Interface Specification for RRD Measurement Archive

Authors	Loukik Kudarimoti
Date	8-12-06
Current Version	1.1

Document Change Log

As SA3-WI15 Document			
Version number	**Date**	**Description of change**	**People**
1.0	04-11-06	First draft issued	Loukik Kudarimoti
1.1	08-12-06	Updated rncs according to latest information	Loukik Kudarimoti

Table of Contents

1. General Information

Service Name: RRD_Type_MA-1.1

Service Type: Measurement Archive

Version/release: 1.1 (for bundle release 1.1)

Service Description: This service provides the capability to read utilisation data (Bps or bps) stored in RRD files. More information about RRD files can be found at http://oss.oetiker.ch/rrdtool/. The service also provides the capability to write data into rrd files. However, because of the way in which rrd files require to be updated, certain restrictions apply. Please see functionality section below for more info.

The service requires the installation of rrdtool and rrdjtool. It is expected that the installer of this service has already installed rrdtool.

Contact Person(s): Roman Lapacz

Contact Information: romradz@rose.man.poznan.pl

2. Functionality – Handle-metadata

2.1. Introduction

The *Handle-metadata* functionality aims to allow users to perform searches on the metadata information stored within the installed service. With the help of such searches, the user can easily check what data can be retrieved via the service.

The metadata information accessible via the service makes use of the metadata configuration information that the deployer/installer of the service has to provide while setting up the service. More information about the metadata configuration file can be found in the document dedicated to describing this configuration file.

The metadata information retrieved by the user consists of 'metadata chains'. Each chain retrieved by the user includes a 'key'. The intention behind providing this 'key' is to enhance the performance of the server. If the user makes use of this key while trying to retrieve data, the key contains information on the location of the data which the service can use to quickly access the data. This efficiency directly benefits the user by reducing

the requests' turn-around time especially when the user is asking for data in bulk or is repeatedly asking for the same data.

This functionality is made available to the users with the help of *MetadataKeyRequest* and *MetadataKeyResponse* messages. Whenever the user makes a *MetadataKeyRequest*, the response from the service is always a *MetadataKeyResponse* (not considering underlying transport protocol errors). If the service encountered one or more problems, they are reported using error codes. These error codes are contained within the *MetadataKeyResponse* as well.

Number of request messages supported: 1

Number of possible error-free response message types: 1

2.2. Request Message – MetadataKeyRequest

The .rnc file for the request message can be found in Appendix 1. This .rnc file is based on NMWG v2 (base 2) xml protocol definitions.

At a high level, the request message consists of a metadata section, a data section and some parameters (these are based on NMWG v2 principles). The data section and the metadata section are linked together using *id*s in the metadata elements and *metadataIdRefs* in the data elements. The metadata section in a message can contain one or two metadata elements. If there are two metadata elements, the subject element of the second metadata element has to contain a reference to the id of the first metadata element. The data element in the data section will have to chain to the second metadata element (see examples).

For the request message, the metadata section can contain one or more metadata sections. The first metadata section should contain a subject which has interface elements or it can contain a key (which was previously obtained). The second metadata section, if present, contains 'filtering' parameters – parameters which specify the criteria for filtering the data.

If the first metadata elements does not contain a key already, then it can contain any or all of the following sections: nmwg:subject, nmwg:parameters, nmwg:eventType. The nmwg:subject section can contain any, all or none of the elements listed in the schema. If the subject section is the only section provided in the request, at least one of its child elements should be present. Otherwise, even though the schema will validate, in order to get an error free response, at least one of the elements should be present in the subject section. The nmwg:eventType section can contain any text but if this section is present in the request, the text for the element will be matched against what is stored in the metadata configuration files. The response will depend on whether any matches are found.

If the first metadata element contains a key, it should not contain nmwg:subject, nmwg:parameters or nmwg:eventType but the key element only. The key element contains parameters which should be seen as opaque.

The second metadata element, if present, contains select:subject which will reference to the first metadata element. This second metadata element will also contain select:parameters. Any or all of the parameters listed in the rnc have to be present. If none of these parameter elements are provided within the parameter block, the service will not fail but may assume defaults. The second metadata element cannot be present by itself as it has to reference the first metadata

The nmwg:data section in the request has to be empty but should reference the second metadata element, if present, or else the first metadata element. It should do this referencing with the help of the *metadataIdRef* attribute.

2.3. ResponseMessage – MetadataKeyResponse for MetadataKeyRequest

The *MetadataKeyResponse* message contains a request to the *MetadataKeyRequest* message depending on the processing of the request by the service. This response message will contain at least one metadata-data chain under all circumstances (apart from transport layer errors such as http 404, etc). Each chain will contain a metadata element and data element. There can be as many chains as there are matches for the given *MetadataKeyRequest* message. If an error was encountered, there is usually only one chain in the response and this chain will contain the error codes.

