Resource Management Interface Specs
Release v. 3.0.4
18 JUL 2005

Scott Jackson
Brett Bode
David Jackson
Kevin Walker

Scalable Systems Software Resource Management and Accounting Protocol (SSSRMAP) Message Format

Status of this Memo

This is a specification defining an XML message format used between Scalable Systems Software components. It is intended that this specification will continue to evolve as these interfaces are implemented and thoroughly tested by time and experience.

Abstract

This document is a specification describing a message format for the interaction of resource management and accounting software components developed as part of the Scalable Systems Software Center. The SSSRMAP Message Format defines a request-response syntax supporting both functional and object-oriented messages. The protocol is specified in XML Schema Definition. The message elements defined in this specification are intended to be framed within the Envelope and Body elements defined in the SSSRMAP Wire Protocol specification document.

Table of Contents

1 Introduction

A major objective of the Scalable Systems Software [SSS] Center is to create a scalable and modular infrastructure for resource management and accounting on terascale clusters including resource scheduling, grid-scheduling, node daemon support, comprehensive usage accounting and user interfaces emphasizing portability to terascale vendor operating systems. Existing resource management and accounting components feature disparate APIs (Application Programming Interfaces) requiring various forms of application coding to interact with other components.

This document proposes a common message format expressed in an XML request-response syntax to be considered as the foundation of a standard for communications between and among resource management and accounting software components. In this document this standard is expressed in two levels of generality. The features of the core SSSRMAP protocol common to all resource management and accounting components in general are described in the main body of this document. The aspects of the syntax specific to individual components are described in component-specific binding documents.

2 Conventions Used in this Document

2.1 Keywords

The keywords “MUST”, “MUST NOT”, “REQUIIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC2119 [RFC2119].

2.2 XML Case Conventions

In order to enforce a consistent capitalization and naming convention across all SSSRMAP specifications “Upper Camel Case” (UCC) and “Lower Camel Case” (LCC) Capitalization styles shall be used. UCC style capitalizes the first character of each word and compounds the name. LCC style capitalizes the first character of each word except the first word. [XML_CONV][FED_XML]

1. SSSRMAP XML Schema and XML instance documents SHALL use the following conventions:
    - Element names SHALL be in UCC convention (example: <UpperCamelCaseElement/>.
    - Attribute names SHALL be in LCC convention (example: <UpperCamelCaseElement lowerCamelCaseAttribute=”Whatever”/>.
2. General rules for all names are:
    - Acronyms SHOULD be avoided, but in cases where they are used, the capitalization SHALL remain (example: XMLSignature).
    - Underscores (_), periods (.) and dashes (-) MUST NOT be used (example: use JobId instead of JOB.ID, Job_ID or job-id).

2.3 Schema Definitions

SSSRMAP Schema Definitions appear like this

In case of disagreement between the schema file and this specification, the schema file takes precedence.

3 Encoding

Encoding tells how a message is represented when exchanged. SSSRMAP data exchange messages SHALL be defined in terms of XML schema [XML_SCHEMA].

3.1 Schema Header and Namespaces

The header of the schema definition is as follows:

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

```
<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:sssrmap="http://www.scidac.org/ScalableSystems/SSSRMAP"
  targetNamespace="http://www.scidac.org/ScalableSystems/SSSRMAP"
  elementFormDefault="qualified">
```

3.2 Element Descriptions

The following subsections describe the elements that make up SSSRMAP messages. SSSRMAP messages are transmitted in the Body and Envelope elements as described in the SSSRMAP Wire Protocol specification [WIRE_PROTOCOL].

3.2.1 The *Request* Element

The *Request* element specifies an individual request. An object-oriented request will have at least one *Object* element while a functional request will not have one. Depending on context, the *Request* element MAY contain one or more *Get* elements or one or more *Set* elements and any number of *Where* elements. *Option*, *Data*, *File* or *Count* elements may also be included. If a component supports it, chunking may be requested where large response data is possible. Setting the chunking attribute to "True" requests that the server break a large response into multiple chunks (each with their own envelope) so they can be processed in separate pieces.

