Celerra XML API v2 Guide

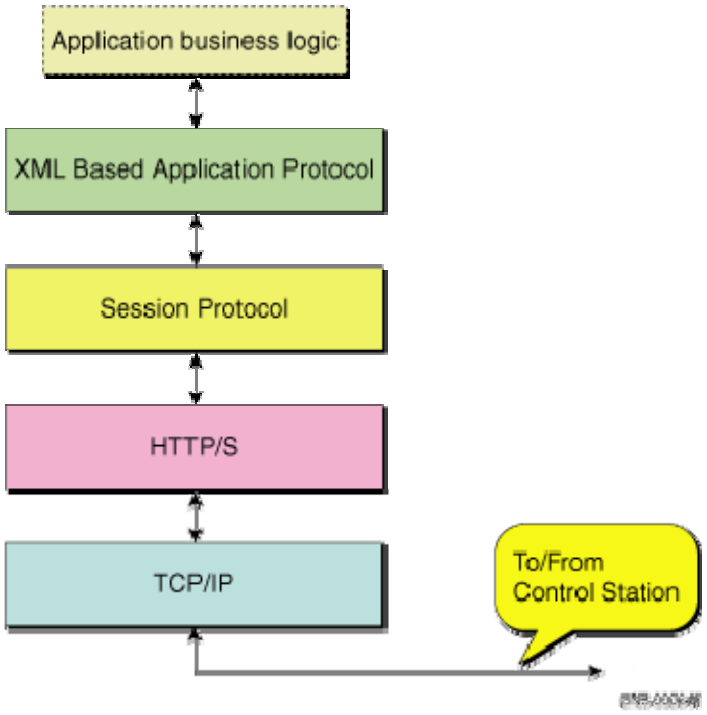
Table Of Contents

Introduction

The Celerra XML API is an interface for remotely managing and monitoring a Celerra Network Server. The interface uses XML formatted messages, and is programming language neutral. The message syntax is described in W3C XML Schema vocabulary. This allows application programmers to use a variety of tools to parse and transform XML structures into objects for the programming language they are using.

Client-side model

An application program typically runs on a client system connected to a Celerra Control Station by means of a TCP/IP network. The following diagram illustrates a typical application-side protocol stack:



All communications between the application program and the Control Station use the HTTP1.1 protocol with Secure Sockets (HTTPS) over TCP/IP. The Session Protocol uses HTTP POST functions to send request messages and receive responses from the Control Station. The XML payload of the request is carried in the body of the HTTP POST request, and the response is carried in the body of the HTTP POST reply.

An XML API application is associated with a session created for this application during the login process. After login, all requests performed by the Control Station on behalf of the application are executed in the context of this session, on behalf of the username with which the application program logged in. Session control information is carried within standard and application specific HTTP headers.

The Control Station can also send the application program one-way XML formatted messages called indications. Indication messages are carried within the reply of a never-ending HTTP GET

request, as separate chunks of HTTP1.1 chunked stream. The session layer always starts a GET request and keeps this connection open. If the connection is temporarily broken, the session protocol can recover it without data loss.

Note: In this release, EMC does not supply libraries that implement the session protocol. The Appendix provides a detailed description of the session level protocol.

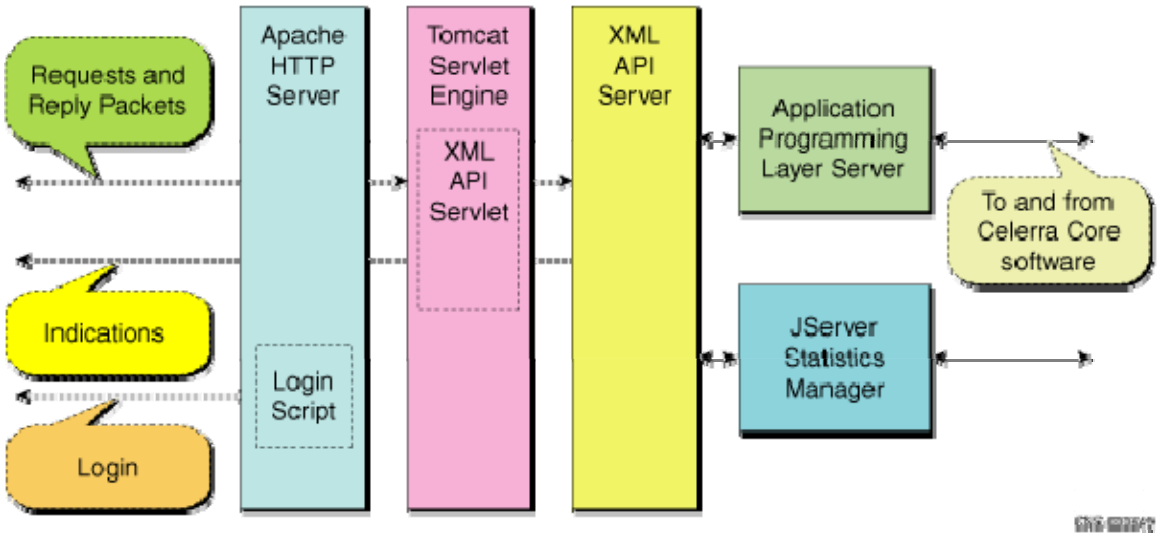
The XML-based application protocol defines the schema for request, response, and indication messages. Basically, a request (carried by the session layer in the body of the HTTP POST request) is a complete XML document that starts at the top with an element called <RequestPacket>. That, in turn, may contain a sequence of elements called <Request>. Each <Request> element contains a particular request, such as <Query>, <StartTask>, or <Subscribe>.

A response (carried in the body of the HTTP POST reply) contains a complete XML document that starts at the top with an element called <ResponsePacket>. That, in turn, may contain a sequence of elements called <Response>. Each <Response> element may contain either a response to the query, task status, or a response to some other request.

Indications (carried in the chunk of the HTTP GET reply) contain a complete XML document that starts at the top with an element called <IndicationPacket>. That, in turn, may contain <ConfigIndication>, <StatsIndication>, or <TaskIndication> elements. They carry structured information about changes in objects, statistical data, and task completions.

Server-side model

The following diagram provides a high-level overview of the architecture of the XML API software on the Control Station and will help in understanding subsequent material.



To work with the XML API, the client application must authenticate itself with the Celerra Control Station using a special login request. The response from the login script carries an HTTP cookie called Ticket in one of the HTTP headers. Each subsequent HTTP request from the user should send this cookie back to the HTTP server.

Now the application can send request packets to the XML API server using HTTP POST requests. The first request also establishes a session between the application and the XML API Servlet. Note, the XML API Server also knows about this session and performs most of the request on behalf of the user who was authenticated during the login request.

After a first request packet is sent (possibly an empty one), the application starts an indication connection with an HTTP GET request. Unless broken by the underlying network, the response to that request never ends. All indication packets are received by the application as chunks of data. The details of the protocol are described in Appendix: Session Level Protocol.

All requests within the request packet are executed sequentially by one thread. Requests from multiple packets are executed concurrently. The application, within limits, can send multiple packets concurrently, even using the same session.

The queries and active management requests are rerouted to the Application Programming Layer (APL) Server. For simplicity, assume that XML API Server repackages the queries and requests in APL format, verifies them, and sends them to the APL. Responses from the APL are repackaged back into XML API format and sent back to the application.

Statistics queries are rerouted to the JServer Statistics Manager. Similar to APL, they are repackaged back and forth from XML API format.

Both APL and JServer can generate indications. The APL generates indications on task completions and on changes in configuration. JServer generates indications each time it polls various Celerra components and generates performance metrics. The XML API server collects these indications and sends them out on a per-session basis and on the basis of subscriptions received from the clients.

XML API data model

The XML API is defined in terms of W3C XML Schema. All data and message objects are defined in the .xsd files. This section outlines the major principles behind the data model of the XML API, the conventions, and terminology.

The XML API is based on the concept of an object. All objects are defined in the XML API schema. Classes of objects are defined as W3C Schema complex types. Specific objects are defined as W3C Schema elements. According to API convention, elements have the same name as complex types; therefore, when the term object is used, it may mean either a complex type or an element. The distinction between object classes and objects is emphasized only when it is unclear.

All objects in the XML API data model are divided into two groups:

- Objects that have meaning within the XML API only. RequestPacket or QueryStats objects are examples. They carry information between the Celerra system and the XML API client application and have no meaning outside the XML API.
- Objects that have meaning within the Celerra system. They are called data objects, and reflect information such as a file system, volume, Data Mover configuration, or statistics.

Data objects are divided into two groups:

- Referenceable objects. These are top-level, primary objects, and they are called out as such in this document. Each primary object has a set of keys used to differentiate it from other objects of the same class.
- Non-referenceable objects. These occur only in the context of a primary object.

For each primary object, you can build an object called a reference. A reference is a set of keys that define this primary object. According to schema convention, reference object names end with the suffix "Ref". For example, the FileSystemRef object references a FileSystem object. All references are defined in the schema file Refs.xsd.

Data objects are not necessarily exact reflections of the appropriate Celerra data structures. A Celerra object may not exist in one specific place within the Celerra database, but its properties can be distributed between the Control Station and the Data Movers. At a particular time, some object properties may be visible and some may not. For example, basic properties of a Celerra system, such as name, ID, and volume size are always visible. However, properties such as logical size and number of inodes are visible only when the file system is mounted. Celerra object classification is not detailed. For the same object class, some properties may exist for one object instance and not exist for another. For example, for a raw file system, it is, in principle, impossible to know the number of inodes it has.

Because Celerra object classification is not detailed and to enhance usability, XML API data objects group related properties of the same Celerra object into separate XML API objects called aspects.

Aspects

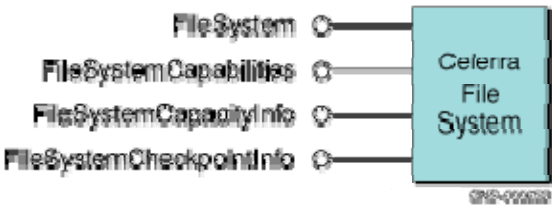
The concept of an aspect is not new. Aspects are also called facets. In Microsoft COM and Java they are known as interfaces. In the XML API, you can query for either a specific aspect or for all aspects of the same object in any combination. All aspects are objects, but not all objects are aspects. All aspects of an object have the same reference.

EMC recommends that client applications query for only the aspects it needs, since retrieving some aspects can be costly and time consuming.

For example, a file system can have up to four aspects and you may not need all of them:

- FileSystem is the base set of properties extracted from the Control Station database.
- FileSystemCapabilities is most often used separately from the base set of properties in a separate context, such as file system extension, and therefore as a separate aspect.
- FileSystemCapacityInfo properties can be obtained only by means of RPC. Therefore, these properties are placed in a separate aspect, and you need to be aware that a special effort is needed to obtain these properties.
- FileSystemCheckpointInfo is normally needed for checkpoint management only and is not queried frequently. It exists only for file systems that have checkpoints.

The following graphic illustrates the four aspects of a file system:



Schema packaging

The entire XML API schema is assembled from include files and can be parsed by starting from the top-level file APISchema.xsd. With few exceptions, the APISchema is divided into distinct schemas that describe distinct parts of the functionality. Each schema can be parsed separately. These schemas are called packages. FileSystem.xsd, Mover.xsd, and Cifs.xsd are examples of packages. A package contains object definitions relevant to its respective Celerra functionality. It is not necessary for all aspects of an object to reside in the same package. For example, Cifs.xsd schema defines an aspect CifsConfig. This object is an aspect of a Celerra mover object or a VDM object, but it is defined in the CIFS package.

The schema file BasicTypes.xsd is used by all packages. It defines mostly simple types and some shared complex types.

The schemas Request.xsd, Response.xsd, Indication.xsd, Task.xsd, Subscribe.xsd, Query.xsd, and QueryStats.xsd define the structure of the XML API request, response, and indication messages. These files define only objects that are XML API specific and do not represent a package.

Following is a list of all the schema files:

- APISchema.xsd
- BasicTypes.xsd
- Cifs.xsd
- Clariion.xsd
- Component.xsd
- FileSystem.xsd
- FileSystemStats.xsd
- Indication.xsd
- Iscsi.xsd
- Mount.xsd
- Mover.xsd
- MoverStats.xsd
- Nfs.xsd
- Query.xsd
- QueryStats.xsd
- Quota.xsd
- Refs.xsd
- Request.xsd
- Response.xsd
- StoragePool.xsd
- Subscribe.xsd
- Symmetrix.xsd
- Task.xsd
- Volume.xsd

Metadata within the schema

The XML API schema defines additional namespace with the namespace prefix "meta" to define metadata that cannot be defined within the W3C Schema. Currently, the following global attributes are used:

- meta:unit – Defines units in which this value is measured (if applicable). For example: <attribute name="time" type="api:UtcTime" meta:unit="sec"/> defines attribute time units as seconds. Since EMC does not supply meta schema at the present moment, for your convenience all possible unit values are enumerated and annotated within the file BasicTypes.xsd.
- meta:supported – Indicates that this property or object is not supported currently, but that EMC intends to implement this property or object in the future. By default, this attribute value is true.

Annotations

The API schema extensively annotates property names, class names, and so on, to enhance usability. However, the annotations are insufficient for learning about Celerra design. This document describes XML API-specific concepts.

Using Celerra XML API

The Celerra XML API provides the information necessary to write applications to manage and monitor a Celerra Network Server, including information about the following topics:

- Requests and responses
- Subscriptions and indications
- Volume management
- iSCSI management
- Mount management
- Mover management
- File system management
- Storage pool management
- Quota management
- NFS management
- CIFS management
- Component management
- CLARiiON management
- Symmetrix management
- Statistics management

For most of these tasks, you will find information about data objects, queries, and active management of your Celerra Network Server including activities such as creating, extending, deleting, or modifying a file system.

Requests and responses

The XML API application sends requests to the Control Station server in the RequestPacket element that is defined in the file Request.xsd. The corresponding response is returned in the ResponsePacket element that is defined in the file Response.xsd.

Request and Response Introduction

The XML API application sends requests to the Control Station server in the RequestPacket element which is defined in the file Request.xsd. The corresponding response is returned in the ResponsePacket element which is defined in the file Response.xsd.

The only attribute RequestPacket uses is apiVersion (except the global attribute of the top-level element that defines the namespace: xmlns). It tells the server the version of the XML API schema to use for responses to this and all subsequent requests. RequestPacket contains a series of elements called Request. Similarly, the ResponsePacket normally contains a series of elements called Response.

The Request element can have an attribute called clientHandle. The content of clientHandle is transparent to the server. It is returned to the application with the matching Response element in the response.

Request, in turn, contains request specifics of different types:

- Query – makes a query against Control Station databases and returns responses with the set of objects that match this query

- StartTask – starts a task on the Control Station and responds with the status of the submission
- QueryStats – makes a request against the historical database of recorded statistics and returns the set of specified samples
- Subscribe – makes a request to subscribe for notification about changes in objects or new instances of statistics
- Unsubscribe – makes a request to terminate the subscription for changes or new instances of statistics
- SetLocale – sets the locale for messages for this and subsequent replies

On the server side, the requests in the packet are executed sequentially and the responses to individual requests return in the same order. However, the applications should not rely on this because the XML API does not commit itself to sequential execution model.

In cases when the entire XML of the request packet is invalid and could not be parsed, the response packet will not contain individual responses, but rather an element called Fault. Individual responses within the response packet may contain the Fault elements as well.