The metadata element will either contain subject information or a key or it will contain a result code event type. The subject information is contained within the subject element. The subject element will contain an interface element. This interface element can contain any or all of the elements seen in the .rnc depending on the configuration applied by the deployer. If the subject element is present, a parameters block can be present. This parameter element lists all the *eventTypes* supported by the service. Each such eventType will appear as a parameter within the parameters block.

The metadata element can also contain a key. This happens when a key was passed in the metadataKeyRequest. In such a case, although the intention is to pass the original key back, currently, a different key is passed back and this is the same key which will be present in the data section as well. From the client/user perspective, the contents of the key shouldn’t matter as the key should always be considered to be unique.

If a result code in the metadata element is passed back, it is contained in an eventType element directly in the metadata block. The data block will also contain a datum element (nmwgr namespace) and the datum element will provide more information about the error.

If no errors occurred and no error codes were passed back, the linked data block will contain a *key*. This key element contains a parameters block. The contents of the key block are opaque. The recipient of this key is not expected to understand it and should not in any case attempt to alter it. This key might be encrypted by the server in the future.

2.4. Examples

A simple *MetadataKeyRequest* is in the first example below

```
<?xml version='1.0' encoding='UTF-8'?>

<nmwg:message type="MetadataKeyRequest"
              id="mdrq1"

xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/"
              xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/"
              xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">

  <nmwg:parameters id="msgparam1">
    <nmwg:parameter name="authToken">PIONIER-Public</nmwg:parameter>
    <nmwg:parameter name="timeValue">1127250495</nmwg:parameter>
    <nmwg:parameter name="timeType">unix</nmwg:parameter>
  </nmwg:parameters>

  <nmwg:metadata id="meta1">
    <netutil:subject id="subj1">
      <nmwgt:interface>
        <nmwgt:ifAddress type="ipv4">212.191.224.106</nmwgt:ifAddress>
        <nmwgt:ifName>ge-2/1/0.102</nmwgt:ifName>
        <nmwgt:direction>out</nmwgt:direction>
      </nmwgt:interface>
    </netutil:subject>
    <nmwg:eventType>utilization</nmwg:eventType>
  </nmwg:metadata>

  <nmwg:data id="data1" metadataIdRef="meta1"/>

</nmwg:message>
```

A *MetadataKeyResponse* for the above request is in the example below:

```
<?xml version="1.0" encoding="UTF-8"?>

<nmwg:message xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/"
id="localhost.-77179f19:10b5b954cea:-7e4b">

  <nmwg:metadata id="meta2">
   <netutil:subject
xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/"
id="subj1">
    <nmwgt:interface
xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
```

```
      <nmwgt:hostName>test-hostName</nmwgt:hostName>
      <nmwgt:ifName>ge-2/1/0.102</nmwgt:ifName>
      <nmwgt:ifDescription>test descripyion</nmwgt:ifDescription>
      <nmwgt:ifAddress type="ipv4">212.191.224.106</nmwgt:ifAddress>
      <nmwgt:direction>out</nmwgt:direction>
      <nmwgt:authRealm>PIONIER-Public</nmwgt:authRealm>
      <nmwgt:capacity>1000BaseT</nmwgt:capacity>
     </nmwgt:interface>
    </netutil:subject>
    <nmwg:eventType>utilization</nmwg:eventType>
    </nmwg:metadata>

  <nmwg:data id="data2" metadataIdRef="meta2">
   <nmwg:key id="localhost.-77179f19:10b5b954cea:-7e4f">
    <nmwg:parameters id="localhost.-77179f19:10b5b954cea:-7e4e">
     <nmwg:parameter name="dataSource">ds0</nmwg:parameter>
     <nmwg:parameter name="file">/home/perfsonar/t320-poznan.ge-
2_1_0.102.rrd</nmwg:paramete
r>
    </nmwg:parameters>
   </nmwg:key>
  </nmwg:data>

</nmwg:message>
```