Only an *action* attribute is required. All other attributes are optional.

- *action* – specifies the action or function to be performed
- *actor* – The authenticated user sending the request
- *id* -- uniquely maps the request to the appropriate response
- *chunking* -- requests that segmentation be used for large response data if set to "True"
- *chunkSize* -- requests that the segmentation size be no larger than the specified amount

```
<complexType name="RequestType">
  <choice minOccurs="0" maxOccurs="unbounded">
    <element ref="sssrmap:Object" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="sssrmap:Option" minOccurs="0" maxOccurs="unbounded"/>
    <choice minOccurs="0" maxOccurs="1">
      <element ref="sssrmap:Get" minOccurs="1" maxOccurs="unbounded"/>
      <element ref="sssrmap:Set" minOccurs="1" maxOccurs="unbounded"/>
    </choice>
    <element ref="sssrmap:Where" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="sssrmap:Data" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="sssrmap:Count" minOccurs="0" maxOccurs="1"/>
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
  </choice>
  <attribute name="action" type="string" use="required"/>
```

```
  <attribute name="actor" type="string" use="required"/>
  <attribute name="id" type="string" use="optional"/>
  <attribute name="chunking" type="sssrmap:BoolType" use="optional"/>
  <attribute name="chunkSize" type="positiveInteger" use="optional"/>
</complexType>

<element name="Request" type="sssrmap:RequestType"/>
```

3.2.2 The *Object* Element

The *Object* element is used in an object-oriented request to specify the object receiving the action. It is possible to have multiple *Object* elements in a request if an implementation supports multi-object queries.

The object class name is specified as text content. All attributes are optional.

- *join* – the type of join to be performed with the preceding object
    - A *join* attribute of "Inner" specifies an inner join. This is the default.
    - A *join* attribute of "FullOuter" specifies a full outer join.
    - A *join* attribute of "LeftOuter" specifies a left outer join.
    - A *join* attribute of "RightOuter" specifies a right outer join.
    - A *join* attribute of "Cross" specifies a cross join.
    - A *join* attribute of "Union" specifies a union join.

```
<complexType name="ObjectType">
  <simpleContent>
    <extension base="string">
      <attribute name="join" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="Object" type="sssrmap:ObjectType"/>
```

3.2.3 The *Get* Element

The *Get* element is used to indicate the data fields to be returned in a query. *Get* is typically used within requests with *action*="query". Multiple *Get* elements cause the fields to be returned in the order specified. If no *Get* elements are specified, the query will return a default set of fields.

Only a *name* attribute is required. All other attributes are optional.

- *name* – the name of the data field to be returned. This MUST be of the form of a "Modified XPATH expression" as described in a later section.

- *op* – the operator to be used to aggregate or perform an operation on the returned values.
    - An *op* attribute of "Sort" specifies an ascending sort operation
    - An *op* attribute of "Tros" specifies a descending sort operation
    - An *op* attribute of "Sum" returns the sum (only valid for numeric values)
    - An *op* attribute of "Max" returns the maximum value
    - An *op* attribute of "Min" returns the minimum value
    - An *op* attribute of "Count" returns the number of values
    - An *op* attribute of "Average" returns the average of the values
    - An *op* attribute of "GroupBy" signifies that aggregates are grouped by this field
- *object* -- specifies the object for which you want the named attribute in a multi-object query.
- *units* – the units in which to return the value (if applicable)

```
<complexType name="GetType">
  <attribute name="name" type=”string” use="required"/>
  <attribute name="object" type="string" use="optional"/>
  <attribute name="op" type="sssrmap:GetOperatorType" use="optional"/>
  <attribute name="units" type=”string” use="optional"/>
</complexType>

<element name="Get" type="sssrmap:GetType"/>

<simpleType name="GetOperatorType">
  <restriction base="string">
    <enumeration value="Sort"/>
    <enumeration value="Tros"/>
    <enumeration value="Count"/>
    <enumeration value="Sum"/>
    <enumeration value="Max"/>
    <enumeration value="Min"/>
    <enumeration value="Average"/>
    <enumeration value="GroupBy"/>
  </restriction>
</simpleType>
```

3.2.4 The *Set* Element

The *Set* element is used to specify the object data fields to be assigned values. *Set* is typically used within requests with *action*=”Create” or *action*=”Modify”. The use of *Get* or *Set* elements within a request are mutually exclusive.

