

Scalable Systems Software Job Object Specification

Status of this Memo

This document describes the job object to be used by Scalable Systems Software compliant components. It is envisioned for this specification to be used in conjunction with the SSSRMAP protocol with the job object passed in the Data field of Requests and Responses. Queries can be issued to a job-cognizant component in the form of modified XPATH expressions to the Get field to extract specific information from the job object as described in the SSSRMAP protocol.

Abstract

This document describes the syntax and structure of the SSS job object. A job model is described that is flexible enough to support the specification of very simple jobs as well as jobs with elaborate and complex specification requirements in a way that avoids complex structures and syntax when it is not needed. The basic assumption is that a solitary job specification should be usable for all phases of the job lifecycle and can be used at submission, queuing, staging, reservations, quotations, execution, charging, accounting, etc. This job specification provides support for multi-step jobs, as well as jobs with disparate task descriptions. It accounts for operational requirements in a grid or meta-scheduled environment where the job is executed by multiple hosts in different administrative domains that support different resource management systems.

Table of Contents

Scalable Systems Software Job Object Specification	1
Table of Contents	1
1. Introduction	3
1.1 Goals	3
1.2 Non-Goals	3
1.3 Examples	4
1.3.1 Very Simple Example	4
1.3.2 Moderate Example	4
1.3.3 Elaborate Example	5
2. Conventions used in this document.....	7
2.1 Keywords	7
2.2 Table Column Interpretations	7

2.3	Element Syntax Cardinality	8
3.	The Job Model.....	8
4.	JobGroup Element.....	10
4.1	JobGroup Properties.....	10
4.1.1	Simple JobGroup Properties	11
4.1.2	Job.....	11
4.1.3	JobDefaults	11
5.	Job and JobDefaults Element	11
5.1	Job Properties.....	12
5.1.1	Simple Job Properties	12
5.1.2	Feature Element	17
5.1.3	OutputFile Element.....	17
5.1.4	ErrorFile Element.....	18
5.1.5	InputFile Element.....	18
5.1.6	NotificationList Element.....	19
5.1.7	ResourceLimit Element	19
5.1.8	Credentials	20
5.1.9	Environment Element	20
5.1.9.1	Variable Element	21
5.1.10	Nodes Element	21
5.1.10.1	Node Element.....	21
5.1.11	TaskDistribution Element	22
5.1.12	Dependency Element	22
5.1.13	Consumable Resources	23
5.1.14	Resource Element	25
5.1.15	Extension Element	25
5.1.16	TaskGroup.....	26
5.1.17	TaskGroupDefaults	26
6.	TaskGroup and TaskGroupDefaults Element	26
6.1	TaskGroup Properties	27
6.1.1	Simple TaskGroup Properties	27
6.1.2	Task.....	27
6.1.3	TaskDefaults	27
7.	Task and TaskDefaults Element.....	28
7.1	Task Properties.....	28
7.1.1	Simple Task Properties	28
8.	Property Categories	29
8.1	Requested Element.....	29
8.2	Delivered Element	31
9.	AwarenessPolicy Attribute.....	32
10.	References	33
	Appendix A	34
	Units of Measure Abbreviations	34

1. Introduction

This specification proposes a standard XML representation for a job object for use by the various components in the SSS Resource Management System. This object will be used in multiple contexts and by multiple components. It is anticipated that this object will be passed via the Data Element of SSSRMAP Requests and Responses.

1.1 Goals

There are several goals motivating the design of this representation.

The representation needs to be inherently flexible. We recognize we will not be able to exhaustively include the ever-changing job properties and capabilities that constantly arise.

The representation should use the same job object at all stages of that job's lifecycle. This object will be used at job submission, queuing, scheduling, charging and accounting, hence it may need to distinguish between requested and delivered properties.

The design must account for the properties and structure required to function in a meta or grid environment. It needs to include the capability to support local mapping of properties, global namespaces, etc.

The equivalent of multi-step jobs must be supported. Each step (job) can have multiple logical task descriptions.

Many potential users of the specification will not be prepared to implement the complex portions or fine-granularity that others need. There needs to be a way to allow the more complicated structure to be added as needed while leaving more straightforward cases simple.

There needs to be guidance for how to understand a given job object when higher order features are not supported by an implementation, and which parts are required, recommended and optional for implementers to implement.

It needs to support composite resources.

It should include the ability to specify preferences or fuzzy requirements.

1.2 Non-Goals

Namespace considerations and naming conventions for most property values are outside of the scope of this document.

1.3 Examples

1.3.1 Very Simple Example

This example shows a simple job object that captures the requirements of a simple job.

```
<Job>
  <JobId>PBS.1234.0</JobId>
  <JobState>Idle</JobState>
  <UserId>scottmo</UserId>
  <Executable>/bin/hostname</Executable>
  <Processors>16</Processors>
  <WallclockDuration>3600</WallclockDuration>
</Job>
```

1.3.2 Moderate Example

This example shows a moderately complex job object that uses features such as required versus delivered properties.

```
<Job>
  <JobId>PBS.1234.0</JobId>
  <JobName>Heavy Water</JobName>
  <ProjectId>nwchemdev</ProjectId>
  <UserId>peterk</UserId>
  <Application>NWChem</Application>
  <Executable>/usr/local/nwchem/bin/nwchem</Executable>
  <Arguments>-input basis.in</Arguments>
  <InitialWorkingDirectory>/home/peterk</InitialWorkingDirectory>
  <MachineName>Colony</MachineName>
  <QualityOfService>BottomFeeder</QualityOfService>
  <QueueName>batch_normal</QueueName>
  <JobState>Completed</JobState>
  <StartTime>1051557713</StartTime>
  <EndTime>1051558868</EndTime>
  <Charge>25410</Charge>
  <Requested>
    <Processors op="GE">12</Processors>
    <Memory op="GE" units="GB">2</Memory>
    <WallclockDuration>3600</WallclockDuration>
  </Requested>
  <Delivered>
    <Processors>16</Processors>
    <Memory metric="Average" units="GB">1.89</Memory>
```

```

        <WallclockDuration>1155</WallclockDuration>
    </Delivered>
    <Environment>
        <Variable name="PATH">/usr/bin:/home/peterk</Variable>
    </Environment>
</Job>