Request packet examples

The following example shows a request packet that sends a single query request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api" >
  <Request clientHandle="381.72">
    <Query>
      <!-Query parameters go here -->
   </Query>
  </Request>
</RequestPacket>
```

A possible response to this request follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Response clientHandle="381.72">
   <QueryStatus maxSeverity="ok"/>
   <FileSystem containsSlices="true" internalUse="false" name="my_fs"
storagePools="1" storages="1" type="uxfs" volume="114" fileSystem="20">
   <RwFileSystemHosts mover="2" moverIdIsVdm="true"/>
   <ProductionFileSystemData cwormState="off"/>
   </FileSystem>
   <FileSystemCapacityInfo volumeSize="2" fileSystem ="20">
    <ResourceUsage filesTotal="7870" filesUsed="5010" spaceTotal="1000"
spaceUsed="823"/>
   </FileSystemCapacityInfo>
  </Response>
</ResponsePacket>
```

The following example shows a request packet containing multiple requests follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api" >
  <Request clientHandle="r.1">
    <StartTask timeout="0">
      <!—task parameters go here -->
    </StartTask>
  </Request>
  <Request clientHandle="r.2">
```

```
    <StartTask timeout="0">
      <!—task parameters go here -->
    </StartTask>
  </Request>
  <Request clientHandle="r.3">
   <Subscribe>
     <!—task parameters go here -->
   </Subscribe>
  </Request>
</RequestPacket>
```

In this example, the application submits two tasks and subscribes for some statistics or other indications. In both requests it sets the timeout value to 0, so it does not wait for the completion of both tasks. Subscriptions are always synchronous, but they are low overhead requests. This is an example of a reasonable use of multiple requests in the same request packet.

Some examples of unreasonable uses of multiple requests in the same packet follow:

- Starting multiple queries. This slows down the application, since queries are performed sequentially. To better use resources, start multiple request packets and make all queries run concurrently.
- Starting multiple tasks with nonzero timeouts, when the next task is dependent on the previous task. This can cause problems because if one request fails for any reason, subsequent requests are still executed.

Queries - query request

A query request makes a query against Control Station databases and replies with the set of objects that match this query. Queries are synchronous. That is, the reply contains the matching result to the query. All queries and the appropriate response are defined by the file Query.xsd.

In most cases, queries are passed for processing to APL server. In a few cases the query goes directly to the Control Station core software. The XML API defines a number of queries that could be done to the system. The type of the query is defined by query parameters.

The following example demonstrates a request packet that sends a single query request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api" >
   <Request clientHandle="381.72">
       <Query>
         <FileSystemQueryParams>
            <AspectSelection fileSystems="true" fileSystemCapacityInfos="true"
/>
            <Alias name="my_fs"/>
        </FileSystemQueryParams>
      </Query>
  </Request>
</RequestPacket>
```

In this example:

- The attribute xmlns of the RequestPacket element defines the namespace of the request packet. This attribute is required. Without it, the server is unable to parse the text.
- The attribute *apiVersion* of the RequestPacket element is missing. In this case, the server uses the version set in one of the previous packets. If it was never set, version "V1_0" is assumed.

- The attribute *clientHandle* of the Request element is set by the application to match the request to the corresponding reply and, possibly, to route the reply to the appropriate software module. The server does not interpret this value in any way.
- The element Query cannot have any attributes. Query requires parameters. In this example, the parameters are specified by the element FileSystemQueryParams.
- In FileSystemQueryParams the *AspectSelection* subelement specifies that only FileSystem and FileSystemCapacityInfo aspects are required.
- To narrow the set of replies to the set of data objects the application needs, query parameters may contain query filters. In this example, the application needs basic data- and capacity-related aspects of the file system named "my_fs."

The following example illustrates what the reply to this request might look like:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Response clientHandle="381.72">
    <QueryStatus maxSeverity="ok"/>
    <FileSystem containsSlices="true" internalUse="false" name="my_fs"
storagePools="1" storages="1"     type="uxfs" volume="114" fileSystem="20">
     <RwFileSystemHosts mover="2" moverIdIsVdm="true"/>
     <ProductionFileSystemData cwormState="off"/>
    </FileSystem>
    <FileSystemCapacityInfo volumeSize="2" fileSystem ="20">
      <ResourceUsage filesTotal="7870" filesUsed="5010" spaceTotal="1000"
spaceUsed="823"/>
    </FileSystemCapacityInfo>
  </Response>
</ResponsePacket>
```

In this example, the <Response> element contains a series of elements of different types. The element <QueryStatus> indicates the query was successful. Next, two requested aspects of the requested file system follow. Although the <QueryStatus> element is present in most query replies, its presence is not necessary. If the <QueryStatus> element is absent, it is assumed to be a success.

In cases when some Celerra components are disabled or not functioning properly and cannot be queried, the entire query can be incomplete. In such cases, the <QueryStatus> element can have *maxSeverity* attribute value set to `"warning"`. For example:

```
<QueryStatus maxSeverity="warning">
      <Problem component="API" facility="Generic" message="The query may be
incomplete or requested object not found" messageCode="18522112101"
severity="warning">
       <Diagnostics>Connection could not be established to virtual data mover
vdm7. The virtual data mover is not loaded.</Diagnostics>
      </Problem>
</QueryStatus>
```

In this example, the VDM named vdm7 is unloaded and cannot be queried. If the *maxSeverity* attribute value is not "ok", the <QueryStatus> element contains one or more elements called <Problem>. In the <Problem> element, the attributes component, facility, and messageCode are for Celerra support use. The value of attribute message is for application use. The content of the element <Diagnostics> is not for application use and it will not be internationalized. However, you can use this message to debug your software.

In some cases, when the query result set is empty (no matching object was found), the status of the query is a warning as well. For example, here is how the query status appears when the requested file system was not found:

```
<QueryStatus maxSeverity="warning">
      <Problem component="API" facility="Generic" message="The query may be
incomplete or requested object not found" messageCode="18522112101"
severity="warning">
        <Diagnostics>File system not found.</Diagnostics>
      </Problem>
      <Problem component="API" facility="Generic" message="The query may be
incomplete or requested object not found" messageCode="18522112101"
severity="warning">
        <Diagnostics>Migration file system not found.</Diagnostics>
      </Problem>
</QueryStatus>
```

This example demonstrates another important feature: There can be multiple <Problem> elements within one <QueryStatus> element. In this example, to find the file system, the server made two queries to the APL server and neither query found anything. This also explains the *maxSeverity* attribute's name; the value *maxSeverity* is computed as the worst severity among severities of all <Problem> elements.

The data objects within a query response can arrive in any order: The application should not assume any particular order, and, in fact, the order can vary from one API version to another.

Here is the full list of queries you can use to specify parameters:

- FileSystemQueryParams – to query file system aspects
- CheckpointQueryParams – to query checkpoints
- TreeQuotaQueryParams – to query tree quotas on file systems and their subdirectories
- UserQuotaQueryParams – to query user quotas on file systems and their subdirectories
- VolumeQueryParams – to query volumes
- StoragePoolQueryParams – to query storage pools
- MoverQueryParams – to query various mover aspects and other objects related to logical mover configuration
- VdmQueryParams – to query VDMs
- CifsConfigQueryParams – to query CIFS settings of movers and VDMs
- CifsShareQueryParams – to query CIFS shares on movers or VDMs
- CifsServerQueryParams – to query the CIFS server configuration on movers or VDMs
- NfsExportQueryParams – to query NFS exports on movers
- CelerraSystemQueryParams - to query general Celerra system data and status
- ControlStationQueryParams – to query the Control Station configuration and status
- StorageSystemQueryParams – to query the storage attached to this Celerra system
- MoverHostQueryParams – to query various physical aspects of Data Movers
- ClariionDeviceQueryParams – to query CLARiiON devices
- ClariionDiskQueryParams – to query CLARiiON disks
- ClariionGeneralConfigQueryParams – to query CLARiiON configuration information
- ClariionRaidGroupQueryParams – to query CLARiiON Raid groups
- ClariionSpQueryParams – to query CLARiiON storage processors
- IscsiLunQueryParams – to query iSCSI LUNs
- IscsiMaskQueryParams – to query iSCSI masks
- IscsiServiceQueryParams – to query iSCSI service
- IscsiSharedSecretQueryParams – to query shared secrets
- IscsiTargetQueryParams – to query iSCSI targets
- SymmDeviceQueryParams – to query Symmetrix devices

- SymmDirectorQueryParams – to query the Symmetrix director
- SymmGeneralConfigQueryParams – to query general Symmetrix configuration information
- SymmPhysicalDiskQueryParams – to query Symmetrix physical disks

Tasks – StartTask request

A task is a job submitted to the Control Station software. Tasks are typically executed by APL in a separate process running on behalf of the user who submits it. Tasks can change the state of Control Station databases or Data Movers. Tasks are not necessarily atomic. Even a failed task can cause side effects and result in state changes. When this happens, the application receives warning or error messages upon the completion of the task. Tasks can be started asynchronously or synchronously. If the timeout attribute of the StartTask element is set to 0, the server responds to the request immediately after the task is submitted. The response indicating the task has started provides the task ID that identifies the task. The application eventually receives a full completion status within the appropriate indication. All tasks and appropriate responses are defined in the file Task.xsd. The list of tasks follows:

- NewFileSystem – creates a new file system
- ExtendFileSystem – extends a file system for volumes or storage pools
- DeleteFileSystem – deletes a file system
- ModifyFileSystem – modifies file system properties
- NewCheckpoint – creates a new checkpoint of a file system
- DeleteCheckpoint – deletes a checkpoint
- RefreshCheckpoint – refreshes a checkpoint with the latest original file system data
- RestoreCheckpoint – restores a file system from a checkpoint
- NewMetaVolume – creates a new metavolume
- NewStripeVolume – creates a new stripe volume
- NewSliceVolume – creates a new slice volume
- DeleteVolume – deletes a volume
- NewTree – creates a Celerra tree
- ModifyTreeQuota – modifies tree quota on a tree
- ModifyTreeSettings – modifies user or group and tree quota settings
- NewUserQuota – creates a new user or group quota or user or group quotas for multiple users
- ModifyUserQuota – modifies a user or group quota
- DeleteTree – deletes a Celerra tree
- NewUserStoragePool – creates a user storage pool
- ModifyStoragePool – modifies properties of a storage pool
- ModifySystemStoragePool – modifies system storage pool properties
- DeleteStoragePool – deletes a user storage pool
- ExtendStoragePool – extends a storage pool with existing volumes
- ShrinkStoragePool – shrinks a storage pool by deleting some volumes from it
- ModifyMover – modifies properties of a mover
- ModifyVdm – modifies properties of a VDM
- NewVdm – creates a new VDM
- DeleteVdm – deletes a VDM
- ModifyMoverNisDomain – modifies or deletes an NIS domain of a mover
- NewMoverDnsDomain – creates a new DNS domain on a mover
- DeleteMoverDnsDomain – deletes a DNS domain on a mover
- NewMoverInterface – creates a new mover network interface (assigns IP address, mask, etc,. to a network device)
- DeleteMoverInterface – deletes a mover interface
- NewMoverRoute – adds an entry to a mover routing table
- DeleteMoverRoute – deletes an entry from a mover routing table
- NewEthernetChannelDevice – creates a new virtual network device of the type Ethernet channel

- NewLacpDevice – creates a new virtual network device of type IEEE LACP
- NewFsnDevice – creates a fail safe virtual network device
- ModifyLogicalNetworkDevice – modifies properties of a network device
- DeleteVirtualDevice – deletes a virtual network device
- NewCifsShare – creates a new CIFS share on a mover or VDM (exports a mounted file system or a directory by means of the CIFS protocol)
- ModifyCifsShare – modifies properties of a CIFS share
- DeleteCifsShare – deletes a CIFS share
- ModifyCifsConfig – modifies WINS and/or user mapper servers on a mover or VDM
- ModifyCifsEnabled – enables or disables CIFS service on a mover
- NewNT40CifsServer – creates a new logical CIFS server that emulates a NT4.0 servers
- NewW2KCifsServer – creates a new logical CIFS server that emulates a W2K servers
- NewStandaloneCifsServer – creates a stand-alone CIFS server
- ModifyNT40CifsServer – modifies properties of a NT4.0 emulating server
- ModifyW2KCifsServer – modifies properties of a W2K emulating server
- DeleteCifsServer – deletes a logical CIFS server
- NewNfsExport – creates a new NFS export on a mover (exports a mounted file system by means of the NFS protocol)
- ModifyNfsExport – modifies properties of an NFS export
- DeleteNfsExport – deletes an NFS export
- NewMount – mounts a file system
- ModifyMount – modifies various mount properties
- DeleteMount – unmounts a file system
- DeleteIscsiLun – deletes an iSCSI LUN
- DeleteIscsiMask – deletes an iSCSI mask
- DeleteIscsiSharedSecret – deletes an iSCSI shared secret
- DeleteIscsiTarget – deletes an iSCSI target
- ExtendIscsiLun – extends an iSCSI LUN
- ModifyIscsiMask – modifies an iSCSI mask
- ModifyIscsiServiceState – modifies the state of an iSCSI service
- ModifyIscsiSharedSecret – modifies an iSCSI shared secret
- ModifyIscsiTarget – modifies an iSCSI target
- ModifyIsnsConfig – modifies an ISNS configuration
- NewIscsiLun – creates a new iSCSI LUN
- NewIscsiMask – creates a new iSCSI mask
- NewIscsiSharedSecret – creates a new iSCSI shared secret
- NewIscsiTarget – creates a new iSCSI target

The following example illustrates task processing:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api" >
  <Request clientHandle="r.1">
    <StartTask timeout="0">
       <NewFileSystem name="tree_fs8" type="uxfs">
         <Mover mover="1" />
          <StoragePool pool="4" storage="1" size="10" mayContainSlices="true"/>
          <Mount path="/tree_fs8" ></Mount>
       </NewFileSystem>
   </StartTask>
 </Request>
</RequestPacket>
```

In this example, the application starts a task to create a new file system. It specifies the attribute timeout value of 0. This means the application may receive a response from the servers similar to the following:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Response clientHandle="r.1">
    <TaskResponse taskId="3771"/>
  </Response>
</ResponsePacket>
```

The response does not indicate if the task has completed. It merely indicates the task has been submitted. In some cases, however, when the timeout is set to a nonzero value, the response may contain an error that means the task failed:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Response clientHandle="r.1">
    <TaskResponse taskId="3771">
      <Status maxSeverity="error">
        <Problem component="APL" message="The following parameters are not
valid: filesystem tree_fs8 already exists." messageCode="13691191305"
severity="error"/>
      </Status>
    </TaskResponse>
  </Response>
</ResponsePacket>
```

If the timeout is set to a low value, the application is not guaranteed that a response means that the task has completed. Therefore, the applications that want to execute their tasks synchronously should set the timeout value relatively high. In any case, the task completes when the application receives the task indication (to which it is automatically subscribed). The indication may be similar to the following example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<IndicationPacket sequenceNumber="75" time="1135006132"
xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <TaskIndication description="New File System" endTime="1134988132"
failed="false" startTime="1134988123" taskId="3782" timeout="0"
userName="nasadmin">
    <Status maxSeverity="ok"/>
  </TaskIndication>
</IndicationPacket>
```

In the event of an error, the application receives an indication similar to the following:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<IndicationPacket sequenceNumber="75" time="1135005577"
xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <TaskIndication description="New File System" endTime="1134987577"
failed="true" startTime="1134987576" taskId="3780" timeout="0"
userName="nasadmin">
    <Status maxSeverity="error">
     <Problem component="APL" message="The following parameters are not valid:
filesystem tree_fs8 already exists." messageCode="13691191305"
severity="error"/>
    </Status>
  </TaskIndication>
</IndicationPacket>
```

Subscriptions and indications

An application can make a request to subscribe for notification about changes in objects or new instances of statistics. The actual data is received within the indication packet. All Subscribe requests complete synchronously. The schema file Subscribe.xsd defines Subscribe and Unsubscribe requests.

The following classes of indications are available:

- Notifications about changes in system configuration, such as adding, deleting, or modifying Celerra objects (such as volumes, file systems, and quotas)
- Notifications about completions of APL tasks
- Latest samples of statistics available from the JServer

Subscribing and unsubscribing to indications – subscribe and unsubscribe requests

To receive indications, the application needs to subscribe to them. To stop receiving indications, the application needs to unsubscribe to them. Issuing multiple subscription requests for the same event class has no effect, since each new subscription replaces the old one. For more information about indications, refer to Indications.