The assignment value (to which the field is being changed) is specified as the text content. A *Set* element without a value may be used as an assertion flag. Only the *name* attribute is required. All other attributes are optional.

- *name* – the name of the field being assigned a value. This MUST be of the form of a "Modified XPATH expression" as described in a later section.

- *op* – the operator to be used in assigning a new value to the name. If an *op* attribute is not specified and a value is specified, the specified value will be assigned to the named field ("assign").
    - An *op* attribute of "Assign" assigns value to the named field
    - An *op* attribute of "Inc" increments the named field by the value
    - An *op* attribute of "Dec" decrements the named field by the value
- *units* – the units corresponding to the value being set

```
<complexType name="SetType">
  <simpleContent>
    <extension base="string">
      <attribute name="name" type=”string” use="required"/>
      <attribute name="op" type="sssrmap:SetOperatorType" use="optional"/>
      <attribute name="units" type=”string” use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="Set" type="sssrmap:SetType"/>

<simpleType name="SetOperatorType">
  <restriction base="string">
    <enumeration value="Assign"/>
    <enumeration value="Inc"/>
    <enumeration value="Dec"/>
  </restriction>
</simpleType>
```

3.2.5 The *Where* Element

A *Request* element may contain one or more *Where* elements that specify the search conditions for which objects the action is to be performed on.

The condition value (against which the field is tested) is specified as the text content. A *Where* element without a value may be used as a truth test. Only the *name* attribute is required. All other attributes are optional.

- *name* – the name of the data field to be tested. This MUST be of the form of a "Modified XPATH expression" as described in a later section.
- *op* – the operator to be used to test the name against the value. If an *op* attribute is not specified and a value is specified, the field will be tested whether it is equal to the value ("EQ").
    - An *op* attribute of "EQ" specifies an equality comparison
    - An *op* attribute of "LT" specifies a "less than" comparison
    - An *op* attribute of "GT" specifies a "greater than" comparison
    - An *op* attribute of "LE" specifies a "less than or equal to" test
    - An *op* attribute of "GE" specifies a "greater than or equal to" test
    - An *op* attribute of "NE" specifies a "not equal to" test

    - An *op* attribute of “Match” specifies a regular expression matching comparison
- *conj* -- indicates whether this test is to be anded or ored with the immediately preceding where condition
    - A *conj* attribute of “And” specifies an “and” conjunction
    - A *conj* attribute of “Or” specifies an “or” condition
    - A *conj* attribute of “AndNot” specifies an “and not” conjunction
    - A *conj* attribute of “OrNot” specifies an “or not” condition
- *group* – indicates an increase or decrease of parentheses grouping depth
    - A positive number indicates the number of left parentheses to precede the condition, i.e. *group*=”2” represents “((condition”.
    - A negative number indicates the number of right parentheses to follow the condition, i.e. *group*=”-2” represents “condition))”.
- *object* -- specifies the object for the first operand in a multi-object query.
- *subject* -- specifies the object for the second operand in a multi-object query.
- *units* – indicates the units to be used in the value comparison

```
<complexType name="WhereType">
  <simpleContent>
    <extension base="string">
      <attribute name="name" type=”string” use=”required”/>
      <attribute name="op" type="sssrmap:OperatorType" use="optional"/>
      <attribute name="conj" type="sssrmap:ConjunctionType" use="optional"/>
      <attribute name="group" type="integer" use="optional"/>
      <attribute name="units" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="Where" type="sssrmap:WhereType"/>

<simpleType name="WhereOperatorType">
  <restriction base="string">
    <enumeration value="EQ"/>
    <enumeration value="GT"/>
    <enumeration value="LT"/>
    <enumeration value="GE"/>
    <enumeration value="LE"/>
    <enumeration value="NE"/>
    <enumeration value="Match"/>
  </restriction>
</simpleType>
```

3.2.6 The *Option* Element

The *Option* element is used to indicate processing options for the command. An option might be used to indicate that command usage or special formatting is desired, or that the command is to be invoked with particular options.