A *MetadataKeyRequest* with two metadata blocks in the request message is in the example below:

```
<?xml version='1.0' encoding='UTF-8'?>

<nmwg:message id="msg1"
              type="MetadataKeyRequest"

xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/"
              xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/"
              xmlns:select="http://ggf.org/ns/nmwg/ops/select/2.0/">

  <nmwg:parameters id="msgparam1">
    <nmwg:parameter name="authToken">PIONIER-Public</nmwg:parameter>
    <nmwg:parameter name="timeValue">1138699961</nmwg:parameter>
    <nmwg:parameter name="timeType">unix</nmwg:parameter>
  </nmwg:parameters>

  <nmwg:metadata id="meta1">
    <nmwg:key>
      <nmwg:parameters id="param1">
        <nmwg:parameter name="file">/home/perfsonar/t320-poznan.ge-
2_1_0.102.rrd</nmwg:parameter>
        <nmwg:parameter name="dataSource">ds0</nmwg:parameter>
      </nmwg:parameters>
    </nmwg:key>
  </nmwg:metadata>

  <nmwg:metadata id="meta2">
    <select:subject id="iusub2" metadataIdRef="meta1"/>
```

```
    <select:parameters id="param1">
      <nmwg:parameter name="startTime">1148370000</nmwg:parameter>
      <nmwg:parameter name="endTime">1148373000</nmwg:parameter>
      <nmwg:parameter
name="consolidationFunction">AVERAGE</nmwg:parameter>
      <nmwg:parameter name="resolution">60</nmwg:parameter>
    </select:parameters>
    <nmwg:eventType>select</nmwg:eventType>
  </nmwg:metadata>

  <nmwg:data id="data1" metadataIdRef="meta2"/>

</nmwg:message>
```

A *MetadataKeyResponse* message for example 3 above is shown in example 4 below:

```
<?xml version="1.0" encoding="UTF-8"?>
<nmwg:message xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/"
id="localhost.42e32c28:10b5c23e9cf:-7ecb">

  <nmwg:metadata id="m1">
      <nmwg:key id="k1">
         <nmwg:parameters id="param1">
            <nmwg:parameter name="dataSource">ds0</nmwg:parameter>
            <nmwg:parameter
name="consolidationFunction">AVERAGE</nmwg:parameter>
            <nmwg:parameter name="file">/home/perfsonar/t320-poznan.ge-
2_1_0.102.rrd</nmwg:parameter>
            <nmwg:parameter name="resolution">60</nmwg:parameter>
            <nmwg:parameter
name="startTime">1148370000</nmwg:parameter>
            <nmwg:parameter name="endTime">1148373000</nmwg:parameter>
         </nmwg:parameters>
      </nmwg:key>
  </nmwg:metadata>

  <nmwg:data id="localhost.42e32c28:10b5c23e9cf:-7ecc"
metadataIdRef="m1">
      <nmwg:key id="localhost.42e32c28:10b5c23e9cf:-7ed4">
         <nmwg:parameters id="param1">
            <nmwg:parameter name="dataSource">ds0</nmwg:parameter>
            <nmwg:parameter
name="consolidationFunction">AVERAGE</nmwg:parameter>
            <nmwg:parameter name="file">/home/perfsonar/t320-poznan.ge-
2_1_0.102.rrd</nmwg:parameter>
            <nmwg:parameter name="resolution">60</nmwg:parameter>
            <nmwg:parameter
name="startTime">1148370000</nmwg:parameter>
            <nmwg:parameter name="endTime">1148373000</nmwg:parameter>
         </nmwg:parameters>
      </nmwg:key>
  </nmwg:data>

</nmwg:message>
```

3. Functionality – Handle-data

3.1. Introduction

3.2. Request Message – SetupDataRequest

3.3. ResponseMessage – SetupDataResponse for setupDataRequest

3.4. RequestMessage – MeasurementArchiveStoreRequest

3.5. ResponseMessage – MeasurementArchiveStoreResponse for MeasurementArchiveStoreRequest

4. Functionality – Handle-LS-interaction

4.1. RequestMessage – LSRegisterRequest

4.2. ResponseMessage – LSRegisterResponse for LSRegisterRequest

4.3. RequestMessage – LSDeregisterRequest

4.4. ResponseMessage – LSDeregisterResponse

Appendix I

This appendix contains RNC (rnc) schema definitions for the handle-metadata functionality.