```

1.3.3 Elaborate Example

This example uses a job group to encapsulate a multi-step job. It shows this protocol's ability to characterize complex job processing capabilities. A component that processes this message is free to retain only that part of the information that it requires. Superfluous information can be ignored by the component or filtered out (by XSLT for example).

```

<JobGroup>
    <JobGroupId>fr15n05.1234</JobGroupId>
    <JobGroupState>Active</JobGroupState>
    <JobGroupName>ShuttleTakeoff</JobGroupName>
    <JobDefaults>
        <StagedTime>1051557859</StagedTime>
        <SubmitHost>asteroid.lbl.gov</SubmitHost>
        <SubmissionTime>1051556734</SubmissionTime>
        <ProjectId>GrandChallenge18</ProjectId>
        <GlobalUserId>C=US,O=LBLN,CN=Keith Jackson</GlobalUserId>
        <UserId>keith</UserId>
        <Environment>
            <Variable name="LD_LIBRARY_PATH">/usr/lib</Variable>
            <Variable name="PATH">/usr/bin:~/bin:</Variable>
        </Environment>
    </JobDefaults>
    <Job>
        <JobId>fr15n05.1234.0</JobId>
        <JobName>Launch Vector Initialization</JobName>
        <Executable>/usr/local/gridphys/bin/lvcalc</Executable>
        <QueueName>batch</QueueName>
        <JobState>Completed</JobState>
        <MachineName>SMP2.emsl.pnl.gov</MachineName>
        <StartTime>1051557713</StartTime>
        <EndTime>1051558868</EndTime>
        <QuoteId>http://www.pnl.gov/SMP2#654321</QuoteId>
        <Charge units="USD">12.75</Charge>
        <Requested>
            <WallclockDuration>3600</WallclockDuration>
            <Processors>2</Processors>
            <Memory>1024</Memory>
        </Requested>
    </Job>
</JobGroup>

```

```

</Requested>
</Delivered>
  <WallclockDuration>1155</WallclockDuration>
  <Processors consumptionRate="0.78">2</Processors>
  <Memory metric="Max">975</Memory>
</Delivered>
<TaskGroup>
  <TaskCount>2</TaskCount>
  <TaskDistribution type="TasksPerNode">1</TaskDistribution>
  <Task>
    <Node>node1</Node>
    <ProcessId>99353</ProcessId>
  </Task>
  <Task>
    <Node>node12</Node>
    <ProcessId>80209</ProcessId>
  </Task>
</TaskGroup>
</Job>
<Job>
  <JobId>fr15n05.1234.1</JobId>
  <JobName>3-Phase Ascension</JobName>
  <QueueName>batch_normal</QueueName>
  <JobState>Idle</JobState>
  <MachineName>Colony.emsl.pnl.gov</MachineName>
  <Priority>1032847</Priority>
  <Hold>System</Hold>
  <StatusMessage>Insufficient funds to start job</StatusMessage>
  <Requested>
    <WallclockDuration>43200</WallclockDuration>
  </Requested>
  <TaskGroup>
    <TaskCount>1</TaskCount>
    <TaskGroupName>Master</TaskGroupName>
    <Executable>/usr/local/bin/stage-coordinator</Executable>
    <Memory>2048</Memory>
    <Resource name="License" type="ESSL2">1</Resource>
    <Feature>Jumbo-Frame</Feature>
  </TaskGroup>
  <TaskGroup>
    <TaskGroupName>Slave</TaskGroupName>
    <TaskDistribution type="Rule">RoundRobin</TaskDistribution>
    <Executable>/usr/local/bin/stage-slave</Executable>
    <NodeCount>4</NodeCount>
    <Requested>
      <Processors group="-1">12</Processors>
    </Requested>
  </TaskGroup>

```

```

        <Processors conj="Or" group="1">16</Processors>
        <Memory>512</Memory>
        <Nodes>
            <Node aggregation="Pattern">fr15n.*</Node>
        </Nodes>
    </Requested>
</TaskGroup>
</Job>
</JobGroup>

```

2. Conventions used in this document

2.1 Keywords

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

2.2 Table Column Interpretations

The columns of the property tables in this document have the following meanings:

Element Name: Name of the XML element (xsd:element) see [DATATYPES]

Type: Data type defined by xsd (XML Schema Definition) as:

String	xsd:string (a finite length sequence of printable characters)
Integer	xsd:integer (a signed finite length sequence of decimal digits)
Float	xsd:float (single-precision 32-bit floating point)
Boolean	xsd:boolean (consists of the literals “true” or “false”)
DateTime	xsd:int (a 32-bit unsigned long in GMT seconds since the EPOCH)
Duration	xsd:int (a 32-bit unsigned long measured in seconds)

Description: Brief description of the meaning of the property

Appearance: An indication of whether the given property must appear in the parent element. It assumes the following meanings:

MUST	This property is REQUIRED when the parent is specified.
SHOULD	This property is RECOMMENDED when the parent is specified.
MAY	This property is OPTIONAL when the parent is specified.

Compliance: An indication of the relative importance of supporting the given property.

MUST	A compliant implementation MUST support this property.
SHOULD	A compliant implementation SHOULD support this property.
MAY	A compliant implementation MAY support this property.

Categories: Some properties may be categorized into one of several categories. Letters in this column indicate that the given property can be classified in the following property categories.

R	This property can be encompassed in a Requested element.
D	This property can be encompassed in a Delivered element.