The option value is specified as the text content. An *Option* element without a value may be used as an assertion flag. Only the *name* attribute is required. All other attributes are optional.

- *name* – the name of the field being assigned a value
- *op* – the operator to be used to disassert the option
  - An *op* attribute of "Not" specifies that the option is not asserted
- *conj* -- indicates whether this test is to be anded or ored with the immediately preceding where condition
  - A *conj* attribute of "And" specifies an "and" conjunction
  - A *conj* attribute of "Or" specifies an "or" condition
  - A *conj* attribute of "AndNot" specifies an "and not" conjunction
  - A *conj* attribute of "OrNot" specifies an "or not" condition

```
<complexType name="OptionType">
  <simpleContent>
    <extension base="string">
      <attribute name="name" type=”string” use="required"/>
      <attribute name="op" type="sssrmap:OptionOperatorType" use="optional"/>
      <attribute name="conj" type="sssrmap:ConjunctionType" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="Option" type="sssrmap:OptionType"/>

<simpleType name="OptionOperatorType">
  <restriction base="string">
    <enumeration value="Not"/>
  </restriction>
</simpleType>
```

3.2.7 The *Data* Element

A *Request* or *Response* element may have one or more *Data* elements that allow the supplying of context-specific data. A request might pass in a structured object via a *Data* element to be acted upon. Typically a query will result in a response with the data encapsulated within a *Data* element.

The following attributes are optional:

- *name* – object name describing the contents of the data
- *type* – describing the form in which the data is represented
  - A *type* attribute of "XML" indicates the data has internal xml structure and can be recursively parsed by an XML parser
  - A *type* attribute of "Binary" indicates the data is an opaque dataset consisting of binary data
  - A *type* attribute of "String" indicates the data is an ASCII string
  - A *type* attribute of "Int" indicates the data is an integer

- A type attribute of “Text” indicates the data is in formatted human-readable text
- A type attribute of “HTML” indicates the data is represented in HTML

```
<complexType name="DataType">
  <sequence>
    <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="name" type=”string” use="optional"/>
  <attribute ref="sssrmap:Type" use="optional"/>
</complexType>

<element name="Data" type="sssrmap:DataType"/>
```

3.2.8 The *File* Element

A *Request* or *Response* element may have one or more *File* elements of type String that allow the inclusion of files. The files may be either text or binary and may be referenced by objects inside the Data element. A file may be compressed using the gzip algorithm [ZIP]. A binary file or a compressed file must be base64 encoded as defined in XML Digital Signatures (“http://www.w3.org/2000/09/xmldsig#base64”). Metadata describing the modes and properties of the resulting file are passed as parameters. The text or base64 encoded file data forms the string content of the *File* element.

The following attributes are optional:

- *id* -- specifies an identifier that allows the file to be referenced from within another object. If more than one *File* elements are specified, this attribute is REQUIRED in each of them.
- *name* -- specifies the name to give the file upon creation on the target system. This can be an absolute or relative pathname (relative to the InitialWorkingDirectory).
- *owner* – indicates what owner the file should be changed to. By default it will be changed to the UserId that the authenticated actor maps to on the target system. Note that this function should succeed only if the requestor has the privileges to do so (i.e. authenticated as root).
- *group* – indicates what group the file should be changed to. By default it will be set to the primary groupid of the UserId that the authenticated actor maps to on the target system. Note that this function should succeed only if the requestor has the proper privileges.
- *mode* – indicates the permissions the file should possess. By default it will be set according to the default umask for the UserId that the authenticated actor maps to on the target system. Note that this function should not set permissions for the file that exceed the privileges for the actor. These

permissions can be specified using either an octal number or symbolic operations (as accepted by the GNU chmod(1) command).