Subscribing for configuration changes

To subscribe to system configuration changes, the application sends the following request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api" >
  <Request clientHandle='abcdef'>
    <Subscribe>
      <ConfigChanges/>
    </Subscribe>
  </Request>
</RequestPacket>
```

The response, if everything is OK (meaning that the XML could be parsed), is an empty <Response> element that appears as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Response clientHandle="abcdef"/>
</ResponsePacket>
```

To unsubscribe, the application sends the following request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api" >
   <Request clientHandle='abcdef'>
     <Unsubscribe>
      <ConfigChanges/>
     </ Unsubscribe >
```

```
   </Request>
</RequestPacket>
```

The response follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <Response clientHandle="abcdef"/>
</ResponsePacket>
```

If the application has not subscribed for this event class, the unsubscribe operation has no effect.

Subscribing for task completions

By default, the application subscribes for task completions for tasks submitted under the username with which the application logged in. To subscribe for all task indications, the application needs to issue the following request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api" >
   <Request clientHandle='abcdef'>
     <Subscribe>
       <TaskCompletions/>
     </Subscribe>
   </Request>
</RequestPacket>
```

The structure of responses and the process of unsubscribing are similar to those in previous examples.

Subscribing for statistics samples

When subscribing to statistics, the application needs to specify the set of statistics it wants. The application can subscribe for the following statistics:

- Mover statistics
- Volume statistics
- File system usage statistics

For more information, see statistics.

You can subscribe for multiple statistics in one request, as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api" >
   <Request clientHandle='abcdef'>
     <Subscribe>
       <MoverStats statsSet='NFS-RPC' mover="1"/>
       <MoverStats statsSet='CIFS-Totals' mover="1"/>
       <MoverStats statsSet='Network-TCP' mover="1"/>
       <MoverStats statsSet='ResourceUsage' mover="1"/>
       <VolumeStats statsSet='Totals' mover="2"/>
       <FileSystemUsage fileSystem="30" />
     </Subscribe>
```

```
   </Request>
</RequestPacket>
```

Subscribing for multiple statistics is handy because subscriptions for statistics do not have optional parameters. Therefore, if you want to subscribe for Volume statistics for all movers, you need to explicitly subscribe for each mover in the system:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api" >
   <Request clientHandle='abcdef'>
     <Subscribe>
       <VolumeStats statsSet='Totals' mover="1"/>
       <VolumeStats statsSet='Totals' mover="2"/>
       <VolumeStats statsSet='Totals' mover="3"/>
       <VolumeStats statsSet='Totals' mover="4"/>
     </Subscribe>
   </Request>
</RequestPacket>
```

The structure of responses and the process of unsubscribing are identical to those in previous examples.

Indications

Note: EMC encourages you to use indications; however, at this time the complete indication functionality has not been fully verified or tested. For this release, Statistics and Tasks indications are supported. EMC is committed to the regular support of this functionality and indication testing will be included in a later release of the XML API.

The stream of indications travels in one direction only, from the Control Station to the client application. Indications are not synchronized with requests and responses. The application should be prepared to process them at any time. The application must not hold an indication socket without an outstanding read for too long, since internal TCP buffers and HTTP server buffers can overflow, causing the application to lose indications.

Indications are defined in the file Indication.xsd. Any indication is wrapped in the element called IndicationPacket. An IndicationPacket element can carry three types of elements: ConfigIndication, TaskIndication, and StatsIndication.

Indications for changes to the configuration – ConfigIndication

As the result of various management operations, the Celerra system database might experience the following transitions:

- An object or a group of objects could be added to the database.
- The properties of an existing object or a group of objects could be modified.
- An object or a group of objects could be deleted.

The transitions happen within distributed database transactions. Each ConfigIndication reflects one transaction.

Within one ConfigIndication, there are often multiple elements:

- Added – the object contained within the Added element has been added to the database.

- Modified – the properties of object contained within the Modified element have been modified. Note, it is unknown which properties have been modified. If the client application keeps a cache of objects, this can be known by comparison of the old object with the newly arrived one.
- Deleted – the object specified by the reference contained within the Deleted object has been deleted.
- Invalidated – the system, for one or more reasons, suggests the application invalidate certain classes of objects and refetch them from the server.

The following example illustrates the stream of indication packets associated with an operation to delete a file system. The example assumes the application subscribed for configuration change indications. By default, the application also subscribed for task completions of tasks that it had submitted.

First, the system receives an indication that the file system has been unmounted. Note that this is a separate transaction against the internal Celerra database. Each configuration change indication reflects a database transaction. This particular indication shows that to delete a mounted file system, it first needs to be unmounted. The attribute sequenceNumber is a sequence number of the indication. Sequence numbers are incremented by 1 for each successive indication. Sequence numbers are per user session. The indications that arrive at the application need to be applied to the client caches in sequence.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<IndicationPacket sequenceNumber="125" time="1135096729"
xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <ConfigIndication>
    <Modified>
      <FileSystem containsSlices="true" internalUse="false" name="tree_fs9"
storagePools="1" storages="1" type="uxfs" volume="198" fileSystem="70">
        <ProductionFileSystemData cwormState="off"/>
      </FileSystem>
    </Modified>
    <Deleted>
      <Mount path="/tree_fs9" mover="1" moverIdIsVdm="false"/>
    </Deleted>
  </ ConfigIndication >
</ IndicationPacket>
```

The next indication shows the actual transaction for deleting the file system. It deletes the file system and underlying volumes. The deleted volumes also change the structure of the volume from which they were carved out.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
< IndicationPacket sequenceNumber="126" time="1135096731"
xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <ConfigIndication>
    <Deleted>
      <FileSystem fileSystem="70"/>
    </Deleted>
    <Invalidate>
      <AspectSelection quotas="true"/>
    </Invalidate>
    <Deleted>
      <Volume volume="198"/>
    </Deleted>
    <Deleted>
      <Volume volume="197"/>
    </Deleted>
    <Modified>
      <Volume clientVolumes="113 115 119 137 140 192" name="v111" size="69048"
storagePool="1" type="stripe" volume="111">
```

```
       <StripeVolumeData stripeSize="8" stripedVolumes="8 9 10 11 12 13 14
15"/>
      </Volume>
    </Modified>
  </ ConfigIndication >
</ IndicationPacket>
```

Indications for task completions – TaskIndication

The stream of indications related to the task of deleting a file system described in the previous section completes with the following indication packet carrying the TaskIndication element:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
< IndicationPacket sequenceNumber="127" time="1135096731"
xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <TaskIndication description="Delete File System" endTime="1135078731"
failed="false" startTime="1135078727" taskId="3791" timeout="0"
userName="nasadmin">
    <Status maxSeverity="ok"/>
  </TaskIndication>
</ IndicationPacket>
```

The indication packet specifies the following information about the task:

- It was started asynchronously (`timeout="0"`) by user "nasadmin" and completed successfully (`failed="false"`).
- It started at 1135078727 and ended at 1135078731 (Control Station UTC time in seconds).
- The task ID is 3791. It can be matched with the task ID the user application receives in response when it submits the task.
- It completes without any problems (`maxSeverity="ok"`)

Some tasks may succeed but have warnings and/or info messages. The following example illustrates the completion of the task to create a new W2K CIFS server in which domain name was misspelled. Also, during the task execution, the example shows that CIFS services were not enabled on this mover, so the client is notified about this too:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Indication sequenceNumber="176" time="1135305093"
xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <TaskIndication description="New Cifs Server" endTime="1135287093"
failed="false" startTime="1135287090" taskId="12" timeout="0"
userName="nasadmin">
    <Status maxSeverity="warning">
      <Problem component="APL" message="Join failed. System was unable to join
the CIFS server to the domain." messageCode="17986748527" severity="warning"/>
      <Problem component="APL" message="You may need to enable the CIFS service
if it is not running." messageCode="26576683104" severity="info"/>
    </Status>
  </TaskIndication>
</Indication>
```

Indications for new samples of statistics – StatsIndication

An application that subscribes to receive fresh samples of statistics receives indications such as the following StatsIndication. The following example shows the latest sample of CPU and memory utilization for mover 2:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
< IndicationPacket sequenceNumber="9" time="1135103769"
xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <StatsIndication>
    <MoverResourceUsage mover="2">
      <Sample cpu="1.0" mem="17.15908" stamp="720" time="1135103767"/>
    </MoverResourceUsage>
  </StatsIndication>
</IndicationPacket>
```

The following example shows the totals for volume I/O on mover 2:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
< IndicationPacket sequenceNumber="10" time="1135103769"
xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <StatsIndication>
    <VolumeSet mover="2">
      <Sample stamp="720" time="1135103767">
        <Totals bytesRead="40993280" bytesWritten="1119639552" reqsRead="10897"
reqsWritten="129587"/>
      </Sample>
    </VolumeSet>
  </StatsIndication>
</IndicationPacket>
```

Volume management

The schema file Volume.xsd defines data structures and operations that are related to Celerra volumes. A Celerra volume object is a container that may or may not contain raw data. Volumes have hierarchical structure. In other words, a volume can be built from other volumes by slicing, striping, and combining them. User visible volumes can be of the following types:

- Disk volume – a Celerra representation of the underlying storage system logical device
- Metavolume – an aggregation of other volumes
- Stripe volume – a volume made up of stripes of storage allocated from the other volumes
- Slice volume – a slice of storage allocated from some other volume
- Pool volume – a special volume allocated for user data and management of checkpoints of a file system

Volume data objects

A volume object is identified by a numeric-string ID. A special object VolumeRef represents a reference to a volume. The XML API uses either volume IDs or references to refer to a particular volume, depending on the context.

An object Volume describes the configuration of a volume.

Volume aspect

This object describes the volume configuration. Most of the properties of this object are relevant to any volume type. Type-specific properties are concentrated in special, non-primary objects that appear as elements within the schema definition of the Volume object. DiskVolumeData, MetaVolumeData, StripeVolumeData, SliceVolumeData, and PoolVolumeData all contain corresponding type specific properties.

The following example shows a fragment from the query response that demonstrates variations of the Volume object:

```
<Volume clientVolumes="82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 100"
            name="root_ldisk" size="11619" type="disk" volume="2">
           <DiskVolumeData diskType="std" lun="0021" movers="1 2"
storageSystem="1"/>
</Volume>
<Volume clientVolumes="" name="root_volume_2" size="128" type="meta"
volume="37">
           <MetaVolumeData clientFileSystems="2" memberVolumes="36 102"/>
</Volume>
<Volume clientVolumes="101 102 103 104" name="root_ldisk_reserve" size="10595"
type="slice" volume="100">
          <SliceVolumeData offset="1024" slicedVolume="2"/>
          <FreeSpace offset="240" size="10355"/>
</Volume>
<Volume clientVolumes="113 115 119 137 140" name="v111" size="69048"
storagePool="1" type="stripe" volume="111">
          <StripeVolumeData stripeSize="8" stripedVolumes="8 9 10 11 12 13 14
15"/>
          <FreeSpace offset="12" size="10"/>
          <FreeSpace offset="160" size="68888"/>
</Volume>
<Volume clientVolumes="" name="vp181" size="8631" type="pool" volume="181">
          <PoolVolumeData clientFileSystems="25" memberVolumes="180"/>
</Volume>
```

Volume queries

The following section describes volume queries.

Note: On a system with a large number of volumes, the query may take considerable time. Therefore, the client application may need to keep a cache of the objects it needs.

VolumeQueryParams specifies parameters for retrieval of Volume objects.

Query filters include:

- A volume ID to fetch a specific volume
- A storage ID to fetch all volumes allocated on this particular storage
- A flag that tells the XML API to fetch only volumes that have available space

Retrieving a specific volume object

The following examples show a request to retrieve a specific Volume object and the corresponding response:

Request:

```
<?xml version="1.0"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <Query>
            <VolumeQueryParams volume="96"/>
        </Query>
```

```
    </Request>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Response>
        <QueryStatus maxSeverity="ok"/>
        <Volume clientVolumes="98" name="root_ufslog_15" size="64" type="slice"
volume="96">
            <SliceVolumeData offset="896" slicedVolume="2"/>
        </Volume>
    </Response>
</ResponsePacket>
```

Retrieving all volumes

The following examples show a request that queries all volumes and the corresponding response:

Request:

```
<?xml version="1.0"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <Query>
            <VolumeQueryParams/>
        </Query>
    </Request>
</RequestPacket>
```

Response:

Retrieving_all_volumes_example

Retrieving all volumes that have free space

The following examples show a request that queries all volumes that have free space and the corresponding response:

Request:

```
<?xml version="1.0"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <Query>
            <VolumeQueryParams  hasAvailableSpace='true'/>
        </Query>
    </Request>
</RequestPacket>
```

Response:

Retrieving_all_volumes_free_space_example

Volume active management

Using the XML API you can do the following:

- Create a metavolume, slice volume, or a stripe volume
- Delete a volume

Note: You cannot create disk and pool volumes directly using the XML API. Disk volumes are normally created using special CLI commands when the new storage is attached to the Celerra. Pool volumes are created only as a side effect of NewCheckpoint operation.

Creating a metavolume

Create a metavolume by starting a request to submit the task NewMetaVolume.

The following example shows a request to create a metavolume from disk volume:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
     <StartTask>
         <NewMetaVolume  name="mv1" memberVolumes="53 54"/>
     </StartTask>
   </Request>
</RequestPacket>
```

Creating a stripe volume

Create a stripe volume by starting a request to submit the task NewStripeVolume.

The following example shows a request to create a stripe from a set of volumes:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask>
            <NewStripeVolume name="test_volume3" size ="16" stripeSize="256"
stripedVolumes="70 71"/>
        </StartTask>
    </Request>
</RequestPacket>
```

Creating a slice volume

Create a slice volume by starting a request to submit the task NewSliceVolume.

The following example shows a request to create a slice volume from a volume:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask>
```

```
            <NewSliceVolume name="test_volume2" size ="16" slicedVolume="108"
/>
        </StartTask>
    </Request>
</RequestPacket>
```

Deleting a volume

Delete a volume by supplying the volume ID within the task DeleteVolume.

The following example shows a request to delete a volume:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask>
            <DeleteVolume volume="307" recursive="true" />
        </StartTask>
    </Request>
</RequestPacket>
```

iSCSI management

iSCSI management

The schema file Iscsi.xsd defines data structures and queries that support iSCSI targets and initiators.

iSCSI data objects

An iSCSI target data object is identified by the mover and the unique target iSCSI name. A special object, IscsiTargetRef, represents a reference to the iSCSI target.

iSCSI target aspect

The target aspect describes the basic iSCSI target attributes such as the target alias, list of portal groups that contain portals to which this target belongs, set of LUNs defined on this target, and number of initiators currently connected to this target.

```
<IscsiTarget initiatorCount="0" nameAlias="XMLAPITEST" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-4">
       <Luns start="99"/>
       <Luns start="100"/>
</IscsiTarget>
```

iSCSI target initiator object

The target initiator aspect describes the name of the initiators currently connected to a specific target.

```
<IscsiTargetInitiators mover="1"  name="iqn.1992-05.com.emc:0002806001360000-
4">
      <Initators>
           <li>iqn.1992-05.com.emc:0002806001360000-5</li>
           <li>iqn.1992-05.com.emc:0002806001360000-6</li>
      </Initators>
</IscsiTargetInitiators>
```

iSCSI LUN object

The iSCSI LUN object is identified by the mover, target iSCSI name, and LUN number. A special object, IscsiLunRef, represents a reference to the iSCSI LUN.

The iSCSI LUN object describes the basic iSCSI LUN attributes such as the ID of the file system containing the LUN, whether the LUN is mapped to the entire file, size of the LUN, maximum possible extension size for the LUN, virtual position, and amount of space owned by the LUN's file.