2.3 Element Syntax Cardinality

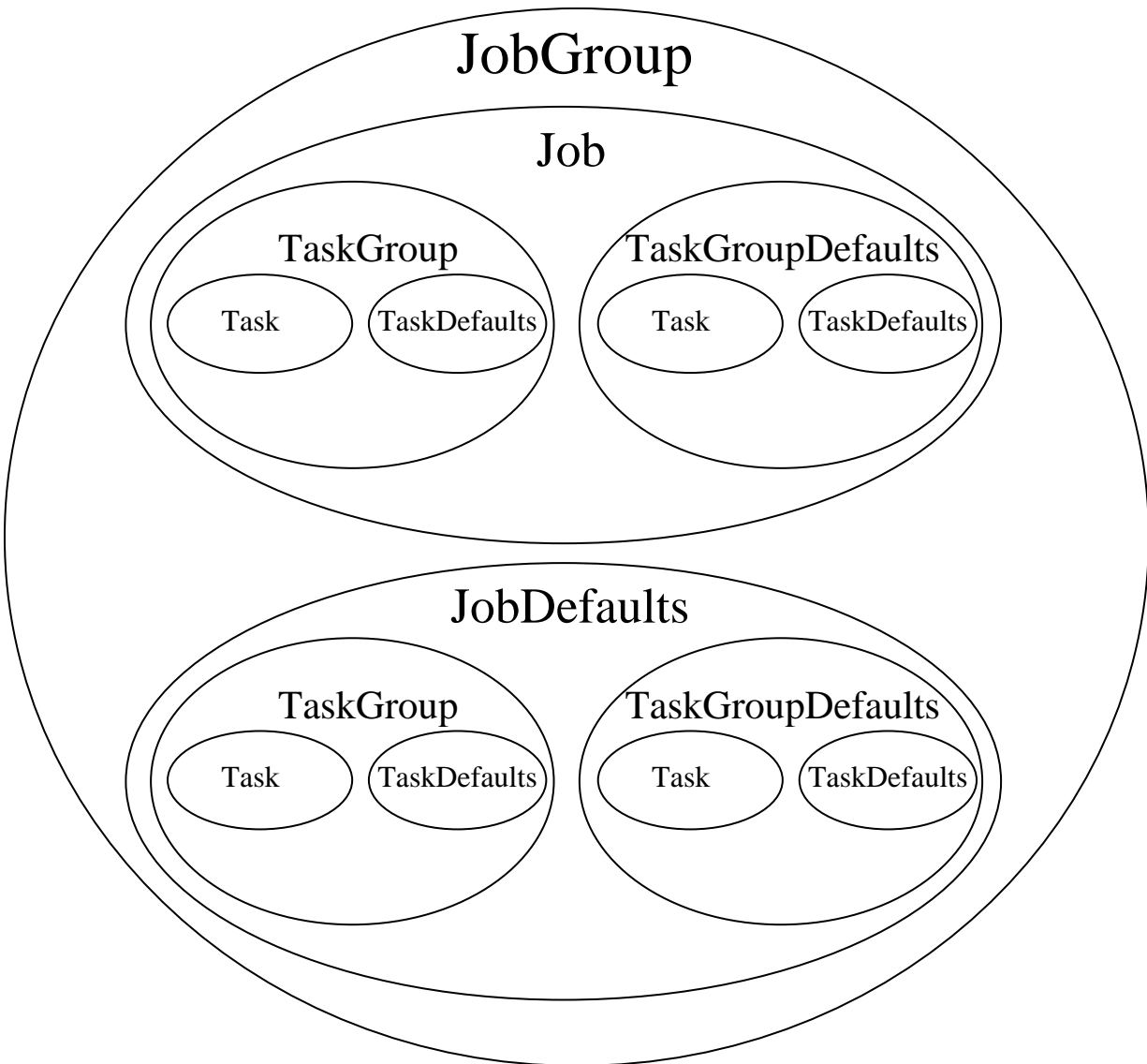
Selected elements in the element syntax sections use regular expression wildcards with the following meanings:

*	Zero or more occurrences
+	One or more occurrences
?	Zero or one occurrences

The absence of one of these symbols implies exactly one occurrence.

3. The Job Model

The primary object within the job model is a job. A job can be thought of as a single schedulable entity and will be the object normally seen in job queues.



Jobs with dependencies on other jobs may be submitted in a job group. Jobs within a job group form a DAG (directed acyclic graph) where the nodes are jobs and the edges represent dependencies on the status of previous jobs. A job group will consist of at least one job. A job group can optionally specify job defaults which are a set of job properties to be assumed by all jobs within the job group unless overridden within the job.

A job may consist of multiple tasks, which are the finest grained work unit and represent an endpoint for executing a given process instance. For example, a job that requests 3 nodes and 4 processors will have 4 tasks, two on one node and one on each of two nodes. Tasks may be grouped into task groups, which are logical aggregations of tasks and their common properties.

Submit filters, prologs, epilogs, notification scripts, etc. run once only for each job. Whereas task groups function as logical descriptions of tasks and their properties, they also describe the number of such tasks and the nodes that they run on. As an example, a master task group (consisting of a single task) might ask for a node with a MATLAB license, 2GB of memory and an internet connected network adapter while a slave task group (consisting of 12 tasks) could be targeted for nodes with more CPU bandwidth -- all within the same job and utilizing a common MPI ring. Tasks (and hence taskgroups) can have different executables or environments, specify different consumable resources or node properties. A job, therefore, may specify one or more task group. A job that does not specify an explicit task group is considered as having a single implicit task group. A job can optionally specify task group defaults which are a set of task group properties to be assumed by all task groups within the job unless overridden within a task group.

A task group may specify one or more tasks. A task group that does not specify an explicit task is considered as having a single implicit task. A task group can optionally specify task defaults which are a set of task properties to be assumed by all tasks within the task group unless overridden within a task.

4. JobGroup Element

A JobGroup is an optional element that aggregates one or more interdependent jobs. Some resource managers support the submission of job groups (multi-step jobs) and queries on the status of an entire job group.

- A compliant implementation MAY support this element.
- A JobGroup MUST specify one or more JobGroup Properties.
- A JobGroup MUST contain one or more Jobs.
- A JobGroup MAY contain zero or more JobsDefaults.

The following illustrates this element's syntax:

```
<JobGroup>  
  <!-- JobGroup Properties -->+  
  <Job/>+  
  <JobDefaults/>?  
</JobGroup>
```

4.1 JobGroup Properties

JobGroup Properties are properties that apply to the job group as a whole. These include the job group id, jobs and job defaults, and other simple optional job properties.

4.1.1 Simple JobGroup Properties

Simple (unstructured) job group properties are enumerated in Table 1.

Table 1 Simple JobGroup Properties

Element Name	Type	Description	Appearance	Compliance	Categories
CreationTime	DateTime	Date and time that the job group was instantiated	MAY	MAY	
Description	String	Description of the job group	MAY	MAY	
JobGroupId	String	Job group identifier	MUST	MUST	
JobGroupName	String	Name of the job group	MAY	SHOULD	
JobGroupState	String	State of the job as a whole. Valid states may include “NotQueued”, “Unstarted”, “Active”, “Completed”.	MAY	SHOULD	

4.1.2 Job

A job group MUST contain one or more jobs.