- *compressed* – indicates whether the file has been compressed
  - A *compressed* attribute of "True" indicates the file has been compressed.
  - A *compressed* attribute of "False" indicates the file has not been compressed. This is the default.
- encoded – indicates whether the file has been base64 encoded
  - An *encoded* attribute of "True" indicates the file has been encoded.
  - An *encoded* attribute of "False" indicates the file has not been encoded. This is the default.

```
<complexType name="FileType">
  <sequence>
    <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="name" type="string" use="optional"/>
  <attribute name="owner" type="string" use="optional"/>
  <attribute name="group" type="string" use="optional"/>
  <attribute name="mode" type="string" use="optional"/>
  <attribute name="compressed" type="boolean" use="optional"/>
  <attribute name="encoded" type="boolean" use="optional"/>
</complexType>

<element name="File" type="sssrmap:FileType"/>
```

3.2.9 The *Count* Element

A single *Count* element may be included within a *Request* or *Response* and is context-specific. This can be used to represent the number of objects acted upon or returned.

```
<element name="Count" type="positiveInteger"/>
```

3.2.10 The *Response* Element

The *Response* element specifies an individual response. It MUST contain a *Status* element. It MAY also contain *Count* and any number of *Data* or *File* elements. If chunking has been requested and is supported by the server, a large response may be broken up into multiple chunks (each with their own envelope). The chunkNum attribute can be used to indicate which chunk the current one is. The chunkMax attribute can be used to determine when all the chunks have been received (all chunks have been received if chunkNum=chunkMax or chunkMax=0).

It MAY have any of the following attributes:

- *id* -- uniquely maps the response to the corresponding request
- chunkNum -- integer indicating the current chunk number [1 is implied when this attribute is missing or blank]
- chunkMax -- integer indicating the number of chunks expected [-1 means unknown but more chunks to follow; 0 means unknown but this is the last chunk; 0 is implied if this attribute is missing or blank]

```
<complexType name="ResponseType">
  <choice minOccurs="0" maxOccurs="unbounded">
    <element ref="sssrmap:Status" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:Count" minOccurs="0" maxOccurs="1"/>
    <element ref="sssrmap:Data" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="sssrmap:File" minOccurs="0" maxOccurs="unbounded"/>
    <any minOccurs="0" maxOccurs="unbounded" namespace="##other"/>
  </choice>
  <attribute name="object" type="string" use="optional"/>
  <attribute name="action" type="string" use="optional"/>
  <attribute name="id" type="string" use="optional"/>
  <attribute name="chunkNum" type="integer" use="optional"/>
  <attribute name="chunkMax" type="integer" use="optional"/>
</complexType>

<element name="Response" type="sssrmap:ResponseType"/>
```

3.2.11 The *Status* Element

A *Response* element MUST contain a single *Status* element that indicates whether the reply represents a success, warning or failure. This element is composed of the child elements *Value*, *Code* and *Message*. Of these, *Value* and *Code* are required, and *Message* is optional.

```
<complexType name="StatusType">
  <choice minOccurs="1" maxOccurs="unbounded">
    <element ref="sssrmap:Value" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:Code" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:Message" minOccurs="0" maxOccurs="1"/>
    <any minOccurs="0" maxOccurs="unbounded" namespace="##other"/>
  </choice>
</complexType>

<element name="Status" type="sssrmap:StatusType"/>
```

3.2.12 The *Value* Element

The *Value* element is of type String and MUST have a value of “Success”, “Warning” or “Failure”.

```
<simpleType name="StatusValueType">
```

```
  <restriction base="string">
    <enumeration value="Success"/>
    <enumeration value=”Warning”/>
    <enumeration value="Failure"/>
  </restriction>
</simpleType>

<element name="Value" type="sssrmap:StatusValueType"/>
```