```
<IscsiLun fileSystem="74" mappedToFile="false" maxExtension="118061" size="20"
spaceAllocated="0" virtuallyProvisioned="false" lun="99" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-4"/>
<IscsiLun fileSystem="74" mappedToFile="false" maxExtension="118061" size="20"
spaceAllocated="0" virtuallyProvisioned="false" lun="100" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-4"/>
```

iSCSI mask object

The iSCSI mask object is defined for the initiators (computers connected to target). An iSCSI mask is identified by the mover, target iSCSI name, and initiator. A special object, IscsiMaskRef, represents a reference to the iSCSI mask.

The iSCSI mask objects describes the set of LUNs for which access is granted for the initiator.

```
<IscsiMask initiator="iqn.1992-05.com.emc:0002806001360000-4" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-4">
       <GrantAccess size="2" start="99"/>
</IscsiMask>
```

iSCSI shared secret object

The iSCSI shared secret object defines the CHAP secret between the mover and initiator or reverse authentication (reverse secret) between initiator and mover. An iSCSI shared secret is identified by the mover and initiator. A special object, IscsiSharedSecretRef, represents a reference to the iSCSI shared secret.

The iSCSI shared secret object describes the initiator for which shared secret is set within that mover.

Reverse Authentication is specified for a mover and is common for all initiators.

```
<IscsiSharedSecret initiator="iqn.1992-05.com.emc:0002806001360000-4"
mover="1"/>
<IscsiSharedSecret initiator="reverseAuthentication" mover="1"/>
```

iSCSI ISNS configuration object

The iSCSI ISNS configuration object represents the ISNS server configuration. An iSCSI ISNS configuration is identified by the mover. A special existing object, MoverRef, represents a reference to the iSCSI ISNS configuration.

The iSCSI ISNS configuration object describes the ISNS server IP address, port, and entry status inquiry port number.

```
<IscsiIsnsConfig address="192.168.50.55" esiPort="200" port="4280" mover="1"/>
```

iSCSI service state object

The iSCSI service state object represents the service state of iSCSI for the specific mover. An iSCSI service state is identified by the mover. A special existing object, MoverRef, represents a reference to the iSCSI service state.

The iSCSI service state object describes the status of iSCSI service and ISNS server status on the mover.

```
<IscsiServiceState enabled="true" mover="1">
     <IsnsStatus maxSeverity="error">
          <Problem component="APL" message="cannot contact iSNSserver."
messageCode="13691256838" severity="error"/>
     </IsnsStatus>
 </IscsiServiceState>
<IscsiServiceState enabled="false" mover="2">
     <IsnsStatus maxSeverity="info">
          <Problem component="APL" message="not in use."
messageCode="26576158727" severity="info"/>
     </IsnsStatus>
</IscsiServiceState>
```

iSCSI queries

The following section describes iSCSI queries.

Querying iSCSI targets

The IscsiTargetQueryParams object specifies parameters for retrieving all aspects of an iSCSI target. Objects returned are either of type IscsiTarget or IscsiTargetInitiator depending on aspect of the query.

Query filters include:

- An Aspect selection element specifying aspect needed by the application program
- A mover ID
- A target iSCSI name

The following examples are for target aspect. If you select the target initiator aspect, the response contains IscsiTargetInitiators objects. The examples show a request that queries all iSCSI targets and the corresponding response:

Request

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
      <Query>
          <IscsiTargetQueryParams>
          <!-- also test by setting initiators in AspectSelection and not
attribute in AspectSelection-->
 <AspectSelection targets="true" ></AspectSelection>
          </IscsiTargetQueryParams>
      </Query>
   </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx clientHandle="abc">
       <QueryStatus maxSeverity="ok"/>
       <IscsiTarget initiatorCount="0" nameAlias="XMLAPITEST" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-4">
           <Luns start="99"/>
           <Luns start="100"/>
       </IscsiTarget>
       <IscsiTarget initiatorCount="0" nameAlias="XMLAPITEST2" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-5"/>
       <IscsiTarget initiatorCount="0" nameAlias="coret1" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-6">
           <Luns start="1"/>
       </IscsiTarget>
       <IscsiTarget initiatorCount="0" nameAlias="coret2" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-7">
           <Luns start="2"/>
       </IscsiTarget>
       <IscsiTarget initiatorCount="0" nameAlias="coret3" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-8"/>
       <IscsiTarget initiatorCount="0" nameAlias="coret0" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-9">
           <Luns start="0"/>
       </IscsiTarget>
       <IscsiTarget initiatorCount="0" nameAlias="t01" mover="2"
name="iqn.1992-05.com.emc:0002806001360000-1"/>
   </ResponseEx>
</ResponsePacket>
```

The following examples show a request that queries all targets for a given mover and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
```

```
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
        <Query>
<!-- set mover id , determine using CLI -->
<IscsiTargetQueryParams mover="1">
<!-- also test by setting initiators in AspectSelection and not attribute in
AspectSelection-->
    <AspectSelection targets="true"></AspectSelection>
</IscsiTargetQueryParams>
         </Query>
      </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx clientHandle="abc">
       <QueryStatus maxSeverity="ok"/>
       <IscsiTarget initiatorCount="0" nameAlias="XMLAPITEST" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-4">
           <Luns start="99"/>
           <Luns start="100"/>
       </IscsiTarget>
       <IscsiTarget initiatorCount="0" nameAlias="XMLAPITEST2" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-5"/>
       <IscsiTarget initiatorCount="0" nameAlias="coret1" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-6">
           <Luns start="1"/>
       </IscsiTarget>
       <IscsiTarget initiatorCount="0" nameAlias="coret2" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-7">
           <Luns start="2"/>
       </IscsiTarget>
       <IscsiTarget initiatorCount="0" nameAlias="coret3" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-8"/>
       <IscsiTarget initiatorCount="0" nameAlias="coret0" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-9">
           <Luns start="0"/>
       </IscsiTarget>
   </ResponseEx>
</ResponsePacket>
```

The following examples show a request that queries a specific target for a specific mover and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
       <Query>
           <!-- set mover id & target name, determine using CLI -->
 <IscsiTargetQueryParams mover="1" target="iqn.1992-
05.com.emc:0002806001360000-4">
```

```
 <!-- also test by setting initiators in AspectSelection and not attribute in
AspectSelection-->
    <AspectSelection targets="true"></AspectSelection>
</IscsiTargetQueryParams>
       </Query>
  </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx clientHandle="abc">
       <QueryStatus maxSeverity="ok"/>
       <IscsiTarget initiatorCount="0" nameAlias="XMLAPITEST" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-4">
           <Luns start="99"/>
           <Luns start="100"/>
       </IscsiTarget>
   </ResponseEx>
</ResponsePacket>
```

Querying iSCSI LUN

The IscsiLunQueryParams object specifies parameters for retrieving an IscsiLun object.

Query filters include:

- A mover ID
- A target iSCSI name
- A LUN number

The following examples show a request that queries for all iSCSI LUNs and the corresponding response:

Request:

```
 <?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
        <Query>
<IscsiLunQueryParams></IscsiLunQueryParams>
        </Query>
   </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx clientHandle="abc">
       <QueryStatus maxSeverity="ok"/>
```

```
       <IscsiLun fileSystem="74" mappedToFile="false" maxExtension="118061"
size="20" spaceAllocated="0" virtuallyProvisioned="false" lun="99" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-4"/>
       <IscsiLun fileSystem="74" mappedToFile="false" maxExtension="118061"
size="20" spaceAllocated="0" virtuallyProvisioned="false" lun="100" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-4"/>
       <IscsiLun fileSystem="120" mappedToFile="false" maxExtension="47"
size="50" spaceAllocated="0" virtuallyProvisioned="false" lun="1" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-6"/>
       <IscsiLun fileSystem="121" mappedToFile="false" maxExtension="47"
size="50" spaceAllocated="0" virtuallyProvisioned="false" lun="2" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-7"/>
       <IscsiLun fileSystem="118" mappedToFile="false" maxExtension="47"
size="50" spaceAllocated="0" virtuallyProvisioned="false" lun="0" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-9"/>
   </ResponseEx>
</ResponsePacket>
```

The following examples show a request that queries all LUNs for a given mover and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
  <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
     <Query>
         <!-- set mover id, determine using CLI -->
<IscsiLunQueryParams mover="1"></IscsiLunQueryParams>
      </Query>
  </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx clientHandle="abc">
       <QueryStatus maxSeverity="ok"/>
       <IscsiLun fileSystem="74" mappedToFile="false" maxExtension="118061"
size="20" spaceAllocated="0" virtuallyProvisioned="false" lun="99" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-4"/>
       <IscsiLun fileSystem="74" mappedToFile="false" maxExtension="118061"
size="20" spaceAllocated="0" virtuallyProvisioned="false" lun="100" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-4"/>
       <IscsiLun fileSystem="120" mappedToFile="false" maxExtension="47"
size="50" spaceAllocated="0" virtuallyProvisioned="false" lun="1" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-6"/>
       <IscsiLun fileSystem="121" mappedToFile="false" maxExtension="47"
size="50" spaceAllocated="0" virtuallyProvisioned="false" lun="2" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-7"/>
       <IscsiLun fileSystem="118" mappedToFile="false" maxExtension="47"
size="50" spaceAllocated="0" virtuallyProvisioned="false" lun="0" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-9"/>
   </ResponseEx>
```

```
</ResponsePacket>
```

The following examples show a request that queries all LUNs for a given target and mover and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
      <Query>
         <!-- set mover id, target, determine using CLI -->
        <IscsiLunQueryParams mover="1" target="iqn.1992-
05.com.emc:0002806001360000-4" ></IscsiLunQueryParams>
      </Query>
   </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx clientHandle="abc">
       <QueryStatus maxSeverity="ok"/>
       <IscsiLun fileSystem="74" mappedToFile="false" maxExtension="118061"
size="20" spaceAllocated="0" virtuallyProvisioned="false" lun="99" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-4"/>
       <IscsiLun fileSystem="74" mappedToFile="false" maxExtension="118061"
size="20" spaceAllocated="0" virtuallyProvisioned="false" lun="100" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-4"/>
   </ResponseEx>
</ResponsePacket>
```

The following examples show a request that queries specific LUNs for a given target, mover, and LUN and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
       <Query>
<!-- set mover id, target, lun, determine using CLI -->
<IscsiLunQueryParams mover="1" target="iqn.1992-05.com.emc:0002806001360000-4"
lun="99"></IscsiLunQueryParams>
        </Query>
      </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx clientHandle="abc">
       <QueryStatus maxSeverity="ok"/>
       <IscsiLun fileSystem="74" mappedToFile="false" maxExtension="118061"
size="20" spaceAllocated="0" virtuallyProvisioned="false" lun="99" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-4"/>
   </ResponseEx>
</ResponsePacket>
```

Querying iSCSI mask for all initiators for a specific target

The IscsiMaskQueryParams object specifies parameters for retrieving IscsiMask objects.

The following examples show a request that queries for a specific iSCSI target mask and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
    <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine
name or IP-->
         <Query>
<!-- Set mover id, target, determine using CLI -->
<IscsiMaskQueryParams target="iqn.1992-05.com.emc:0002806001360000-4"
mover="1"></IscsiMaskQueryParams>
         </Query>
    </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx clientHandle="abc">
       <QueryStatus maxSeverity="ok"/>
       <IscsiMask initiator="iqn.1992-05.com.emc:0002806001360000-4" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-4">
           <GrantAccess size="2" start="99"/>
       </IscsiMask>
   </ResponseEx>
</ResponsePacket>
```

Querying for iSCSI shared secret (CHAP password)

The IscsiSharedSecretQueryParams object specifies the parameters for retrieving IscsiSharedSecret objects.

Query filters include:

- A mover ID
- Initiator name

The following examples show a request that queries all iSCSI shared secrets and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
      <Query>
         <IscsiSharedSecretQueryParams></IscsiSharedSecretQueryParams>
      </Query>
   </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx clientHandle="abc">
       <QueryStatus maxSeverity="ok"/>
       <IscsiSharedSecret initiator="iqn.1992-05.com.emc:0002806001360000-4"
mover="1"/>
       <IscsiSharedSecret initiator="reverseAuthentication" mover="1"/>
   </ResponseEx>
</ResponsePacket>
```

The following examples show a request that queries all shared secrets for a given mover and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
  <RequestEx clientHandle="abc"> <!--Set the clientHandle param to machine name
or IP-->
     <Query>
        <!-- set mover id, determine using CLI -->
<IscsiSharedSecretQueryParams mover="1"></IscsiSharedSecretQueryParams>
     </Query>
  </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx clientHandle="abc">
```

```
       <QueryStatus maxSeverity="ok"/>
       <IscsiSharedSecret initiator="iqn.1992-05.com.emc:0002806001360000-4"
mover="1"/>
       <IscsiSharedSecret initiator="reverseAuthentication" mover="1"/>
   </ResponseEx>
</ResponsePacket>
```

The following examples show a request that queries all shared secrets for a given mover and initiator and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
  <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
     <Query>
        <!-- set mover id, initiator, determine using CLI -->
        <IscsiSharedSecretQueryParams mover="1" initiator="iqn.1992-
05.com.emc:0002806001360000-4"></IscsiSharedSecretQueryParams>
      </Query>
  </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx clientHandle="abc">
       <QueryStatus maxSeverity="ok"/>
       <IscsiSharedSecret initiator="iqn.1992-05.com.emc:0002806001360000-4"
mover="1"/>
   </ResponseEx>
</ResponsePacket>
```

Querying for both iSCSI service status and ISNS configuration

The IscsiServiceQueryParams object specifies parameters for retrieving all aspects of iSCSI services. Objects returned are either of type IscsiIsnsConfig or IscsiServiceState depending on the aspect specified in the query.

Query filters include:

- An Aspect selection element specifying aspect needed by the application program
- A mover ID

The following examples show a request that queries all iSCSI service objects and the corresponding response:

Request:

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

```
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
      <Query>
           <IscsiServiceQueryParams>
<!-- also test by setting isnsConfig in AspectSelection and by both -->
        <AspectSelection isnsConfig="true" iscsiState="true"></AspectSelection>
           </IscsiServiceQueryParams>
        </Query>
    </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx clientHandle="abc">
       <QueryStatus maxSeverity="warning">
           <Problem component="API" facility="Generic" message="The query may
be incomplete or requested object not found" messageCode="18522112101"
severity="warning">
               <Diagnostics>The requested action is not allowed on
server_4</Diagnostics>
           </Problem>
       </QueryStatus>
       <IscsiIsnsConfig address="192.168.50.55" esiPort="200" port="4280"
mover="1"/>
       <IscsiServiceState enabled="true" mover="1">
           <IsnsStatus maxSeverity="error">
               <Problem component="APL" message="cannot contact iSNSserver."
messageCode="13691256838" severity="error"/>
           </IsnsStatus>
       </IscsiServiceState>
       <IscsiIsnsConfig mover="2"/>
       <IscsiServiceState enabled="false" mover="2">
           <IsnsStatus maxSeverity="info">
               <Problem component="APL" message="not in use."
messageCode="26576158727" severity="info"/>
           </IsnsStatus>
       </IscsiServiceState>
   </ResponseEx>
</ResponsePacket>
```

The following examples show a request that queries all service objects for a given mover and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
      <Query>
         <!-- set mover id, determine from CLI -->
         <IscsiServiceQueryParams mover="1">
```

```
              <!-- also test by setting iscsiState in AspectSelection and by
both -->
 <AspectSelection iscsiState="true"></AspectSelection>
         </IscsiServiceQueryParams>
      </Query>
   </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx clientHandle="abc">
       <QueryStatus maxSeverity="warning">
           <Problem component="API" facility="Generic" message="The query may
be incomplete or requested object not found" messageCode="18522112101"
severity="warning">
               <Diagnostics>The requested action is not allowed on
server_4</Diagnostics>
           </Problem>
       </QueryStatus>
       <IscsiIsnsConfig address="192.168.50.55" esiPort="200" port="4280"
mover="1"/>
       <IscsiServiceState enabled="true" mover="1">
           <IsnsStatus maxSeverity="error">
               <Problem component="APL" message="cannot contact iSNSserver."
messageCode="13691256838" severity="error"/>
           </IsnsStatus>
       </IscsiServiceState>
   </ResponseEx>
</ResponsePacket>
```

iSCSI active management

Using the XML API, you can do the following:

- Create, modify, and delete iSCSI targets.
- Create, extend, and delete iSCSI LUNs.
- Create, modify, and delete iSCSI masks.
- Create, modify, and delete iSCSI shared secret.
- Modify iSCSI ISNS server configuration.
- Modify (enable/disable) iSCSI service state for a mover.