See the next section for element details.

4.1.3 JobDefaults

A job group MAY contain zero or one job defaults.

See the next section for element details.

5. Job and JobDefaults Element

The Job and JobDefaults elements are of the same structure. A Job element encapsulates a job and may be expressed as a standalone object. A JobDefaults element may only appear within a

JobGroup and represents the defaults to be taken by all jobs within the job group. Job properties in Job elements override any properties found in a sibling JobDefaults element.

- A compliant implementation **MUST** support the Job element.
- A compliant implementation **MAY** support the JobDefaults element only if it supports the JobGroup element.
- A job **MUST** specify one or more Job Properties.
- One or more TaskGroup elements **MAY** appear at this level.
- Zero or one TaskGroupDefaults elements **MAY** appear at this level.

The following illustrates this element’s syntax:

```
<Job>
  <!-- Job Properties -->+
  <TaskGroup/>*
  <TaskGroupDefaults/>?
</Job>
```

5.1 Job Properties

Job Properties apply to a particular job or as default properties to all jobs. They include the job id, job credentials, task groups, task group defaults, and other simple optional properties.

5.1.1 Simple Job Properties

Simple (unstructured) job properties are enumerated in Table 2.

Table 2 Simple Job Properties

Element Name	Type	Description	Appearance	Compliance	Categories
Application	String	Type of application such as “Gaussian” or “Nwchem”.	MAY	MAY	
Architecture	String	The architecture for the nodes on which this job must run.	MAY	MAY	RD
Arguments	String	The arguments for the executable.	MAY	SHOULD	

Charge	Float	The amount charged for the job.	MAY	SHOULD	
Checkpointable	Boolean	Can this job be checkpointed?	MAY	MAY	
CpuDuration	Duration	Number of cpu seconds used by the job.	MAY	SHOULD	
DeadlineTime	DateTime	Date and time that a job must end by.	MAY	MAY	
EligibleTime	DateTime	Date and time that a job must start after.	MAY	MAY	
EndTime	DateTime	Date and time that a job ended (independent of success or failure).	MAY	MUST	
Executable	String	Executable. This may be an absolute or relative path or a URI.*	MAY	MUST	
ExitCode	Integer	Exit code for the job	MAY	SHOULD	
GlobalJobId	String	Globally unique job identifier (possibly in the form of a URI).	MAY	SHOULD	
Hold	String	Hold(s) on the job. There may be multiple instances of this element if there are more than one holds on the job.	MAY	SHOULD	

InitialWorkingDirectory	String	Initial working directory	MAY	SHOULD	
Interactive	Boolean	Is this an interactive job?	MAY	SHOULD	
JobId	String	A local job identifier assigned to the job by the local resource manager.	MUST	MUST	
JobName	String	Name of the job	MAY	SHOULD	
JobState	String	State of the job. Valid states may include "Idle", "Hold", "Running", "Suspended", "Completed".	MAY	MUST	
JobType	String	Type of job. Meaning of this extension property is context specific.	MAY	MAY	
MachineName	String	Name of the system or cluster that runs the job.	MAY	MUST	RD
NetworkType	String	Type of network adapter required by the job.	MAY	MAY	RD
NodeCount	Integer	Number of nodes used by the job.	MAY	MUST	RD
OperatingSystem	String	Operating System required by the job.	MAY	MAY	RD
PartitionName	String	Name of the	MAY	MAY	RD

		partition in which the job should run.			
Priority	Integer	Current queue priority (or rank) for the job.	MAY	SHOULD	
QualityOfService	String	Name of the Quality of Service (QOS).	MAY	SHOULD	RD
QueueName	String	Name of the Queue (or class) that the job runs in.	MAY	SHOULD	RD
QuoteId	String	Identifier for a guaranteed charge rate quote obtained by the job.	MAY	MAY	
ReservationId	String	Identifier for a reservation used by the job.	MAY	MAY	RD
ReservationTime	DateTime	Date and time that a reservation was placed for the job.	MAY	MAY	
ResourceManagerType	String	Type of resource manager required to run this job.	MAY	MAY	RD
Restartable	Boolean	Can this job be restarted?	MAY	MAY	
ShellName	String	Specified the shell necessary to interpret the job script.	MAY	MAY	
StagedTime	DateTime	Date and time that a job was staged to the local resource	MAY	MAY	

		management system.			
StartCount	Integer	Number of times the scheduler tried to start the job.	MAY	MAY	
StartTime	DateTime	Date and time that the job started.	MAY	MUST	
StatusMessage	String	Natural language message that can be used to provide detail on why a job failed, isn't running, etc.	MAY	SHOULD	
SubmitTime	DateTime	Date and time that a job was submitted.	MAY	SHOULD	
SubmitHost	String	FQDN of host where the job was submitted from.	MAY	SHOULD	
Suspendable	Boolean	Can this job be suspended?	MAY	MAY	
SuspendDuration	Integer	Number of seconds the job was in the "Suspended" state.	MAY	MAY	
TimeCategory	String	This allows the specification of shifts like "PrimeTime" for charging purposes.	MAY	MAY	
WallclockDuration	Duration	Number of seconds in the "Running" state.	SHOULD	MUST	RD

* The Executable may be a script or a binary executable. If it is already on the target system it may be referenced by an absolute or relative pathname (relative to InitialWorkingDirectory). If it is passed with the job in a File object (see SSSRMAP), it can be referenced by an absolute or relative URI. An absolute URI would specify a URL where the file can be downloaded (like with wget). A relative URI is specified by preceding an identifier by a pound sign as in `<Executable>#Script</Executable>` and will be found in a File object included along with the Job object with the Script as an identifier as in `<File id="Script">echo hello world</File>`.

5.1.2 Feature Element

The Feature element connotes an arbitrary named feature of a node.

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or one times within a given set of Job Properties.
- This element is of type String.
- This element MAY have an **aggregation** attribute of type String that provides a way to indicate multiple values with a single expression. A compliant implementation MAY support the *aggregation* attribute if the Feature element is supported. Possible values for this attribute include:
 - List a comma-separated list of features
 - Pattern a regular expression (perl5) matching desired features
- This element MAY be categorized as a requested or delivered property by being encompassed by the appropriate element.