3.2.13 The *Code* Element

A *Response* element must contain a single *Code* element that specifies the 3-digit status code for the response. Refer to the next section on Error Reporting for a description and listing of supported status codes.

```
      <simpleType name="CodeType">
        <restriction base="string">
        <pattern value="[0-9]{3}"/>
        </restriction>
</simpleType>

<element name="Code" type="sssrmap:CodeType"/>
```

3.2.14 The *Message* Element

A *Response* element may contain a single *Message* element that is context specific to the success or failure response. The message should be an error message if status is false. If present for a successful response, it may be used as a human readable message for a user interface.

```
<element name="Message" type="string"/>
```

3.3 Modified XPATH Expressions

The *name* attribute used within the *Get*, *Set* and *Where* Elements MUST have the form of a modified XPATH expression as defined in this section. Usually this will just be the simple name of the object property. Some complex objects, such as the SSS Job Object and the SSS Node Object, however, are represented in a structured way with nested elements. In order to define a consistent and flexible way to access and manipulate these objects as well as keeping the flat XML objects simple and straightforward, SSSRMAP specifies that a “Modified XPATH” syntax be used.

In essence, “Modified XPATH” is defined to be an XPATH [XPATH] expression with the exception that the “//” may be omitted from the beginning of the expression when a document search is desired. Thus, on the server side, a standard

XPATH routine can be used by prepending "//" to any expression that does not begin with a "/".

The response data should always include all of the structure of the queried object necessary to place the requested data in its proper context.

See the XPATH specification for a full description of XPATH. The XPath 1.0 Recommendation is http://www.w3.org/TR/1999/REC-xpath-19991116. The latest version of XPath 1.0 is available at http://www.w3.org/TR/xpath.

3.3.1 Sample Modified XPATH expressions

Consider the following hypothetical object(s) (which might be returned within a Data element).

```
<Job>
        <JobId>PBS.1234.0</JobId>
        <Requested>
                <Memory op="GE">512</Memory>
                <Processors>2</Processors>
                <WallDuration>P3600S</WallDuration>
        </Requested>
        <Utilized>
                <Memory metric="Average">488</Memory>
        </WallDuration>P1441S</WallDuration>
</Job>
```

To get everything above for this job you would not need a Get element:

```
<Request action="Query">
        <Object>Job</Object>
        <Where name="JobId">PBS.1234.0</Where>
</Request>
```

If you used <Get name="**JobId**"/> you would get back:

```
<Job>
        <JobId>PBS.1234.0</JobId>
</Job>
```

If you used <Get name="**Memory**"/> (or name="**/Job/*/Memory**") you would get:

```
<Job>
        <Requested>
                <Memory op="GE">512</Memory>
```

```
        </Requested>
        <Utilized>
            <Memory metric="Average">488</Memory>
        </Utilized>
</Job>
```

If you used <Get name="**Requested/Memory**"/> (or name="**/Job/Requested/Memory**") you would get:

```
<Job>
        <Requested>
            <Memory op="GE">512</Memory>
        </Requested>
</Job>
```

If you used <Get name="**Memory[@metric='Average']**"/> (or name="**Memory[@metric]**") you would get:

```
<Job>
        <Utilized>
            <Memory metric="Average">488</Memory>
        </Utilized>
</Job>
```

3.4 Examples

3.4.1 Sample Requests

Requesting a list of nodes with a certain configured memory threshold (batch format):

```
<Request action="Query" id="1">
        <Object>Node</Object>
        <Get name="Name" />
        <Get name="Configured/Memory" />
        <Where name="Configured/Memory" op="GE"
units="MB">512</Where>
</Request>
```

Activating a couple of users:

```
<Request action="Modify">
        <Object>User</Object>
        <Set name="Active">True</Set>
```

```
        <Where name="Name">scott</Where>
        <Where name="Name" conj="Or"/>brett</Where>
</Request>
```

Submitting a simple job:

```
<Request action="Submit">
        <Object>Job</Object>
        <Data>
                <Job>
                        <User>xdp</User>
                        <Account>youraccount</Account>
                        <Command>myprogram</Command>

        <InitialWorkingDirectory>/usr/home/scl/xdp</InitialWorkingDirectory>
                        <RequestedNodes>4</RequestedNodes>
                        <RequestedWCTime>100</RequestedWCTime>
                </Job>
        </Data>
</Request>
```