MetadataKeyRequest

```
namespace netutil="http://ggf.org/ns/nmwg/characteristic/utilisation/2.0/"
namespace nmwg="http://ggf.org/ns/nmwg/base/2.0/"
namespace nmwgt="http://ggf.org/ns/nmwg/topology/2.0/"
namespace select="http://ggf.org/ns/nmwg/ops/select/2.0/"

start = element nmwg:message { MessageContent }

MessageContent =
      Identifier? &
      MessageIdentifierRef? &
      Type &
      Parameters? &
      (
            Metadata+,
            Data
      )+

Identifier =
      attribute id { xsd:string }

MessageIdentifierRef =
      attribute messageIdRef { xsd:string }

Type =
      attribute type { “MetadataKeyRequest” }


Metadata =
      element nmwg:metadata {
            (
                  Identifier &
                  MetadataIdentifierRef? &
                  (
                    Subject|
                    FilterMetadataContent |
                    MetadataKeyContent
                  ) &
```

```
                EventType?
            )
        }

MetadataIdentifierRef =
        attribute metadataIdRef { xsd:string }


Subject =
        element netutil:subject { SubjectContent }

SubjectContent =
        (
            Identifier &
            MetadataIdentifierRef? &
            InterfaceContent?
        )


InterfaceContent =
        element nmwgt:interface
            {
                InterfaceAddress? &
                HostName? &
                Direction? &
                InterfaceDescription? &
                InterfaceName? &
                AuthRealm? &
                Capacity?
            }


InterfaceAddress =
        element nmwgt:ifAddress {
                attribute type{"ipv4" | "ipv6"},
                xsd:string
        }

HostName =
        element nmwgt:hostname {xsd:string}

Direction =
        element nmwgt:direction {"in" | "out"}

InterfaceDescription =
        element nmwgt:ifDescr {xsd:string}
```

```
InterfaceName =
      element nmwgt:ifName {xsd:string}

AuthRealm =
      element nmwgt:authRealm {xsd:string}

Capacity =
      element nmwgt:capacity {xsd:string}


FilterMetadataContent =

      element select:subject
      {
            attribute id {xsd:string},
            attribute metadataIdRef {xsd:string}
      },

      element select:parameters { ParametersContent }


Parameters =
      element nmwg:parameters { ParametersContent }

ParametersContent =
      Identifier &
      Parameter+

Parameter =
      element nmwg:parameter {
      attribute name { xsd:string } &
            (
                  attribute value { xsd:string } |
                  text
            )
      }


MetadataKeyContent =
         element nmwg:key
            {
                  attribute id {xsd:string}?,
                  Parameters
            }
```

```
EventType =
    element nmwg:eventType { xsd:string }


Data =
    element nmwg:data { Identifier & MetadataIdentifierRef }
```

MetadataKeyResponse for MetadataKeyRequest

```
namespace netutil="http://ggf.org/ns/nmwg/characteristic/utilisation/2.0/"
namespace nmwg="http://ggf.org/ns/nmwg/base/2.0/"
namespace nmwgt="http://ggf.org/ns/nmwg/topology/2.0/"
namespace nmwgr="http://ggf.org/ns/nmwg/result/2.0/"


start = element nmwg:message { MessageContent }


MessageContent =
    Identifier? &
    MessageIdentifierRef? &
    Type &
    Parameters? &
    (
        (Metadata|ResultCodeMetadata),
        Data
    )+

Identifier =
    attribute id { xsd:string }

MessageIdentifierRef =
    attribute messageIdRef { xsd:string }

Type =
    attribute type { “MetadataKeyResponse” }


Metadata =
    element nmwg:metadata {
        (   Identifier &
            MetadataIdentifierRef? &
            Subject? &
```

```
                Parameters?
            ) | Key

        }

MetadataIdentifierRef =
        attribute metadataIdRef { xsd:string }


Subject =
        element netutil:subject
        {
                Identifier &
                MetadataIdentifierRef? &
                InterfaceContent?
        }


InterfaceContent =
        element nmwgt:interface
                {
                        InterfaceAddress?,
                        HostName?,
                        Direction?,
                        InterfaceDescription?,
                        InterfaceName?,
                        AuthRealm?,
                        Capacity?
                }


InterfaceAddress =
        element nmwgt:ifAddress {
                        attribute type{"ipv4" | "ipv6"},
                        xsd:string
        }

HostName =
        element nmwgt:hostname {xsd:string}

Direction =
        element nmwgt:direction {"in" | "out"}

InterfaceDescription =
        element nmwgt:ifDescr {xsd:string}
```

```
InterfaceName =
    element nmwgt:ifName {xsd:string}

AuthRealm =
    element nmwgt:authRealm {xsd:string}

Capacity =
    element nmwgt:capacity {xsd:string}



Parameters =
    element nmwg:parameters { ParametersContent }

ParametersContent =
    Identifier &
    Parameter+

Parameter =
    element nmwg:parameter {
    attribute name { xsd:string } &
        (
            attribute value { xsd:string } |
            text
        )
    }

ResultCodeMetadata =
    element nmwg:metadata {
        attribute id {xsd:string},
        element nmwg:eventType {text}
    }


Data = (KeyData|ResultCodeData)

KeyData =
    element nmwg:data
    {
        Identifier &
        MetadataIdentifierRef &
        Key
    }

Key =
    element nmwg:key
```

```
        {
            attribute id {xsd:string}?,
            Parameters
        }

ResultCodeData =
        element nmwg:data {
            attribute id {xsd:string},
            attribute metadataIdRef {xsd:string},
            element nmwgr:datum { text }
        }
```

Appendix II

Request Message – SetupDataRequest

ResponseMessage – SetupDataResponse for setupDataRequest

RequestMessage – MeasurementArchiveStoreRequest

ResponseMessage – MeasurementArchiveStoreResponse for MeasurementArchiveStoreRequest

Appendix III

RequestMessage – LSRegisterRequest

ResponseMessage –LSRegisterResponse for LSRegisterRequest

RequestMessage – LSDeregisterRequest

ResponseMessage – LSDeregisterResponse