

  DMCCustom::DMCCustom(Config *cfg) : DMC(cfg) {
    Register(this);
  }

  DMCCustom::~DMCCustom() {
    Unregister(this);
  }

  DMC* DMCCustom::Instance(Arc::Config *cfg, Arc::ChainContext*) {
    return new DMCCustom(cfg);
  }

  DataPoint* DMCCustom::iGetDataPoint(const URL& url) {
    if (url.Protocol() != "custom") return NULL;
    return new DataPointCustom(url);
  }

} // namespace Arc

dmc_descriptors ARC_DMC_LOADER = {
  { "custom", 0, &Arc::DMCCustom::Instance },
  { NULL, 0, NULL }
};
```

The DataPointCustom class in the above example is the specialization of the DataPoint class for this DMC (see chapter 3).

The DMC can support more than one protocol in which case the iGetDataPoint should be changed accordingly.

Chapter 3

The DataPoint class

The DataPoint class is the most central class in the arcdata2 library. This is an abstract base class that must be specialized for each DMC. Two partial specialisations of the DataPoint class are available in the libarcdata2 library. These are the DataPointIndex and DataPointDirect classes. The DataPointIndex class implements common functionality for DMCs that support file catalogues, while the DataPointDirect implement common functionality for DMCs that support direct file access. The specialisation of the DataPoint class used by a DMC should normally inherit from one of these classes rather than from the DataPoint base class directly.

3.1 DataPointDirect

To create a DataPoint class for a direct access protocol, the following methods should be provided:

```
namespace Arc {

  class DataPointCustomDirect : public DataPointDirect {
   public:
    DataPointCustomDirect(const URL& url);
    virtual ~DataPointCustomDirect();
    virtual bool start_reading(DataBufferPar& buffer);
    virtual bool start_writing(DataBufferPar& buffer,
                               DataCallback *space_cb = NULL);
    virtual bool stop_reading(void);
    virtual bool stop_writing(void);
    virtual bool analyze(analyze_t& arg);
    virtual bool check(void);
    virtual bool remove(void);
    virtual bool list_files(std::list<FileInfo>& files, bool resolve = true);
  };

} // namespace Arc
```

3.2 DataPointIndex

To create a DataPoint class for an index protocol, the following methods should be provided:

```
namespace Arc {

  class DataPointCustom2 : public DataPointIndex {
   public:
    DataPointCustom2(const URL& url);
    ~DataPointCustom2() {};
    virtual bool meta_resolve(bool source);
    virtual bool meta_preregister(bool replication, bool force = false);
    virtual bool meta_postregister(bool replication);
    virtual bool meta_preunregister(bool replication);
    virtual bool meta_unregister(bool all);
    virtual bool list_files(std::list<FileInfo> &files, bool resolve = true);
  };

} // namespace Arc
```