3.4.2 Sample Responses

A response to the available memory nodes query (batch format)

```
<Response id="1">
        <Status>
                <Value>Success</Value>
                <Code>000</Code>
        </Status>
        <Count>2</Count>
        <Data>
                <Node>
                        <Name>fr01n01</Name>
                        <Configured>
                                <Memory>512</Memory>
                        </Configured>
                </Node>
                <Node>
                        <Name>fr12n04</Name>
                        <Configured>
                                <Memory>1024</Memory>
                        </Configured>
                </Node>
        </Data>
```

```
</Response>
```

Two users successfully activated

```
<Response>
        <Status>
                <Code>000</Code>
                <Message>Two users were successfully modified</Message>
        </Status>
        <Count>2</Count>
</Response>
```

A failed job submission:

```
<Response>
        <Status>
                <Value>Failure</Value>
                <Code>711</Code>
                <Message>Invalid account specified. The job was not
submitted.</Message>
        </Status>
</Response>
```

4 Error Reporting

SSSRMAP requests will return a status and a 3-digit response code to signify success or failure conditions. When a *request* is successful, a corresponding *response* is returned with the *status* element set to Success and the *code* element set to "000". When a *request* results in an error detected by the server, a *response* is returned with the *status* element set to Failure and a 3-digit error code in the *code* element. An optional human-readable *message* may also be include in a failure response providing context-specific detail about the failure. The default message language is US English. (The status flag makes it easy to signal success or failure and allows the receiving peer some freedom in the amount of parsing it wants to do on failure [BXXP]).

Category	Code	Response Text in US English
Success	0xx	Request was successful
	000	General Success
	010	Help/usage reply
	020	Status reply
	030	Subscription successful
	035	Notification successful (Ack)

	040	Registration successful
	050-079	Component-defined
	080-099	Application-defined
Warning	1xx	Request was successful but includes a warning
	100	General warning (examine message for details)
	102	Check result (Did what you asked but may not have been what you intended -- or information is suspect)
	110	Wire Protocol or Network warning
	112	Redirect
	114	Protocol warning (something was wrong with the protocol but best effort guesses were applied to fulfill the request)
	120	Message Format warning
	122	Incomplete specification (request missing some essential information -- best effort guess applied)
	124	Format warning (something was wrong with the format but best effort guesses were applied to fulfill the request)
	130	Security warning
	132	Insecure request
	134	Insufficient privileges (Response was sanitized or reduced in scope due to lack of privileges)
	140	Content or action warning
	142	No content (The server has processed the request but there is no data to be returned)
	144	No action taken (nothing acted upon -- i.e. deletion request did not match any objects)
	146	Partial content
	148	Partial action taken
	150-179	Component-defined
	180-199	Application-defined
Wire Protocol	2xx	A problem occurred in the wire protocol or network
	200	General wire protocol or network error
	210	Network failure
	212	Cannot resolve host name
	214	Cannot resolve service port
	216	Cannot create socket
	218	Cannot bind socket
	220	Connection failure
	222	Cannot connect
	224	Cannot send data
	226	Cannot receive data
	230	Connection rejected
	232	Timed out

	234	Too busy
	236	Message too large
	240	Framing failure
	242	Malformed framing protocol
	244	Invalid payload size
	246	Unexpected end of file
	250-279	Component-defined
	280-299	Application-defined
Message Format	3xx	A problem occurred in the message format
	300	General message format error
	302	Malformed XML document
	304	Validation error (XML Schema)
	306	Namespace error
	308	Invalid message type (Something other than Request or Response in Body)
	310	General syntax error in request
	311	Object incorrectly (or not) specified
	312	Action incorrectly (or not) specified
	313	Invalid Action
	314	Missing required element or attribute
	315	Invalid Object (or Object-Action combination)
	316	Invalid element or attribute name
	317	Illegal value for element or attribute
	318	Illegal combination
	319	Malformed Data
	320	General syntax error in response
	321	Status incorrectly (or not) specified
	322	Code incorrectly (or not) specified
	324	Missing required element or attribute
	326	Invalid element or attribute name
	327	Illegal value for element or attribute
	328	Illegal combination
	329	Malformed Data
	340	Pipelining failure
	342	Request identifier is not unique
	344	Multiple messages not supported
	346	Mixed messages not supported (Both requests and responses in same batch)
	348	Request/response count mismatch
	350-379	Component-defined
	380-399	Application-defined
Security	4xx	A security requirement was not fulfilled
	400	General security error
	410	Negotiation failure