The following is an example of a feature element:

```
<Feature aggregation="List">node1,node2</Feature>
```

5.1.3 OutputFile Element

The *OutputFile* element specifies the name of the file to which the output stream (stdout) from the job will be written.

- This element's character content is the name of the file. If this element is omitted or it is empty, then an appropriate output file is auto-determined by the queuing system.
- This element MAY have a **redirectList** attribute which is a comma-separated list of output redirection attributes of type String. A compliant implementation SHOULD support this attribute if *OutputFile* is supported. Possible values for this attribute include:
 - Append opens the output file for append
 - Close closes and discards the output stream
 - Flush output is written to output file as it is generated
 - Keep leave the output file on the execution host
 - Merge merges the output stream into the error stream

Note that when using the *redirectList* attributes, the cumulative affect of the *ErrorFile* and *OutputFile* directives may be order dependent.

The following is an example of an *OutputFile* element:

```
<OutputFile redirectList="Append">~/myjob.out</OutputFile>
```

5.1.4 ErrorFile Element

The *ErrorFile* element specifies the name of the file to which the error stream (stderr) from the job will be written.

- This element's character content is the name of the file. If this element is omitted or it is empty, then an appropriate error file is auto-determined by the queuing system.
- This element MAY have a ***redirectList*** attribute which is a comma-separated list of error redirection attributes of type String. A compliant implementation SHOULD support this attribute if *ErrorFile* is supported. Possible values for this attribute include:
 - Close closes and discards the error stream
 - Append opens the error file for append
 - Flush output is written to output file as it is generated
 - Keep leave the output file on the execution host
 - Merge merges the error stream into the output stream

Note that when using the *redirectList* attributes, the cumulative affect of the *ErrorFile* and *OutputFile* directives may be order dependent.

The following is an example of an *ErrorFile* element:

```
<ErrorFile redirectList="Merge"></ErrorFile>
```

5.1.5 InputFile Element

The *InputFile* element specifies the name of the file from which the input stream (stdin) for the job will be read.

- This element's character content is the name of the file. If this element is omitted or it is empty, then an appropriate input file is auto-determined by the queuing system.
- This element MAY have a ***redirectList*** attribute which is a comma-separated list of input attributes of type String. A compliant implementation SHOULD support this attribute if *InputFile* is supported. Possible values for this attribute include:
 - Close closes and discards the input stream

The following is an example of an *InputFile* element:

```
<InputFile redirectList="Close"></InputFile>
```

5.1.6 NotificationList Element

The *NotificationList* element specifies the job-related events or conditions for which a notification will be sent.

- This element's character content is a comma-separated list of events or conditions for which a notification should be sent. Possible values for the elements of this list include:
 - JobStart send a notification when the job starts
 - JobEnd send a notification when the job ends
 - All send notifications for all notifiable events
 - None do not send notifications for any events
- This element MAY have a *uri* attribute of type String which indicates where the notification is to be sent. A compliant implementation MAY support this attribute if *NotificationList* is supported. The *uri* is in the format: [scheme://]authority with the scheme being smtp and the authority being an email address by default.

The following is an example of a *NotificationList* element:

```
<NotificationList uri="smith@business.com">JobStart,JobEnd</NotificationList>
```

5.1.7 ResourceLimit Element

The *ResourceLimit* element represents a resource limit with its name and value.

- This element MUST have a *name* attribute of type String. A compliant implementation MUST support the name attribute if ResourceLimit is supported.
- This element MAY have a *type* attribute of type String that may have the values "Hard" or "Soft". If the limit is enforced by the operating system, a hard limit is one that cannot be increased once it is set while a soft limit may be increased up to the value of the hard limit. If the *type* attribute is omitted, both the soft and hard limits are set.
- This element's character content is the resource limit's value.

Some typical names include:

CoreFileSize	Maximum core file size
CpuTime	CPU time in seconds
DataSegSize	Maximum data size
FileSize	Maximum file size
MaxMemorySize	Maximum resident set size
MaxProcesses	Maximum number of processes
MaxSwap	Virtual memory limit
MaxMemLock	Maximum locked-in-memory address space
MaxProcessors	Maximum processors
MaxMemory	Maximum memory
MaxDisk	Maximum disk space

MaxNetwork	Maximum network bandwidth
MaxFileIO	Maximum file i/o
OpenFiles	Maximum number of open files
StackSize	Maximum stack size

The following is an example of a ResourceLimit element:

```
<ResourceLimit name="CPUTime">1000000</ResourceLimit>
```

5.1.8 Credentials

Credentials are a special group of job properties that characterize an authenticated token or id. They can be categorized in both requested and delivered forms.

Credential job properties are enumerated in Table 3.

Table 3 Credential Job Properties

Element Name	Type	Description	Appearance	Compliance	Categories
ProjectId	String	Name of the Project or Charge Account	MAY	SHOULD	RD
GlobalUserId	String	Globally unique user identifier. This may be an X.509 DN for example.	MAY	SHOULD	RD
GroupId	String	Name of the local group id.	MAY	MAY	RD
UserId	String	Name of the local userid for the job.	MAY	MUST	RD

5.1.9 Environment Element

The Environment element encapsulates environment variables.

- This element MAY have an *export* attribute of type Boolean that which if set to True indicates that all environment variables in the context of the job submission process should be exported in the job's execution environment.
- A compliant implementation SHOULD support this element.
- An Environment element MAY appear zero or one times within a given set of Job (or TaskGroup) Properties.

- An Environment element MAY contain one or more Variable elements.

The following illustrates this element's syntax:

```
<Environment>
  <Variable/>+
</Environment>
```

5.1.9.1 Variable Element

The Variable element represents an environment variable with its name and value. This element MUST have a *name* attribute of type String. A compliant implementation MUST support the name attribute if Variable is supported. This element's character content is the environment variable's value.

The following is an example of a Variable element:

```
<Variable name="PATH"/>/usr/bin:/home/sssdemo</Variable>
```

5.1.10 Nodes Element

The Nodes element aggregates nodes.

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or one times within a given set of Job Properties.
- This element MUST contain one or more Node elements.
- This element MAY be categorized as a requested or delivered property by being encompassed by the appropriate element.

The following illustrates this element's syntax:

```
<Nodes>
  <Node/>+
</Nodes>
```

5.1.10.1 Node Element

The Node element represents a node.