Creating an iSCSI target

Create an iSCSI target by starting a request to submit the task NewIscsiTarget.

The following example shows a request to create an iSCSI target from the target alias and mover:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
  <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
      <StartTask>
```

```
          <NewIscsiTarget mover="1" targetAlias="target1"  taskDescription="New
iSCSI Target">
          </NewIscsiTarget>
      </StartTask>
   </RequestEx>
</RequestPacket>
```

The following example shows a request to create an iSCSI target from the target alias, mover, and portals:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
      <StartTask>
         <NewIscsiTarget mover="1" targetAlias="target1"  taskDescription="New
iSCSI Target">
<Portals address="192.168.30.58" groupTag="100" port="5670">
</Portals>
        </NewIscsiTarget>
      </StartTask>
   </RequestEx>
</RequestPacket>
```

Modifying an iSCSI target

Modify an iSCSI target by starting a request to submit the task ModifyIscsiTarget.

The following example shows a request to modify an iSCSI target alias:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
       <StartTask>
<ModifyIscsiTarget mover="1" name="iqn.1992-05.com.emc:0002806001360000-3"
newTargetAlias="target1" taskDescription="Modify iSCSI Target">
</ModifyIscsiTarget>
        </StartTask>
   </RequestEx>
</RequestPacket>
```

The following example shows a request to modify an iSCSI target portal:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
       <StartTask>
<ModifyIscsiTarget mover="1" name="iqn.1992-05.com.emc:0002806001360000-3"
taskDescription="Modify iSCSI Target">
   <Portals address="192.168.50.58" groupTag="200" port="4320">
</Portals>
</ModifyIscsiTarget>
```

```
         </StartTask>
      </RequestEx>
</RequestPacket>
```

Deleting an iSCSI target

Delete an iSCSI target by starting a request to submit the task DeleteIscsiTarget.

The following example shows a request to delete an iSCSI target:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
        <StartTask>
<DeleteIscsiTarget mover="1" name="iqn.1992-05.com.emc:0002806001360000-3">
</DeleteIscsiTarget>
        </StartTask>
   </RequestEx>
</RequestPacket>
```

Creating an iSCSI LUN for a target

Create an iSCSI LUN by starting a request to submit the task NewIscsiLun.

The following example shows a request to create an iSCSI LUN by specifying the mover, target iSCSI name, LUN number, file system ID, and size:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
       <StartTask>
<NewIscsiLun target="iqn.1992-05.com.emc:0002806001360000-3" lun="20"
fileSystem="74" mover="1" size="25" taskDescription="New iSCSI Lun">
</NewIscsiLun>
       </StartTask>
   </RequestEx>
</RequestPacket>
```

Extending a LUN

Extend an iSCSI LUN by starting a request to submit the task ExtendIscsiLun.

The following example shows a request to extend an iSCSI LUN by specifying the reference to a LUN and an extension size:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
      <StartTask>
```

```
         <ExtendIscsiLun lun="99" mover="1" name="iqn.1992-
05.com.emc:0002806001360000-4" size="20" taskDescription="Extend Storage Pool">
         </ExtendIscsiLun>
      </StartTask>
   </RequestEx>
</RequestPacket>
```

Deleting a LUN associated with a target

Delete an iSCSI LUN by starting a request to submit the task DeleteIscsiLun.

The following example shows a request to delete an iSCSI LUN by specifying the reference to a LUN object:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
      <StartTask>
<DeleteIscsiLun lun="20" mover="1" name="iqn.1992-05.com.emc:0002806001360000-
3">
</DeleteIscsiLun>
      </StartTask>
  </RequestEx>
</RequestPacket>
```

Creating an iSCSI mask

Create an iSCSI mask by starting a request to submit the task NewIscsiMask.

The following example shows a request to create an iSCSI mask by specifying the mover, target iSCSI name, initiator, and list of LUNs:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
      <StartTask>
<NewIscsiMask target="iqn.1992-05.com.emc:0002806001360000-3"
initiator="iqn.1992-05.com.emc:0002806001360000-5" mover="1"
taskDescription="New iSCSI Mask">
     <GrantAccess start="23" size="5"></GrantAccess>
</NewIscsiMask>
        </StartTask>
    </RequestEx>
</RequestPacket>
```

Modifying an iSCSI mask

Modify an iSCSI mask by starting a request to submit the task ModifyIscsiMask.

The following example shows a request to modify an iSCSI mask by modifying the set of LUNs associated with the initiator by specifying the reference to the mask and set of LUNs:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
       <StartTask>
<ModifyIscsiMask initiator="iqn.1992-05.com.emc:0002806001360000-5" mover="1"
name="iqn.1992-05.com.emc:0002806001360000-3" taskDescription="Modify iSCSI
Mask">
    <GrantAccess start="26"></GrantAccess>
</ModifyIscsiMask>
         </StartTask>
     </RequestEx>
</RequestPacket>
```

Deleting an iSCSI mask

Delete an iSCSI mask by starting a request to submit the task DeleteIscsiMask.

The following example shows a request to delete an iSCSI mask by specifying the reference to the mask object:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
  <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
        <StartTask>
         <DeleteIscsiMask initiator="iqn.1992-05.com.emc:0002806001360000-5"
mover="1" name="iqn.1992-05.com.emc:0002806001360000-3" taskDescription="Delete
iSCSI Mask">
</DeleteIscsiMask>
        </StartTask>
   </RequestEx>
</RequestPacket>
```

Creating a shared secret or reverse authentication

Create an iSCSI shared secret or reverse authentication by starting a request to submit the task NewIscsiSharedSecret.

The following example shows a request to create an iSCSI shared secret by specifying the mover ID, initiator name, and password:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
    <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine
name or IP-->
        <StartTask>
         <NewIscsiSharedSecret mover="1" password="passwd123456789"
initiator="iqn.1992-05.com.emc:0002806001360000-3" taskDescription="Create
iSCSI Security type">
</NewIscsiSharedSecret>
         </StartTask>
    </RequestEx>
</RequestPacket>
```

The following example shows a request to create a iSCSI reverse authentication secret by specifying the mover ID and password:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
    <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine
name or IP-->
        <StartTask>
         <NewIscsiSharedSecret mover="1" password="passwd123456789"
taskDescription="Create iSCSI Security type">
</NewIscsiSharedSecret>
         </StartTask>
    </RequestEx>
</RequestPacket>
```

Modifying shared secret or reverse authentication

Modify an iSCSI shared secret or reverse authentication by starting a request to submit the task ModifyIscsiSharedSecret.

The following example shows a request to modify an iSCSI shared secret by specifying the reference to the shared secret object and the new password:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
       <StartTask>
<ModifyIscsiSharedSecret mover="1" password="1234567890123"
initiator="iqn.1992-05.com.emc:0002806001360000-3" taskDescription="Modify
iSCSI Security type">
</ModifyIscsiSharedSecret>
        </StartTask>
    </RequestEx>
</RequestPacket>
```

The following example shows a request to modify a iSCSI reverse authentication by specifying the mover ID and the new password:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
       <StartTask>
<ModifyIscsiSharedSecret mover="1" password="1234567890123"
taskDescription="Modify iSCSI Security type">
</ModifyIscsiSharedSecret>
        </StartTask>
    </RequestEx>
</RequestPacket>
```

Deleting a shared secret or reverse authentication

Delete an iSCSI shared secret or reverse authentication by starting a request to submit the task DeleteIscsiSharedSecret.

The following example shows a request to delete an iSCSI shared secret by specifying the reference to the shared secret object:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
       <StartTask>
<DeleteIscsiSharedSecret mover="1" initiator="iqn.1992-
05.com.emc:0002806001360000-3" taskDescription="Delete iSCSI Security type">
</DeleteIscsiSharedSecret>
        </StartTask>
     </RequestEx>
</RequestPacket>
```

The following example shows a request to delete an iSCSI reverse authentication by specifying the mover ID:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
       <StartTask>
<DeleteIscsiSharedSecret mover="1" taskDescription="Delete iSCSI Security
type">
</DeleteIscsiSharedSecret>
        </StartTask>
     </RequestEx>
</RequestPacket>
```

Modifying an ISNS configuration

Modify an iSCSI ISNS configuration by starting a request to submit the task ModifyIsnsConfig.

The following example shows a request to modify an iSCSI ISNS server configuration IP address and port by specifying the mover ID as the reference:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
       <StartTask>
<ModifyIsnsConfig mover="1" address="192.168.50.55" port="4280"
taskDescription="Modify iSCSI Config">
</ModifyIsnsConfig>
        </StartTask>
    </RequestEx>
</RequestPacket>
```

The following example shows a request to modify an iSCSI ISNS server configuration entry status inquiry port by specifying the mover ID as the reference:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
   <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine name
or IP-->
       <StartTask>
<ModifyIsnsConfig mover="1" esiPort="200"  taskDescription="Modify iSCSI
Config">
</ModifyIsnsConfig>
        </StartTask>
    </RequestEx>
</RequestPacket>
```

Modifying an iSCSI service state

Modify an iSCSI service state on the mover by starting a request to submit the task ModifyIscsiServiceState.

The following example shows a request to enable iSCSI service:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
    <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine
name or IP-->
        <StartTask>
<ModifyIscsiServiceState enabled="true" mover="1" taskDescription="Modify iSCSI
Service State">
</ModifyIscsiServiceState>
        </StartTask>
    </RequestEx>
</RequestPacket>
```

The following example shows a request to disable iSCSI service:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
    <RequestEx clientHandle="abc"><!--Set the clientHandle param to machine
name or IP-->
        <StartTask>
<ModifyIscsiServiceState enabled="false" mover="1" taskDescription="Modify
iSCSI Service State">
</ModifyIscsiServiceState>
        </StartTask>
    </RequestEx>
</RequestPacket>
```

Mount management

The schema file Mount.xsd defines data structures and operations related to file system mount configuration.

Mount data objects

A Celerra mount is identified by the mover ID and the mount path (This is a directory where the file system is mounted. In Celerra terminology it is called the mount point.) in the root file system of the mover or VDM. A special object MountRef represents a reference to a mount object. The package has only one data object: Mount.

Mount object

The object Mount describes the configuration of a mount. The following example shows a typical mount:

```
<Mount disabled="false" fileSystem="24" ntCredential="false" path="/fs24"
mover="2" moverIdIsVdm="false">
      <NfsOptions prefetch="true" ro="false" uncached="false"
virusScan="true"/>
      <CifsOptions accessPolicy="NATIVE" cifsSyncwrite="false"
lockingPolicy="nolock" notify="true" notifyOnAccess="false"
notifyOnWrite="false" oplock="true" triggerLevel="256"/>
</Mount>
```

Mount queries

The following section describes mount queries.

Retrieving mount objects

MountQueryParams object specifies parameters for retrieving mount objects.

Query filters include:

- A mover ID
- A mount path

The following examples show a request to query all mounts on a specific mover and the corresponding response:

Request:

```
<?xml version="1.0"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
     <Request>
         <Query>
             <MountQueryParams>
                 <MoverOrVdm mover="3" moverIdIsVdm="false"/>
             </MountQueryParams>
```

```
         </Query>
     </Request>
</RequestPacket>
```

Response:

Retrieving all mounts on a specific mover example

Mount active management

Using the XML API, you can create, modify, or delete a mount.

Creating a mount

Create a mount by starting a request to submit the task NewMount.

Note: When you create a file system by using the task NewFileSystem, the resulting file system is automatically mounted. The mount is created implicitly for this file system. This is not how it functions in CLI, where you must to create an unmounted file system and a mount point and then mount the file system on this mount point

The following example shows a request to create a mount:

```
<?xml version="1.0"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
     <StartTask>
        <NewMount path="/tree_fs0" fileSystem="23" ntCredential="true">
           <MoverOrVdm mover="1" moverIdIsVdm="false" />
           <NfsOptions prefetch="false" ro="false" uncached="true"
virusScan="false"/>
           <CifsOptions accessPolicy="NT" lockingPolicy="nolock"
cifsSyncwrite="true" notify="true"
            notifyOnAccess="true" notifyOnWrite="true" oplock="true"
triggerLevel="128"/>
        </NewMount>
     </StartTask>
  </Request>
</RequestPacket>
```

Modifying a mount

Modify a mount by starting a request to submit a task ModifyMount.

The following example shows a request to specify CIFS-specific mount options:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request clientHandle="abcdef">
    <StartTask>
      <ModifyMount mover="1" path="/tree_fs0" moverIdIsVdm="false" >
        <NfsOptions prefetch="false" ro="false" uncached="false"
virusScan="true"/>
        <CifsOptions accessPolicy="MIXED" lockingPolicy="wlock"
cifsSyncwrite="false" notify="false"
```

```
         notifyOnAccess="false" notifyOnWrite="false" oplock="false"
triggerLevel="1024"/>
      </ModifyMount>
    </StartTask>
  </Request>
</RequestPacket>
```

Deleting a mount

Delete a mount by starting a request to submit a task DeleteMount.