	412	Not understood
	414	Not supported
	416	Not accepted
	420	Authentication failure
	422	Signature failed at client
	424	Authentication failed at server
	426	Signature failed at server
	428	Authentication failed at client
	430	Encryption failure
	432	Encryption failed at client
	434	Decryption failed at server
	436	Encryption failed at server
	438	Decryption failed at client
	440	Authorization failure
	442	Authorization failed at client
	444	Authorization failed at server
	450-479	Component-defined
	480-499	Application-defined
Event Management	5xx	Failure conditions in event messaging
	500	General Event Management failure
	510	Subscription failed
	520	Notification failed
	550-579	Component-defined
	580-599	Application-defined
Reserved	6xx	Reserved for future use
Server Application	7xx	A server-side application-specific error occurred
	700	General failure
	710	Not supported
	712	Not understood
	720	Internal error
	730	Resource unavailable (insufficient resources -- software, hardware or a service I rely upon is down)
	740	Business logic
	750-779	Component-defined
	780-799	Application-defined
Client Application	8xx	A client-side application-specific error occurred
	800	General failure
	810	Not supported
	812	Not understood
	820	Internal error
	830	Resource unavailable
	840	Business logic

	850-879	Component-defined
	880-899	Application-defined
Miscellaneous	9xx	Miscellaneous failures
	999	Unknown failure

5 Acknowledgements

6 References

[BEEP] M. Rose, "The Blocks Extensible Exchange Protocol Core", RFC 3080, March 2001.

[FED_XML] "U.S. Federal XML Guidelines".

[HMAC] H. Krawczyk, M. Bellare, R. Canetti, "HMAC, Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[HTTP] "Hypertext Transfer Protocol – HTTP/1.1", RFC 2616, June 1999.

[RFC2119] S. Bradner, "Key Words for Use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[RFC3117] M. Rose, "On the Design of Application Protocols", Informational RFC 3117, November 2001.

[SHA-1] U.S. Department of Commerce/National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-1.

[SSS] "Scalable Systems Software", http://www.scidac.org/ScalableSystems

[WIRE_PROTOCOL] S. Jackson, B. Bode, D. Jackson, K. Walker, "Systems Software Resource Management and Accounting Protocol (SSSRMAP) Wire Protocol", SSS Resource Management and Accounting Documents, January 2004.

[XML] Bray, T., et al, "Extensible Markup Language (XML) 1.0 (Second Edition)", 6 October 2000.

[XML_CONV] "I-X and <I-N-CA> XML Conventions".

[XML_DSIG] D. Eastlake, J. Reagle Jr., D. Solo, "XML Signature Syntax and Processing", W3C Recommendation, 12 February 2002.

[XML_ENC] T. Imamura, B. Dillaway, E. Smon, "XML Encryption Syntax and Processing", W3C Candidate Recommendation, 4 March 2002.

[XML_SCHEMA] D. Beech, M. Maloney, N. Mendelshohn, "XML Schema Part 1: Structures Working Draft", April 2000.

[XPath 1.0] J. Clark, S. DeRose, "*XML Path Language (XPath) Version 1.0*", 16 November 1999.

[XRP] E. Brunner-Williams, A. Damaraju, N. Zhang, "Extensible Registry Protocol (XRP)", Internet Draft, expired August 2001.

[ZIP] J. Gailly, M. Adler, "The gzip home page", http://www.gzip.org/