- A compliant implementation SHOULD support this element.
- This element is of type String.
- This element MAY appear one or one times within a Nodes element.

- This element MAY have an *aggregation* attribute of type String that provides a way to indicate multiple values with a single expression. A compliant implementation MAY support the *aggregation* attribute if the Node element is supported. Possible values for this attribute include:
 - List a comma-separated list of features
 - Pattern a regular expression (perl5) matching desired features
- This element MAY be categorized as a requested or delivered property by being encompassed by the appropriate element.

The following is an example of a Node element:

```
<Node aggregation="Pattern">node[1-5]</Node>
```

5.1.11 TaskDistribution Element

The TaskDistribution element describes how tasks are to be mapped to nodes. This mapping may be expressed as a rule name, a task per node ratio or an arbitrary geometry.

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or one times in a given set of Job (or TaskGroup) Properties.
- This element is of type String.
- This element MAY have a *type* attribute of type String that provides a hint as to the type of mapping guidance provided. It may have values including “Rule”, “TasksPerNode”, “ProcessorsPerTask” or “Geometry”. A compliant implementation MAY support the *type* attribute if the TaskDistribution element is supported.
- It is possible to use Processors, NodeCount and TaskCount elements to specify a set of mutually contradictory task parameters. When this occurs, components are responsible for resolving conflicting requirements.

The following are three examples of a TaskDistribution element:

```
<TaskDistribution type="TasksPerNode">2</TaskDistribution>
```

```
<TaskDistribution type="Rule">RoundRobin</TaskDistribution>
```

```
<TaskDistribution type="Geometry">{1,4}{2}{3,5}</TaskDistribution>
```

5.1.12 Dependency Element

The Dependency element allows a job’s execution to depend on the status of other jobs. In a job group (multi-step job), some jobs may delay execution until the failure or success of other jobs creating in general a Directed Acyclic Graph relationship between the jobs. This element’s content is of type String and represents the job that the current job is dependent upon. Since a job

may have two or more dependencies, this element may appear more than once in a given job scope. A compliant implementation SHOULD support this element if job groups are supported.

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or more times in a given set of Job (or TaskGroup) Properties.
- This element is of type String and contains the JobId that the current job is dependent upon.
- This element MAY have a *condition* attribute of type String that indicates the basis for determining when the current job executes in relation to the specified job. A compliant implementation MUST support this attribute if this element is supported.
Possible values for this attribute include:
 - OnSuccess this job should run after the referenced job only if it completes successfully (this is the default if the *type* attribute is omitted)
 - OnFailure this job should run after the referenced job only if it fails
 - OnExit this job should run after the referenced job exits
- If the *condition* attribute is equal to “OnExit”, this element MAY have a *code* attribute of type Integer that indicates the exit code that will trigger this job to run. If the code attribute is omitted, then the current job should run after the referenced job for any exit status.
- This element MAY have a *designator* attribute of type String that indicates that indicates the property of the job that identifies it as the dependent job. A compliant implementation MAY support this attribute if this element is supported.
Possible values for this attribute include:
 - JobId the job this job is dependent upon is specified by JobId (this is the default if the *designator* attribute is omitted)
 - JobName the job(s) this job is dependent upon are specified by JobName

The following is an example of a Dependency element:

```
<Dependency condition="OnSuccess" designator="JobId">PBS.1234.0</Dependency>
```

5.1.13 Consumable Resources

Consumable Resources are a special group of properties that can have additional attributes and can be used in multiple contexts. In general a consumable resource is a resource that can be consumed in a measurable quantity.

- A consumable resource MAY have a *context* attribute of type String that indicates the sense in which the resource is used. A compliant implementation MAY support this attribute. Possible values for this attribute include:
 - Configured run this task only on nodes having the specified configured resources
 - Available run this task only on nodes having the specified available resources. (this is the default if the *context* attribute is omitted)

- Used the task used the indicated resources (this is analogous to being including in a Delivered block)
- Dedicated the indicated amount of the resource should be dedicated to the task
- A consumable resource MAY have a ***units*** attribute that is of type String that specifies the units by which it is being measured. If this attribute is omitted, a default unit is implied. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a ***metric*** attribute that is of type String that specifies the type of measurement being described. For example, the measurement may be a Total, an Average, a Min or a Max. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a ***wallDuration*** attribute of type Duration that indicates the amount of time for which that resource was used. This need only be specified if the resource was used for a different amount of time than the wallDuration for the job. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a ***consumptionRate*** attribute of type Float that indicates the average percentage that a resource was used over its wallDuration. For example, an overbooked SMP running 100 jobs across 32 processors may wish to scale the usage and charge by the average fraction of processor usage actually delivered. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a ***dynamic*** attribute of type Boolean that indicates whether the resource allocated for this job should be allowed to grow or shrink dynamically. For example, if processors is specified with dynamic equal to “True”, the job may be dynamically allocated more processors as they become available. The growth bounds can be indicated via the *op* attribute which is inherited when a consumable resource element is encapsulated within a *Requested* element. A compliant implementation MAY support this attribute if the element is supported.

A list of simple consumable resources is listed in Table 4.

Table 4 Simple Consumable Resources

Element Name	Type	Description	Appearance	Compliance	Categories
Disk	Float	Amount of disk.	MAY	SHOULD	RD
Memory	Float	Amount of memory.	MAY	SHOULD	RD
Network	Float	Amount of network.	MAY	MAY	RD
Processors	Integer	Number of processors.	MAY	MUST	RD
Swap	Float	Amount of virtual memory.	MAY	MAY	RD

The following are two examples for specifying a consumable resource:

```
<Memory metric="Max" units="GB">483</Memory>
```

```
<Processors wallDuration="1234" consumptionRate="0.63">4</Processors>
```

5.1.14 Resource Element

In addition to the consumable resources enumerated in the above table, an extensible consumable resource is defined by the Resource element.

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or more times within a given set of job (or task group) properties.
- Like the other consumable resources, this property MAY be categorized as a requested or delivered property by being encompassed in the appropriate element.
- This element is of type Float.
- This element shares the **same properties and attributes as the other consumable resources** but it requires an additional *name* (and optional *type*) attribute to describe it.
- It MUST have a *name* attribute of type String that indicates the type of consumable resource being measured. A compliant implementation MUST support this attribute if the element is supported.
- It MAY have a *type* attribute of type String that distinguishes it within a general resource class. A compliant implementation SHOULD support this attribute if the element is supported.