The following example shows a request to delete a mount:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <DeleteMount mover="1" path="/tree_fs0"/>
    </StartTask>
  </Request>
</RequestPacket>
```

Mover management

The schema file Mover.xsd defines data structures and operations related to Celerra Data Movers. A Mover and other related aspects in the XML API are logical entities. They reflect the logical, administrative, and status configuration of a Data Mover and the Celerra objects referenced in the context of that Data Mover. Since during failover the logical configuration of the failed Data Mover is assumed by the standby Data Mover, the properties that do not move with the failover are described by the MoverHost object in the Component.xsd package. A consequence of this, for example, is that Celerra network devices have two aspects: logical and physical. A logical device has a speed that is currently set for this device, and a physical device has a list of speeds that are allowed on this device by its physical capabilities. This is a departure from the conventional Celerra model.

Since a Celerra Virtual Data Mover (VDM), although a CIFS-related concept, is similar to a Data Mover, the VDM functionality is covered by the Mover package as well.

Mover data objects

A Data Mover (logical part) is identified by a numeric-string ID. A special object, MoverRef, represents a reference to a mover. In cases where the functionality applies either to a mover or a VDM, the reference is MoverOrVdmRef.

Depending on the configuration of a mover, it can have the following aspects:

Mover aspect

The Mover aspect describes the basic mover configuration.

The following example shows a fragment from the query response that demonstrates a typical Mover object:

```
<Mover failoverPolicy="manual" host="2" i18NMode="UNICODE" name="server_3"
nbsEnabled="true" ntpServers="172.24.145.170 172.24.152.170 172.24.153.170"
role="primary" standbyFors="" standbys="5" mover="2"/>
```

Note: The `host` property in the previous example has the same value as `id` property. This is normal since this mover has not failed over. Only in case of a failover would the values be different.

MoverStatus aspect

The MoverStatus aspect primarily describes properties that constantly change or are potentially unavailable (if the mover goes down). The application should not keep this object in the long-term cache.

The following example shows a fragment from the query response that demonstrates a typical MoverStatus object:

```
<MoverStatus clock="1133885742" csTime="1133903673" uptime="3257"
version="T5.5.15.1" mover="3">
          <Status maxSeverity="ok"/>
</MoverStatus>
```

MoverNisDomain object

The MoverNisDomain object covers mover NIS configuration, that is, the mover NIS domain name, the list of NIS lookup servers, and the domain name. For example:

```
<MoverNisDomain name="south" servers="172.24.145.170 172.24.152.170
172.24.153.170" mover="2"/>
```

MoverDnsDomain object

The MoverDnsDomain object describes mover DNS configuration for a given domain name. Its reference, MoverDnsDomain, contains two keys:

- Mover ID
- Domain name

For example:

```
<MoverDnsDomain protocol="udp" servers="172.24.146.176 172.24.146.175"
mover="3" name="sales_d2.sales_d.acme"/>
```

MoverDnsDomain is not an aspect of a mover; however, the entire set of objects is an aspect of a mover. In this version of the XML API, you cannot query a specific MoverDnsDomain object on a mover.

MoverInterface object

The MoverInterface object describes a mover network interface. Its reference, MoverInterfaceRef, contains two keys:

- Mover ID
- IP address of the interface

For example:

```
<MoverInterface broadcastAddr="172.24.142.255" device="trk71"
macAddr="0:60:16:5:4:b1" mtu="1500" name="gfsdvt11s301" netMask="255.255.255.0"
up="true" vlanid="0" ipAddress="172.24.142.10" mover="2"/
```

MoverInterface is not an aspect of a mover; however, the entire set of interfaces is an aspect of a mover. In this version of the XML API, you cannot query a specific MoverInterface object on a mover.

MoverRoute object

The MoverRoute object describes an entry in the mover routing table. Its reference, MoverRouteRef, contains two keys:

- Mover ID
- IP address of the destination

For example:

```
<MoverRoute gateway="192.168.2.5" interface="192.168.2.5"
netMask="255.255.255.0" destination="192.168.2.0" mover="4"/>
```

MoverRoute is not an aspect of a mover; however, the entire set of routing table entries is an aspect of a mover. In this version of the XML API, you cannot query a specific MoverRoute object on a mover.

LogicalNetworkDevice object

The LogicalNetworkDevice object describes a part of a network device that is configurable by means of software. Its reference, LogicalDeviceRef, contains two keys:

- Mover ID
- Celerra device name

For example:

```
<LogicalNetworkDevice interfaces="" speed="FD1000" type="physical-ethernet"
mover="1"
        name="fge1"/>
<LogicalNetworkDevice interfaces="172.24.139.2 172.24.139.85" speed="auto"
type="lacp"
        mover="2" name="lnk2">
          <VirtualDeviceData>
             <devices>
```

```
               <li>cge1</li>
               <li>cge3</li>
            </devices>
            <TrunkDeviceData/>
          </VirtualDeviceData>
</LogicalNetworkDevice>
```

LogicalNetworkDevice is not an aspect of a mover; however, the entire set of logical network devices is an aspect of a mover. In this version of the XML API, you cannot query a specific LogicalNetworkDevice object on a mover.

Vdm object

A Celerra VDM allows grouping CIFS servers and mover interfaces for the purpose of easy reconfiguration. It is represented in XML API by a Vdm object. For example:

```
<Vdm mover="2" name="vdm1" rootFileSystem="19" state="loaded" vdm="2">
       <Status maxSeverity="ok"/>
       <Interfaces>
          <li>cge0</li>
          <li>cge1</li>
       </Interfaces>
</Vdm
```

A VDM ID is in a different namespace than mover ID. VdmRef is a reference to a Vdm object; however, often when an operation applies to both movers and VDMs, a reference to MoverOrVdmRef is used.

Mover queries

Retrieving mover aspects

MoverQueryParams object specifies parameters for retrieving all aspects of movers, including sets of interfaces, routes, logical devices, and DNS domains.

Query filters include:

- An AspectSelection element specifying aspects needed by the application program
- A mover ID

The following examples show a request to query all information for a specific mover and the corresponding response:

Request:

```
<?xml version="1.0"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <Query>
            <MoverQueryParams mover="2" >
                <AspectSelection movers="true" moverDnsDomains="true"
moverInterfaces="true"
                    moverNetworkDevices="true"  moverNisDomains="true"
 moverRoutes="true"
```

```
                    moverStatuses="true" />
            </MoverQueryParams>
        </Query>
    </Request>
</RequestPacket>
```

Response:

Request to query all information for a specific mover example

The following examples show a request to query all DNS domains on all movers and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <Query>
            <MoverQueryParams>
                <AspectSelection moverDnsDomains="true"/>
            </MoverQueryParams>
        </Query>
    </Request>
</RequestPacket>
```

Response:

Request to query all DNS domains on all movers example

Retrieving Vdm objects

VdmQueryParams object specifies parameters for retrieving Vdm objects.

The query filter is a VDM ID.

The following examples show a request to query a specific VDMs in the system and the corresponding reply:

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <Query>
            <VdmQueryParams vdm="301"/>
        </Query>
    </Request>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
```

```
  <Response>
    <QueryStatus maxSeverity="ok"/>
    <Vdm mover="1" name="server_vdm2" rootFileSystem="4073" state="loaded"
vdm="301">
      <Status maxSeverity="ok"/>
      <Interfaces>
        <li>vdm_dvt11s2001</li>
        <li>vdm_dvt11s2002</li>
      </Interfaces>
    </Vdm>
  </Response>
</ResponsePacket>
```

Mover active management

Using the XML API, you can do the following:

- Modify movers
- Create, delete, or modify VDMs
- Modify, create, or delete a mover NIS domain
- Create or delete DNS domains
- Create or delete network interfaces
- Create or delete network routes
- Modify all logical network devices
- Create or delete virtual network devices

Modifying a mover

Modify mover properties by starting a request to submit the task ModifyMover.

The following example shows a request to modify the role of the mover:

```
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <ModifyMover mover="1">
        <Role>standby</Role>
      </ModifyMover>
    </StartTask>
  </Request>
</RequestPacket>
```

Creating a VDM

Create a VDM by starting a request to submit the task NewVdm. You can only create VDMs loaded on a mover. To create an unloaded VDM, create it on a mover and then use the ModifyVdm task to unload it.

The following example shows a request to create a VDM on a mover:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
```

```
      <NewVdm mover="1" name="vdmtest"/>
    </StartTask>
  </Request>
</RequestPacket>
```

Modifying a VDM

Currently, the Modify operation only allows you to change a VDM name (alias), moving it to another mover, or unload it. The request must submit the task ModifyVdm.

The following example shows a request to unload a VDM:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <ModifyVdm vdm="3">
        <HostMover/>
      </ModifyVdm>
    </StartTask>
  </Request>
</RequestPacket>
```

Deleting a VDM

Delete a VDM by supplying its ID within the task DeleteVdm.

The following example shows a request to delete a VDM:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request clientHandle='abcdef'>
    <StartTask>
      <DeleteVdm vdm="12"/>
    </StartTask>
  </Request>
</RequestPacket>
```

Modifying, creating, and deleting a mover NIS domain

Since there is only one NIS domain on a mover, the Modify operation is used to create, delete, or modify the domain properties. Modify a mover NIS domain by submitting the task ModifyMoverNisDomain. If the <Domain> element is not present within the task, this is effectively a request to delete NIS domain settings for this mover.

The following example shows a request to modify an NIS domain setting:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request clientHandle='abcdef'>
    <StartTask>
      <ModifyMoverNisDomain mover="2">
        <Domain name="dvt" servers="172.24.144.170" />
      </ModifyMoverNisDomain>
    </StartTask>
  </Request>
</RequestPacket>
```

Creating a new DNS domain

Create a new DNS domain on a mover by submitting the task NewMoverDnsDomain.

The following example shows a request to create a DNS domain:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request clientHandle='abcdef'>
    <StartTask>
      <NewMoverDnsDomain mover="1" name="w2k.eng.com" servers="172.24.198.65
172.24.198.65"
       protocol="udp" taskDescription="New dns domain w2k.eng.com"/>
    </StartTask>
  </Request>
</RequestPacket>
```

Deleting a DNS domain

Delete a DNS domain on a mover by submitting the task DeleteMoverDnsDomain.

The following example shows a request to delete a DNS domain:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request clientHandle='abcdef'>
    <StartTask>
      <DeleteMoverDnsDomain mover="2" name="w2k.eng.com"
      taskDescription="Delete dns domain w2k.eng.com"/>
    </StartTask>
  </Request>
</RequestPacket>
```

Creating an Interface

Create an interface by submitting the task NewMoverInterface.

The following example shows a request to create an interface:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request clientHandle='abcdef'>
    <StartTask>
      <NewMoverInterface mover="2" ipAddress="10.173.0.166" device="cge0"
mtu="1500" name="cge0"
       netMask="255.255.255.0" vlanid="0" taskDescription="Create iface"/>
    </StartTask>
  </Request>
</RequestPacket>
```

Deleting an Interface

Delete an interface by submitting the task DeleteMoverInterface.

The following example shows a request to delete an interface:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request clientHandle='abcdef'>
    <StartTask>
      <DeleteMoverInterface mover="2" ipAddress="10.173.0.166"/>
    </StartTask>
  </Request>
</RequestPacket>
```

Creating a Route entry

Create a route by submitting the task NewMoverRoute.

The following example shows a request to create a route:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <NewMoverRoute mover="1" gateway="172.24.8.182"
destination="172.24.173.31"
       netMask="255.255.255.0" taskDescription="create route to
172.24.173.31"/>
    </StartTask>
  </Request>
</RequestPacket>
```

Deleting a Route entry

Delete a route by submitting the task DeleteMoverRoute.

The following example shows a request to delete a route:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <DeleteMoverRoute mover="2" destination="172.24.173.31"/>
    </StartTask>
  </Request>
</RequestPacket>
```

Creating a Virtual Device

Depending on a type of the device:

- Create an Ethernet channel device by submitting the task NewEthernetChannelDevice.
- Create an lacp device by submitting the task NewLacpDevice.
- Create a fail-safe network device by submitting the task NewFsnDevice.

The following example shows a request that creates a new Ethernet channel device:

```
<?xml version="1.0"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask>
```

```
            <NewEthernetChannelDevice speed="HD1000" name="ethc2" mover="2">
                <Devices>
                   <li>ana1</li>
                   <li>ana3</li>
                </Devices>
            </NewEthernetChannelDevice>
        </StartTask>
    </Request>
</RequestPacket>
```

Modifying a Logical Device

Modify a device by submitting the task ModifyLogicalNetworkDevice. The only property you can modify is device speed.

The following example shows a request to modify the speed of the device:

```
<?xml version="1.0"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask>
            <ModifyLogicalNetworkDevice mover="2" name="ethc1" speed="FD100"/>
        </StartTask>
    </Request>
</RequestPacket>
```

Deleting a virtual device

Delete virtual device by submitting the task DeleteVirtualDevice.

The following example shows a request that deletes a virtual device:

```
<?xml version="1.0"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <DeleteVirtualDevice mover="2" name="lacp1"/>
    </StartTask>
  </Request>
</RequestPacket>
```

File system management

The schema file FileSystem.xsd defines data structures and operations related to Celerra file systems. A user-visible Celerra file system is one of the following types:

- Production file system
- Raw file system (no visible internal structure)
- Migration file system
- Checkpoint (a snapshot in time of a production file system)

File system data objects

A file system is identified by a numeric-string ID. A special object FileSystemRef represents a reference to a file system or a checkpoint. The XML API uses either file system IDs or references to refer to a particular file system, depending on the context.

Depending on the configuration of a file system, it can have the following aspects:

FileSystem aspect

The FileSystem aspect describes the basic file system configuration. Most of the properties of this object are relevant to any file system type (except checkpoint, which does not have this aspect). Type-specific properties are concentrated in special, non-primary objects that appear as elements within the schema definition of the FileSystem aspect:

- ProductionFileSystemData (data relevant to production file systems)
- MigrationFileSystemData (data relevant to migration file systems)

Raw file systems do not have type-specific data.

The following example shows a fragment from the query response that demonstrates two FileSystem aspects for two typical file system objects:

```
<FileSystem contains Slices="true" internalUse="false" name="mymgfs"
storagePools="1" storages="1" type="mgfs" volume="150" id="37">
         <MigrationFileSystemData state="idle"/>
</FileSystem>
<FileSystem containsSlices="true" internalUse="true"
name="root_fs_vdm_vdmtest7" storagePools="" storages="1" type="uxfs"
volume="108" id="18">
         <RwFileSystemHosts mover="2" moverIdIsVdm="false"/>
         <ProductionFileSystemData cwormState="off"/>
</FileSystem>
```

FileSystemCapacityInfo aspect

The FileSystemCapacityInfo aspect describes properties related to total and used capacity and the maximum and used files (inodes) count. For example:

```
<FileSystemCapacityInfo volumeSize="128" id="19">
           <ResourceUsage filesTotal="130942" filesUsed="16665"
spaceTotal="111" spaceUsed="73"/>
</FileSystemCapacityInfo>
```

For a file system that is not mounted, the entire structure <ResourceUsage/> is unknown and does not appear within the <FileSystemCapacityInfo/> element.

FileSystemCapabilities aspect

The FileSystemCapabilities aspect covers options for extending a file system. That is, if the file system is allocated from a storage pool, it contains the list of available and recommended storage pools for extending this file system. If the file system is allocated directly from storage volumes, it contains a list of available and recommended storage systems. For example:

```
<FileSystemCapabilities id="37"> <StoragePoolBased recommendedPool="1"
validPools="1 30"/>
</FileSystemCapabilities>
```

FileSystemCheckpointInfo aspect

If a file system has checkpoints, the FileSystemCheckpointInfo aspect describes the information related to file system checkpoints, such as the volume from which they are allocated and information related to this volume's space usage. For example:

```
<FileSystemCheckpointInfo id="25" savVolume="181" scalingFactor="1000"
spaceTotal="8123" spaceUsed="241" volumeSize="8631"/>
```

Checkpoint aspect

A file system checkpoint is a unique file system that has only one aspect: Checkpoint. Even though it is referenced by an ID from the same ID-space, it contains a set of properties untypical of a file system, such as the ID of the file system of which it is a checkpoint, or the UTC time of this checkpoint. For example:

```
<Checkpoint id="49" checkpointOf="25" fileSystemSize="352" name="ui_fs_ckpt2"
state="active" time="1133116307">
        <roFileSystemHosts mover="1" moverIdIsVdm="false"/>
</Checkpoint>
```

FileSystemAutoExInfo object

Automatic file system extension does not have a separate data object, but FileSystemAutoExtInfo, if available, is retrieved when querying for file systems with apiVersion="V1_1" or later.

The FileSystem object contains the element, FileSystemAutoExtInfo, if automatic file system extension information exists. For example:

```
<FileSystem containsSlices="true" internalUse="false" name="FSN9"
storagePools="15" storages="1" type="uxfs" volume="527" fileSystem="469">
       <RwFileSystemHosts mover="1" moverIdIsVdm="false"/>
      <FileSystemAutoExtInfo autoExtensionMaxSize="4" highWaterMark="90"
virtualProvisioning="false"/>
      <ProductionFileSystemData cwormState="off"/>
 </FileSystem>
```

NewFilesystem object

The NewFileSystem object is modified to provide automatic file system extension information when creating file systems. For example:

```
<NewFileSystem name="">
       <Mover mover="1">
       </Mover>
       <StoragePool pool="1">
              <EnableAutoExt autoExtensionMaxSize="4" highWaterMark="85"
virtualProvisioning="true">
              </EnableAutoExt>
```

```
       </StoragePool>
</NewFileSystem>
```

ModifyFilesystem object

The ModifyFileSystem object is modified to provide automatic file system extension information when changing file systems. For example:

```
<ModifyFileSystem fileSystem="477" >
<EnableAutoExt highWaterMark="72" autoExtensionMaxSize="87"
virtualProvisioning="false"/>
</ModifyFileSystem>
```

File system queries

Note: On a system with a large number of file systems or checkpoints, a query can take considerable time. Therefore, the client application may need to keep a cache of the aspects it needs.

Retrieving file system aspects (except checkpoints)

FileSystemQueryParams object specifies parameters for retrieving all aspects of file systems except Checkpoint objects.

Query filters include:

- An AspectSelection element specifying aspects needed by the application program
- A file system ID
- A file system name
- A mover ID for the mover on which the file system is mounted
- A VDM ID for the VDM on which the file system is mounted

The following examples show a request to query all information for multiple file systems and the corresponding response:

Request:

```
 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <Query>
            <FileSystemQueryParams>
                <AspectSelection fileSystems="true"
fileSystemCapacityInfos="true"
                    fileSystemCheckpointInfos="true"
fileSystemCapabilities="true" />
            </FileSystemQueryParams>
        </Query>
    </Request>
</RequestPacket>
```

Response:

Retrieving_file_system_aspects_example

The following examples show a request for file system checkpoint information for a specific file system and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <Query>
            <FileSystemQueryParams>
                <AspectSelection  fileSystemCheckpointInfos="true" />
                <FileSystem fileSystem="30"/>
            </FileSystemQueryParams>
        </Query>
    </Request>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Response>
        <QueryStatus maxSeverity="ok"/>
        <FileSystemCheckpointInfo savVolume="336" scalingFactor="1000"
spaceTotal="18823" spaceUsed="10601" volumeSize="20000" fileSystem="30"/>
    </Response>
</ResponsePacket>
```

Retrieving Checkpoint objects

CheckpointQueryParams object specifies parameters for retrieving Checkpoint objects.