The following are two examples for specifying a Resource element:

```
<Resource name="License" type="MATLAB">1</Resource>
```

```
<Resource name="Telescope" type="Zoom2000" wallDuration="750"  
metric="KX">10</Resource>
```

5.1.15 Extension Element

The Extension element provides a means to pass extensible properties with the job object. Some applications may find it easier to use a named extension property than discover and handle elements they do not understand or anticipate by name.

- A compliant implementation MAY support this element.
- This element MUST have a *name* attribute of type String that gives the extension property's name. A compliant implementation MUST support this attribute if this element is supported.

- This element MAY have a *type* attribute of type String that characterizes the context within which the property should be understood. A compliant implementation SHOULD support this attribute if this element is supported.
- This element's character content, which is of type String, is the extension property's value.

The following is an example of an Extension element:

```
<Extension type="Scheduler" name="Restartable">true</Extension>
```

5.1.16 TaskGroup

A job MAY specify one or more task groups.

See the next section for element details.

5.1.17 TaskGroupDefaults

A job MAY specify zero or more task group defaults.

See the next section for element details.

6. TaskGroup and TaskGroupDefaults Element

The TaskGroup and TaskGroupDefaults elements have the same structure. A TaskGroup element aggregates tasks. A TaskGroupDefaults element may only appear within a Job (or JobDefaults) and represents the defaults to be taken by all task groups within the job. Task group properties in TaskGroup elements override any properties found in a sibling TaskGroupDefaults element.

- A compliant implementation MAY support the TaskGroup element.
- A compliant implementation MAY support the TaskGroupDefaults element.
- A task group MUST specify one or more TaskGroup Properties.
- One or more Task elements MAY appear at this level.
- Zero or one TaskDefaults elements MAY appear at this level.

The following illustrates this element's syntax:

```
<TaskGroup>
  <!-- TaskGroup Properties -->+
  <!-- Job Properties -->*
  <Task>+
  <TaskDefaults>?
```

</TaskGroup>

6.1 TaskGroup Properties

TaskGroup Properties apply to a particular task group or as default properties to encompassed task groups. These properties include the task group id, its tasks, task defaults, and other simple task group properties.

6.1.1 Simple TaskGroup Properties

Simple (unstructured) task group properties are enumerated in Table 6.

Table 6 Simple TaskGroup Properties

Element Name	Type	Description	Appearance	Compliance	Categories
TaskCount	Integer	Number of tasks in this taskgroup	MAY	MUST	
TaskGroupId	String	A task group identifier unique within the job.	MAY	MAY	
TaskGroupName	String	A task group name (such as "Master").	MAY	SHOULD	

6.1.2 Task

A task group MAY specify zero or more tasks.

See the next section for element details.

6.1.3 TaskDefaults

A task group MAY specify zero or more task defaults.

See the next section for element details.

7. Task and TaskDefaults Element

The Task and TaskDefaults elements have the same structure. A Task element contains information specific to a task (like the process id or the host it ran on). A TaskDefaults element may only appear within a TaskGroup (or TaskGroupDefaults) element and represents the defaults for all tasks within the task group. Task properties in Task elements override any properties found in a sibling TaskDefaults element.

- A compliant implementation MAY support the TaskGroup element.
- A compliant implementation MAY support the TaskGroupDefaults element.
- A task group MUST specify one or more TaskGroup Properties.
- One or more Task elements MAY appear at this level.
- Zero or one TaskDefaults elements MAY appear at this level.

The following illustrates this element's syntax:

```
<Task>  
  <!-- Task Properties -->+  
  <!-- Job Properties -->*  
</Task>
```

7.1 Task Properties

Task Properties are properties that apply to a particular task or as default properties to encompassed tasks. These properties include the task id and other task properties.

7.1.1 Simple Task Properties

Simple (unstructured) task properties are enumerated in Table 7.

Table 7 Simple Task Properties

Element Name	Type	Description	Appearance	Compliance	Categories
Node	String	Name of the node this task ran on.	MAY	MUST	
SessionId	Integer	Session id for the task group or job.	MAY	MAY	
TaskId	String	A task identifier unique	MAY	MAY	

		within the taskgroup.			
--	--	-----------------------	--	--	--

8. Property Categories

Certain properties need to be classified as being in a particular category. This is done when it is necessary to distinguish between a property that is requested versus a property that was delivered. When no such distinction is necessary, it is recommended that the property not be enveloped in one of these elements. In general, a property should be enveloped in a category element only if it is expected that the property will need to be attributed to more than one property category, or if it needs to make use of some of the special attributes inherited from the category.

8.1 Requested Element

A requested property reflects properties as they were requested. A disparity might occur between the requested value and the value delivered if a preference was expressed, if multiple options were specified, or if ranges or pattern matching was specified.

- A compliant implementation SHOULD support this element.

The following illustrates the syntax of this element:

```
<Requested>
  <!-- Requested Properties -->+
</Requested>
```

The following describes the attributes and elements for the example above:

```
/Requested
  This element is used to encapsulate requested properties.
/Requested/<Requested Property>
  Requested properties appear at this level.
```

Requested Properties inherit some additional attributes.

- A requested property MAY have an *op* attribute of type String that indicates a conditional operation on the value. A compliant implementation SHOULD support this attribute. Valid values for the *op* attribute include “EQ” meaning equals (which is the default), “NE” meaning not equal, “LT” meaning less than, “GT” meaning greater than, “LE” meaning less than or equal to, “GE” meaning greater than or equal to, “Match” which implies the value is a pattern to be matched.

- A requested property MAY have a *conj* attribute of type String that indicates a conjunctive relationship with the previous element. A compliant implementation MAY support this attribute. Valid values for the *conj* attribute include “And” (which is the default), “Or”, “Nand” meaning and not, and “Nor” meaning or not.
- A requested property MAY have a *group* attribute of type Integer that indicates expression grouping and operator precedence much like parenthetical groupings. A compliant implementation MAY support this attribute. A positive grouping indicates the number of nested expressions being opened with the property while a negative grouping indicates the number of nested expressions being closed with the property.
- A requested property MAY have a *preference* attribute of type Integer that indicates a preference for the property along with a weight (the weights are taken as a ratio to the sum of all weights in the same group). A compliant implementation MAY support this attribute. If a group of positive valued preference alternatives are specified, at least one of the preferences must be satisfied for the job to run. If a group of negative valued preferences are specified, the preferences will try to be met according to their weights but the job will still run even if it can’t satisfy any of the preferred properties. (Weight ranking can be removed by making all weights the same value (1 or -1 for example)).
- A requested property MAY have a *performanceFactor* attribute of type Float that provides a hint to the scheduler of what performance tradeoffs to make in terms of resources and start time. A compliant implementation MAY support this attribute.

The following are four examples of using Requested Properties:

```
<Requested>
  <Processors op="GE">8</Processors>
  <Processors op="LE">16</Processors>
  <WallclockDuration>3600</WallclockDuration>
</Requested>
```

```
<Requested>
  <NodeCount>1</NodeCount>
  <Nodes>
    <Node aggregation="Pattern">fr15.*</Node>
  </Nodeses>
</Requested>
```

```
<Requested>
  <UserId group="1">scottmo</UserId>
  <AccountName group="-1">mconfops</AccountName>
  <UserId conj="Or" group="1">monkeyboy</UserId>
  <AccountName group="-1">junglehunt</AccountName>
</Requested>
```

```
<Requested>
  <Memory preference="2">1024</Memory>
  <Memory preference="1">512</Memory>
```

</Requested>

8.2 Delivered Element

A delivered property reflects properties as they were actually utilized, realized or consumed. It reflects the actual amounts or values that are used, as opposed to a limit, choice or pattern as may be the case with a requested property.

- A compliant implementation SHOULD support this element.

The following illustrates the syntax of this element:

```
<Delivered>
  <!-- Delivered Properties -->+
</Delivered>
```

The following describes the attributes and elements for the example above:

```
/Delivered
  This element is used to encapsulate delivered properties.
/Delivered/<Delivered Property>
  Delivered properties appear at this level.
```

Delivered Properties inherit some additional attributes.

- A delivered property MAY have a **group** attribute of type Integer that indicates expression grouping and operator precedence much like parenthetical groupings. A compliant implementation MAY support this attribute. A positive grouping indicates the number of nested expressions being opened with the property while a negative grouping indicates the number of nested expressions being closed with the property. The purpose of this attribute would be to logically group delivered properties if they were used in certain aggregations (like a job that spanned machines).

The following are the same four examples distinguishing the delivered amounts and values:

```
<Delivered>
  <Processors>12</Processors>
  <WallclockDuration>1234</WallclockDuration>
</Delivered>
```

```
<Delivered>
  <Nodes>
    <Node>fr15n03</Node>
  </Nodes>
</Delivered>
```

```
<Delivered>  
  <UserId>scottmo</UserId>  
  <AccountName>mcsfops</AccountName>  
</Delivered>
```

```
<Delivered>  
  <Memory>1024</Memory>  
</Delivered>
```

9. AwarenessPolicy Attribute

A word or two should be said about compatibility mechanisms. With all the leeway in the specification with regard to implementing various portions of the specification, problems might arise if an implementation simply ignores a portion of a job specification that is critical to the job function in certain contexts. Given this situation, it might be desirable in some circumstances for jobs to be rejected by sites that fail to fully support that job's element or attributes. At other times, it might be desirable for a job to run, using a best-effort approach to supporting unimplemented features. Consequently, we define an *awarenessPolicy* attribute which can be added as an optional attribute to the Job element or any other containment or property element to indicate how the property (or the default action for the elements that the containment element encloses) must react when the implementation does not understand an element or attribute.

An awareness policy of "Reject" will cause the server to return a failure if it receives a client request in which it does not support an associated element name or attribute name or value. It is reasonable for an implementation to ignore (not even look for) an element or attribute that would not be critical to its function as long as ignoring this attribute or element would not cause an incorrect result. However, any element or attribute that was present that would be expected to be handled in a manner that the implementation does not support must result in a failure.

An awareness policy of "Warn" will accept the misunderstood element or attribute and continue to process the job object on a best effort basis. However a warning **MUST** be sent (if possible) to the requestor enumerating the elements and attributes that are not understood.

An awareness policy of "Ignore" will accept the unsupported element or attribute and continue to process the job object on a best effort basis. The action could be to simply ignore the attribute.

- This name of this attribute is *awarenessPolicy*.
- This attribute is of type String.
- This attribute can have values of "Reject", "Warn" or "Ignore".
- A compliant implementation **MAY** support this attribute.
- An implementation that does not support an attribute **MUST** reject any job object which contains elements or attributes that it does not support. Furthermore, it **SHOULD** return a message to the requestor with an indication of the element or attribute name it did not understand.

- This attribute MAY be present in a property or containment element.
- If an implementation does support the attribute, but it is absent, the default value of “Reject” is implied.
- Individual elements in the job object may override the containing object’s awareness policy default by including this attribute. For example, a job might specify an awarenessPolicy of “Reject” at its root (the Job element) but may want to allow a particular subset of elements or attributes to be ignored if not understood. Conversely, a job with a default awarenessPolicy of “Ignore” might want to classify a subset of its optional elements as “Reject” if they are indispensable to its correct interpretation. An implementation can opt to check or not check for this attribute at any level it wants but must assume a “Reject” policy for any elements it does not check.

10. References

ISO 8601

ISO (International Organization for Standardization). *Representations of dates and times*, 1988-06-15. <http://www.iso.ch/markete/8601.pdf>

DATATYPES

XML Schema Part 2: Datatypes. *Recommendation*, 02 MAY 2001. <http://www.w3.org/TR/xmlschema-2/>

Appendix A

Units of Measure Abbreviations

Abbreviation	Definition	Quantity
B	byte	1 byte
KB	Kilobyte	2^{10} bytes
MB	Megabyte	2^{20} bytes
GB	Gigabyte	2^{30} bytes
TB	Terabyte	2^{40} bytes
PB	Petabyte	2^{50} bytes
EB	Exabyte	2^{60} bytes
ZB	Zettabyte	2^{70} bytes
YB	Yottabyte	2^{80} bytes
NB	Nonabyte	2^{90} bytes
DB	Doggabyte	2^{100} bytes