Query filters include:

- A checkpoint ID
- A checkpoint name
- A mover ID for the mover on which the checkpoint is mounted
- A VDM ID for the VDM on which the checkpoint is mounted

The following examples show a request to query all checkpoints in the system and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
 <Request>
     <Query>
         <CheckpointQueryParams/>
     </Query>
```

```
 </Request>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Response>
        <QueryStatus maxSeverity="ok"/>
        <Checkpoint checkpointOf="30" fileSystemSize="235"
name="demo_fs1_ckpt1" state="active" time="1136744988" checkpoint="33"/>
        <Checkpoint checkpointOf="30" fileSystemSize="238"
name="demo_fs1_ckpt2" state="active" time="1137526425" checkpoint="34">
            <roFileSystemHosts mover="1" moverIdIsVdm="false"/>
        </Checkpoint>
    </Response>
</ResponsePacket
```

Retrieving automatic file system extension information

The FileSystemAutoExtInfo element is a part of FileSystem object and is available when users query for file system objects with apiVersion="V1_1" or later. The query filter is an aspect selection element specifying the file system's aspect for apiVersion="V1_1" or later.

The following examples show a request to query for automatic file system extension information and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
<Request>
      <Query>
           <FileSystemQueryParams>
    <AspectSelection fileSystems="true"></AspectSelection>
           </FileSystemQueryParams>
     </Query>
</Request>
</RequestPacket>
```

Response:

Retrieve automatic file system extension information example

File system active management

Using the XML API, you can do the following:

- Create, extend, delete, or modify file systems
- Create or delete checkpoints
- Perform a refresh of a checkpoint
- Restore a file system from a checkpoint
- Create, modify, enable, and disable automatic file system extension

Creating a file system

The XML API can create a production file system only. Create a file system by starting a request to submit the task NewFileSystem.

Note: The application cannot create an unmounted file system.

The following example shows a request to create a file system from a storage pool:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask timeout="30">
      <NewFileSystem name="FS_TEST">
        <Mover mover="2" />
        <StoragePool pool="1" storage="1" size="10" mayContainSlices="true"/>
        <Mount path="/FS_TEST_MOUNT" />
      </NewFileSystem>
    </StartTask>
  </Request>
</RequestPacket>
```

Extending a file system

Extend a file system by starting a request to submit the task ExtendFileSystem.

Note: For the request to succeed, the system must be mounted.

The following example shows a request to extend a file system by specifying the storage from which additional space is allocated:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request clientHandle='abcdef'>
    <StartTask>
      <ExtendFileSystem fileSystem="59">
        <StoragePool pool="3" size="1000" storage="1"/>
      </ExtendFileSystem>
    </StartTask>
  </Request>
</RequestPacket>
```

Modifying a file system

Currently, the Modify operation changes only the file system name (alias). The request must submit the task ModifyFileSystem.

The following example shows a request to modify a file system:

```
<?xml version="1.0"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request clientHandle='zyz124'>
    <StartTask>
      <ModifyFileSystem fileSystem="30" newName="abcdefg"/>
    </StartTask>
```

```
  </Request>
</RequestPacket>
```

Deleting a file system

Delete a file system by supplying the file system ID within the task DeleteFileSystem.

The following example shows a request to delete a file system:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <DeleteFileSystem fileSystem="96"
       taskDescription="Delete file system &quot;SALES&quot;" />
    </StartTask>
  </Request>
</RequestPacket>
```

Creating a Checkpoint

Create a checkpoint of a file system by submitting the task NewCheckpoint. If the file system does not have at least one previously created checkpoint, the NewCheckpoint object needs to specify the space allocation method for the savVol (pool volume on which all checkpoints reside).

The following examples show requests to create a checkpoint for a storage pool, a storage system, and a volume.

Storage pool allocation:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request clientHandle='abcdef'>
    <StartTask>
      <NewCheckpoint checkpointOf="58" name="test_fs1_ckpt0">
        <SpaceAllocationMethod>
          <StoragePool pool="1" size="64"/>
        </SpaceAllocationMethod>
      </NewCheckpoint>
    </StartTask>
  </Request>
</RequestPacket>
```

Storage system allocation:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request clientHandle='abcdef1'>
    <StartTask>
      <NewCheckpoint checkpointOf="59" name="test_fs4_ckpt0">
        <SpaceAllocationMethod>
          <StorageSystem storage="1"/>
        </SpaceAllocationMethod>
      </NewCheckpoint>
    </StartTask>
  </Request>
</RequestPacket>
```

Volume allocation:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <NewCheckpoint checkpointOf="61" name="test_fs5_ckpt0">
        <SpaceAllocationMethod>
          <Volume volume="198"/>
        </SpaceAllocationMethod>
      </NewCheckpoint>
    </StartTask>
  </Request>
</RequestPacket>
```

Refreshing a Checkpoint

Refresh an existing checkpoint with up-to-date file system content by submitting the task RefreshCheckpoint. A refresh requires the checkpoint ID.

The following example shows a request to refresh a checkpoint:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <RefreshCheckpoint checkpoint="68"/>
    </StartTask>
  </Request>
</RequestPacket>
```

Restoring a file system from a Checkpoint

Restore a file system from a checkpoint by submitting the task RestoreCheckpoint.

Note: Optionally, you can specify the name of a checkpoint that receives the overwritten content of the restored file system.

The following example shows a request to restore a checkpoint:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <RestoreCheckpoint checkpoint="70" oldContentName="ckpt2"/>
    </StartTask>
  </Request>
</RequestPacket>
```

Deleting a Checkpoint

Delete a checkpoint by submitting the task DeleteCheckpoint.

The following example shows a request to delete a checkpoint:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <DeleteCheckpoint checkpoint="64"/>
    </StartTask>
  </Request>
</RequestPacket>
```

Creating a file system with automatic file system extension enabled

Create a file system by starting a request to submit the task NewFileSystem and allocate space from the storage pool.

The following example shows a request to create by specifying the mover, storage pool, and automatic file system extension setting:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
   <Request>
       <StartTask>
           <NewFileSystem name="">
               <Mover mover="1"></Mover>
               <StoragePool pool="1">
                   <EnableAutoExt autoExtensionMaxSize="4" highWaterMark="85"
virtualProvisioning="true"></EnableAutoExt>
                </StoragePool>
           </NewFileSystem>
       </StartTask>
   </Request>
</RequestPacket>
```

Modifying a file system to enable automatic file system extension

Modify the file system to enable automatic file system extension by starting a request to submit the ModifyFileSystem task and specifying the file system ID and automatic file system extension setting.

Automatic file system extension can be modified only for file systems created on a storage pool.

The following example shows a request to enable automatic file system extension and to modify automatic file system extension settings using the file system ID:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
   <Request>
       <StartTask>
           <ModifyFileSystem fileSystem="1">
               <AutoExtend enableAutoExtension="true" autoExtensionMaxSize="5"
highWaterMark="95" virtualProvisioning="true"></AutoExtend>
           </ModifyFileSystem>
       </StartTask>
   </Request>
```

```
</RequestPacket>
```

Disabling automatic file system extension

The following example shows a request to disable automatic file system extension:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
   <Request>
       <StartTask>
           <ModifyFileSystem fileSystem="1">
               <AutoExtend enableAutoExtension="false"></AutoExtend>
           </ModifyFileSystem>
       </StartTask>
   </Request>
</RequestPacket>
```

Storage pool management

The schema file StoragePool.xsd defines data structures and operations related to Celerra storage pools.

Storage pool data objects

A Celerra storage pool is a group of volumes from which storage can be allocated by AVM. There are two types of pools: system- and user-defined. System-defined pools can automatically manage their own expansion and contraction. User-defined pools are populated by the system administrator, who explicitly adds and removes volumes. A storage pool is identified by the storage pool ID. A special object, StoragePoolRef, represents a reference to a storage object. Most of the properties of storage pools are shared between user- and system-defined storage pools; however, a system storage pool has some additional data associated with it.

StoragePool object

The object StoragePool describes the configuration of a storage pool. Here are two examples, one for a system storage pool and another for a user storage pool:

```
<StoragePool autoSize="172620" description="Symmetrix STD" diskType="std"
mayContainSlicesDefault="true" memberVolumes="111 145 148 151 154" movers="1 2"
name="symm_std" size="172620" storageSystems="1" usedSize="10933" id="1">
        <SystemStoragePoolData dynamic="true" greedy="true"
potentialAdditionalSize="0"/>
</StoragePool>
<StoragePool autoSize="17262" description="" diskType="std"
mayContainSlicesDefault="true" memberVolumes="30 31" movers="1 2"
name="my_pool" size="17262" storageSystems="1" usedSize="0" id="33">
        <UserStoragePoolData/>
 </StoragePool>
```

Note: The element <UserStoragePoolData> does not have content or attributes. This is only a tagging element.

Storage pool queries

The following section describes storage pool queries.

Retrieving StoragePool objects

StoragePoolQueryParams object specifies parameters for retrieving StoragePool objects. Query filter is storage pool ID.

The following examples show a request to query all storage pools and the corresponding response:

Request:

```
<?xml version="1.0"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
 <Request>
     <Query>
         <StoragePoolQueryParams/>
     </Query>
   </Request>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Response>
    <QueryStatus maxSeverity="ok"/>
    <StoragePool autoSize="181251" description="Symmetrix STD" diskType="std"
mayContainSlicesDefault="false" memberVolumes="110       130" movers="1 2"
name="symm_std" size="138096" storageSystems="1" usedSize="89100" pool="1">
      <SystemStoragePoolData dynamic="true" greedy="false"
potentialAdditionalSize="43155"/>
    </StoragePool>
    <StoragePool autoSize="34524" description="User symmetrix pool"
diskType="std" mayContainSlicesDefault="true" memberVolumes="23       24 25 26"
movers="1 2" name="u_pool1" size="34524" storageSystems="1" usedSize="0"
pool="12">
      <UserStoragePoolData/>
     </StoragePool>
  </Response>
</ResponsePacket>
```

Storage active management

Using the XML API you can create or delete user storage pools, modify parameters of system storage pools, and extend or shrink storage pools.

Creating a user storage pool

Create a user storage pool by starting a request to submit the task NewUserStoragePool.

The following example shows a request to create a user storage pool:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask>
            <NewUserStoragePool memberVolumes="53 54 55 56 57 58"
name="u_pool1" description="" mayContainSlicesDefault="true"/>
        </StartTask>
    </Request>
</RequestPacket>
```

Modifying a system storage pool

Modify a system storage pool by starting a request to submit the task ModifySystemStoragePool or ModifyStoragePool. The former task modifies some properties of system storage pools only, while the latter modifies properties of all pools.

The following example shows a request to modify a storage pool:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask>
            <ModifyStoragePool pool="13" newName="new-user-pool-name"
mayContainSlicesDefault="true"/>
        </StartTask>
    </Request>
</RequestPacket>
```

Deleting a storage pool

Delete a storage pool by starting a request to submit the task DeleteStoragePool. Only user storage pools can be deleted.

The following example shows a request to delete a storage pool:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask>
            <DeleteStoragePool pool="15" recursive="true"  />
        </StartTask>
    </Request>
</RequestPacket>
```

Extending a storage pool

Extend a storage pool by starting a request to submit the task ExtendStoragePool.

The following example shows a request to extend a storage pool:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask>
            <ExtendStoragePool pool="13" memberVolumes="60 61 62 63 64"  />
        </StartTask>
    </Request>
</RequestPacket>
```

Shrinking a storage pool

Shrink a storage pool by starting a request to submit the task ShrinkStoragePool.

The following example shows a request to shrink a storage pool:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask>
            <ShrinkStoragePool pool="13" memberVolumes="53"  />
        </StartTask>
    </Request>
</RequestPacket>
```

Quota management

The schema file Quota.xsd defines data structures and operations related to Celerra quotas on file systems and their subdirectories.

Quota data objects

The following section provides a description of the XML API quota model.

Any quota is associated with a tree (a directory and its subdirectories to which NAS quotas apply) within a file system and is identified by the file system ID and the directory path. However, since a tree can be moved within a file system, each directory path has a numeric identifier called tree ID. This ID is immutable. The tree ID of the top-level directory within the file system is always "0". Other IDs are created by the system when the administrator creates a directory affected by quotas. There can be no more than 2,047 tree IDs per file system.

Paths are mutable, but applications can refer to any quota object within the XML model specifying the file system ID and the directory path. The ID of the tree, however, is always returned in queries, so that the application can internally organize its databases or caches by using the immutable tree ID as a key instead of the directory path.

Note: Only administrators can create trees. They can do so locally by means of the CLI or, remotely, by means of the XML API or Celerra Manager.

Tree quotas

In Celerra you can impose resource usage limits on a tree. These limits are called tree quotas. The object TreeRef represents a reference to a tree within a file system. You cannot create tree quotas on the top-level directory (ID = 0). The objects that reflect the quota limits and the usage of trees in the XML API are respectively TreeQuota and TreeQuotaUsage.

User and Group quotas

In addition to limits that can be imposed on trees, individual users or user groups can be limited in the usage of trees in two ways:

- By specifying limitations on resources owned by any user or user group – default user or group limits (see TreeSettings object)
- By specifying limitations on resources owned by a specific user or user group that override default user or group limits (see UserQuota object in which the default limits are set)

Specific user or group limits can be imposed at any level, including the top-level directory for the entire file system. User and group limits are described in the XML model by the UserQuota object, and they are referred by UserQuotaRef, which inherits them from the TreeRef object, but also specifies either the user or a group ID. Similar to tree quotas, the resource usage by this user or group is described in the model by UserQuotaUsage object.

Controlling events generation and grace periods

User and group quotas share a set of configuration and default parameters. These parameters configure system behavior when various limits are reached. It also sets default user and group limits. These parameters are described in the XML model by the object TreeSettings. This object inherits from TreeRef object.

A special note: The TreeSettings object for the top-level directory (tree ID 0) controls not only configuration and default parameters for users/groups, but also configuration and default parameters for all TreeQuota objects in this file system. For example, if the user crosses soft limits of both tree quota and user quota, the grace periods for this tree are set using the values set in the TreeSettings object that has tree ID 0, but the grace period for this particular user is set based on the TreeSettings for this particular tree.

Note: In the XML API, TreeQuota, TreeQuotaUsage, and TreeSettings objects are cached by the Control Station server and retrieved from the Data Mover once a day.

Summarizing, a tree has three aspects:

- TreeQuota – configuration of tree quotas
- TreeQuotaUsage – information about tree quota usage
- TreeSettings – configuration settings and defaults for users/groups for this tree; also see a special note above.

A specific user or group working within a tree can have two aspects:

- UserQuota– user or group quota configuration
- UserQuotaUsage – user or group quota usage

TreeQuota aspect

The TreeQuota aspect of tree quota describes tree quota limits. For example:

```
<TreeQuota comment="Admin J. Smith, x 6831" treeId="1" fileSystem="20"
path="/sales">
           <Limits filesHardLimit="3000" filesSoftLimit="2000"
spaceHardLimit="20000"
             spaceSoftLimit="10000"/>
</TreeQuota>
```

TreeQuotaUsage aspect

The TreeQuotaUsage aspect of tree quota describes the current usage of the resource limited by a tree quota. For example:

```
<TreeQuotaUsage filesUsed="159" spaceUsed="2759" treeId="5" fileSystem="22"
path="/athlantis">
            <TimeRemains space="78226" >
            </TimeRemains>
</TreeQuotaUsage>
```

TreeSettings aspect

The TreeSettings aspect describes configuration settings that apply to a tree for user or group quotas. It contains flags and parameters that control the behavior of the system when any of the specified limits are crossed. It also contains default settings for user and group limits.

The following example shows a fragment from the query response that demonstrates the TreeSettings aspect for a typical configuration:

```
<TreeSettings treeId="5" fileSystem="20" path="/acme/marketing">
        <QuotaOptions checkEndEvent=" true" checkStartEvent="true"
crossedSoftEvent="true"
            enableGroupQuotas="true" enableUserQuotas=" true"
exceededHardEvent="true"
            hardLimitEnforced="true"/>
        <QuotaGracePeriod files="1209600" space="1209600"/>
        <UserLimits filesHardLimit="40000" filesSoftLimit="50000"
spaceHardLimit="100000"
            spaceSoftLimit="80000"/>
        <GroupLimits filesHardLimit="40000" filesSoftLimit="50000"
spaceHardLimit="120000"
            spaceSoftLimit="100000"/>
</TreeSettings>
```

UserQuota aspect

The UserQuota aspect describes limits that are imposed on a specific user or user group. For example:

```
<UserQuota treeId="3" winSecurityIds="" isGroup="false" uid="202"
        fileSystem="22" path="/testpath">
            <Limits filesHardLimit="4000" filesSoftLimit="3000"
               spaceHardLimit="30000" spaceSoftLimit="20000"/>
</UserQuota>
```

UserQuotaUsage aspect

The UserQuotaUsage aspect of tree quota describes the current usage of the resource limited by a current user or group quotas. For example (in this example the director has not yet been used):

```
<UserQuotaUsage filesUsed="0" spaceUsed="0" treeId="0" isGroup="false"
              uid="202" fileSystem="20" path="/">
          <TimeRemains/>
</UserQuotaUsage>
```

Quota queries

The following section describes quota queries.

Retrieving tree aspects

TreeQuotaQueryParams object specifies parameters for retrieving all aspects of a tree.

Query filters include:

- An AspectSelection element specifying aspects needed by the application program
- A file system ID
- A directory path

The following examples show a request to query all quota information for a specific file system and the corresponding response:

Request:

```
<?xml version="1.0"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
 <Request>
     <Query>
         <TreeQuotaQueryParams fileSystem="30">
             <AspectSelection treeSettings="true" treeQuotas="true"
treeQuotaUsages="true" />
         </TreeQuotaQueryParams>
     </Query>
 </Request>
</RequestPacket>
```

Response:

Retrieving all quota information for a specific file system example

Retrieving user and group aspects

UserQuotaQueryParams object specifies parameters for retrieving aspects of user quotas:

Query filters are:

- An AspectSelection element specifying aspects needed by the application program

- A file system ID
- A directory path

The following examples show a request to query all aspects for a specific user on a specific file system and directory and the corresponding response.

Request:

```
<?xml version="1.0"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <Query>
            <UserQuotaQueryParams fileSystem="30" path="/tree0" uid="201"
isGroup="false">
                <AspectSelection userQuotas="true" userQuotaUsages="true" />
            </UserQuotaQueryParams>
        </Query>
    </Request>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Response>
        <QueryStatus maxSeverity="ok"/>
        <UserQuota tree="1" winSecurityIds="" isGroup="false" uid="201"
fileSystem="30" path="/tree0">
            <Limits filesHardLimit="3000" filesSoftLimit="2000"
spaceHardLimit="30" spaceSoftLimit="20"/>
        </UserQuota>
        <UserQuotaUsage filesUsed="1245" spaceUsed="12" tree="1"
isGroup="false" uid="201" fileSystem="30" path="/tree0">
            <TimeRemains/>
        </UserQuotaUsage>
    </Response>
</ResponsePacket>
```

Quota active management

Using the XML API, you can do the following:

- Create a tree and, optionally, set a tree quota on it
- Modify and delete (turn off) quotas on a tree
- Create and modify user quotas. You cannot delete user quotas, but all user quotas on the tree are removed when the tree is deleted (turned off)

Creating a tree

Create a tree by starting a request to submit the task NewTree. This operation implicitly creates TreeQuota and TreeSettings objects for this tree. When creating a tree, optionally, you can specify TreeQuota object limits. If you do not specify limits, all limits are set to 0. The TreeSettings object, created with the task NewTree, will have all flags set to false, all limits set to 0, and all grace period values set to 1 week.

The following example shows a request to create a tree:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask>
            <NewTree fileSystem="30" path="/eng_div"/>
        </StartTask>
    </Request>
</RequestPacket>
```

Modifying a tree quota

Modify tree quota limits by starting a request to submit the task ModifyTreeQuota. If limits are not specified, no changes to the limits are made.

The following example shows a request to modify tree quota limits:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask>
            <ModifyTreeQuota fileSystem="24" path="/arie1">
                <Limits filesSoftLimit="10000" filesHardLimit="20000"
                spaceSoftLimit="10000" spaceHardLimit="20000"/>
            </ModifyTreeQuota>
        </StartTask>
    </Request>
</RequestPacket>
```

Modifying a tree settings

Modify tree settings by starting a request to submit the task ModifyTreeSettings. If limits are not specified, no changes to the limits are made.

The following example shows a request to modify tree settings:

```
<?xml version="1.0"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask>
            <ModifyTreeSettings fileSystem="82" path="/tree1" >
                <QuotaOptions crossedSoftEvent="true" enableGroupQuotas="true"
                    enableUserQuotas="true" exceededHardEvent="true"
                    hardLimitEnforced="true"/>
                <QuotaGracePeriod files="1209600" space="1209600"/>
                <UserLimits filesSoftLimit="5000" filesHardLimit="10000"
                    spaceSoftLimit="5000" spaceHardLimit="8000"/>
                <GroupLimits filesSoftLimit="10000" filesHardLimit="12000"
                    spaceSoftLimit="10000" spaceHardLimit="13000"/>
            </ModifyTreeSettings>
        </StartTask>
    </Request>
</RequestPacket>
```

Deleting a tree

Delete a tree by starting a request to submit the task DeleteTree.

The following example shows a request to delete a tree:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask>
            <DeleteTree fileSystem="30" path="/tree5"/>
        </StartTask>
    </Request>
</RequestPacket>
```

Creating user quotas

Create one or more user quotas by starting a request to submit the task NewUserQuota. This operation creates user quotas for a list of users specified by user IDs. To create a user quota, the tree must exist.

The following example shows a request to create a user quota:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask>
            <NewUserQuota fileSystem="25">
                <UserOrGroupList Ids="202 203" isGroup="false"/>
                <Limits filesSoftLimit="1000" filesHardLimit="2000" s
                paceSoftLimit="1000" spaceHardLimit="2000"/>
            </NewUserQuota>
        </StartTask>
    </Request>
</RequestPacket>
```

Modifying a user quota

Modify user quota starting a request to submit the task ModifyUserQuota.

The following example shows a request to modify a user quota:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask>
            <ModifyUserQuota uid="203" fileSystem="24" path="/zyx"
isGroup="false" >
                <Limits filesSoftLimit="1000" filesHardLimit="2000"
                spaceSoftLimit="1000" spaceHardLimit="2000"/>
            </ModifyUserQuota>
        </StartTask>
    </Request>
</RequestPacket>
```

NFS management

The schema file Nfs.xsd defines data structures and operations related to exporting Celerra file systems using NFS protocol.

NFS data objects

The NFS protocol is available on movers only. It is not available on VDMs. An NFS export is identified by the mover on which the file system is exported and the mount path. A special object NfsExportRef represents a reference to an NFS export object. The package has only one data object: NfsExport.

NfsExport object

This object describes the configuration of an NFS export. The following example shows an export that does not restrict access client hosts:

```
<NfsExport anonUser="0" readOnly="false" mover="3" path="/server4fs4"/>
```

NFS queries

The following section describes NFS queries.

Retrieving NfsExport objects

NfsExportQueryParams object specifies parameters for retrieving NfsExport objects.

Query filters include:

- A mover ID
- A mount path

The following examples show a request to query an export with a specified path on any mover and the corresponding response:

Request:

```
<?xml version="1.0"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
     <Request>
         <Query>
             <NfsExportQueryParams path="/server3fs5"/>
         </Query>
     </Request>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
```

```
  <Response>
    <QueryStatus maxSeverity="ok"/>
    <NfsExport anonUser="0" readOnly="false" mover="2" path="/server3fs5"/>
  </Response>
</ResponsePacket>
```

NFS active management

Using the XML API, you can create, modify, or delete an NFS export.

Creating a NFS export

Create an NFS export by starting a request to submit the task NewNfsExport.

The following example shows a request to create an NFS export:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask timeout="600" >
            <NewNfsExport mover="1" path="/test_fs1" anonUser="0"
readOnly="false" taskDescription="Delete Nfs Export" >
                <RootHosts>
                    <li>172.24.198.78</li>
                </RootHosts>
            </NewNfsExport>
        </StartTask>
    </Request>
</RequestPacket>
```

Modifying an NFS export

Modify an NFS export by starting a request to submit the task ModifyNfsExport.

The following example shows a request to limit the export to read-only mode:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request clientHandle="PASSED">
        <StartTask>
            <ModifyNfsExport mover="2" path="/fs24" readOnly="false">
                <RwHosts>
                    <li>172.24.198.78</li>
                    <li>172.24.198.66</li>
                </RwHosts>
            </ModifyNfsExport>
        </StartTask>
    </Request>
</RequestPacket>
```

Deleting an NFS Export

Delete an NFS export by starting a request to submit the task DeleteNfsExport.

The following example shows a request to delete an export:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask>
            <DeleteNfsExport mover="1" path="/test_fs1"/>
        </StartTask>
    </Request>
</RequestPacket>
```

CIFS management

The schema file Cifs.xsd defines data structures and operations related to Celerra CIFS service. It allows managing mover or VDM CIFS configurations, CIFS servers, and CIFS shares.

CIFS data objects

The CIFS service is managed by the Data Mover or VDM. Each mover or VDM has its own CIFS configuration. Therefore, CIFS manageable data needs to be manageable at the level of a mover or a VDM, and all references to CIFS objects must be inherited from the MoverOrVdmRef object.

A CIFS server is identified by the ID of the mover or VDM on which it resides and the server NETBIOS name. It is referenced by the object CifsServerRef that incorporates both fields.

A CIFS share is identified by:

- The ID of the mover or VDM on which it is exported.
- The list of CIFS servers on this mover or VDM that provide access to this share.
- The share name.

A special object, CifsShareRef, represents a reference to a share. It incorporates all three fields.

CifsConfig aspect

This aspect describes the basic CIFS configuration on the specified mover or VDM. Therefore, it is an aspect of a mover or a VDM. The following example shows a fragment from the query response that demonstrates a CifsConfig aspect:

```
<CifsConfig cifsEnabled="true" userMapperServers="172.24.173.201
172.24.173.205" winServers="172.24.16.11 172.24.16.45" mover="1"
moverIdIsVdm="false"/>
<CifsConfig userMapperServers="172.24.173.201 172.24.173.205"
winServers="172.24.16.11 172.24.16.45" mover="2" moverIdIsVdm="true"/>
```

CisfServer aspect

A CifsServer is the only aspect of a Celerra CIFS server. There can be three types of a CIFS server:

- A stand-alone server that does not require any Windows infrastructure and has user accounts local to the mover
- An NT4.0 server that emulates a NT4.0 system
- A W2K server that emulates a Windows 2000 system

Type-specific properties are concentrated in special, non-primary objects that appear as elements within the schema definition of the CifsServer aspect:

- StandaloneServerData (data relevant to stand-alone servers)
- NT40ServerData (data relevant to NT4.0 servers)
- W2KServerData (data relevant to W2K servers)

The following example shows a fragment from the query response, which demonstrates two CifsServer aspects for two typical server objects:

```
<CifsServer interfaces="172.24.113.1" localUsers="false" type="NT4"
name="H20SAL11S2" mover="1" moverIdIsVdm="false">
           <Aliases>
               <li>H20SAL11S2-1</li>
               <li>H20SAL11S2-2</li>
           </Aliases>
           <NT40ServerData domain="<MA_D2" domainJoined="true"/>
</CifsServer>
<CifsServer interfaces="172.24.143.3" localUsers="false" type="W2K"
name="GGSSAL11S302"
mover="2" moverIdIsVdm="false">
           <Aliases>
               <li>GGSSAL11S302-1</li>
               <li>GGSSAL11S302-2</li>
           </Aliases>
           <W2KServerData compName="ggssal11s302" domain="ACME.SALES.MI_D1"
domainJoined="true"/>
</CifsServer>
```

CifsShare aspect

A CIFS share (a directory or a file system exported by means of the CIFS protocol) has only one aspect: CifsShare. The following example demonstrates a typical CifsShare aspect:

```
<CifsShare fileSystem="223" path="\server2fs1" name="svr2sh1" mover="1"
moverIdIsVdm="false">
           <CifsServers>
                <li>H20SAL11S2</li>
           </CifsServers>
</CifsShare>
```

CIFS queries

The following section describes the functions of CIFS parameters.

Retrieving CifsConfig objects

CifsConfigQueryParams specifies parameters for retrieving CifsConfig objects.

Query filters include:

- A mover
- A VDM ID

The following examples show a request to retrieve a CifsConfig aspect for a specified mover and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <Query>
            <CifsConfigQueryParams>
                <MoverOrVdm mover="4" moverIdIsVdm="false" />
            </CifsConfigQueryParams>
        </Query>
    </Request>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Response>
        <QueryStatus maxSeverity="ok"/>
        <CifsConfig cifsEnabled="true" userMapperServers="192.168.1.2
192.168.2.2" winServers="" mover="4" moverIdIsVdm="false"/>
    </Response>
</ResponsePacket>
```

Retrieving CifsServer objects

CifsServerQueryParams specifies parameters for retrieving CifsServer objects.

Query filters include:

- A mover or VDM ID
- A CIFS server NETBIOS name

The following examples show a request to retrieve a specific CifsServer object and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
     <Request>
         <Query>
             <CifsServerQueryParams name="GG3DVT11S603">
                 <MoverOrVdm mover="5"/>
             </CifsServerQueryParams>
         </Query>
```

```
     </Request>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Response>
        <QueryStatus maxSeverity="ok"/>
        <CifsServer interfaces="172.24.143.24" localUsers="false" type="W2K"
name="GG3DVT11S603" mover="5" moverIdIsVdm="false">
            <Aliases>
                <li>GG3DVT11S603-1</li>
                <li>GG3DVT11S603-2</li>
                <li>GG3DVT11S603-3</li>
            </Aliases>
            <W2KServerData compName="gg3dvt11s603" domain="DVT_D2.DVT_D.DVT"
domainJoined="true"/>
        </CifsServer>
    </Response>
</ResponsePacket>
```

Retrieving CifsShare objects

CifsShareQueryParams specifies parameters for retrieving CifsShare objects.

Query filters include:

- A mover or VDM ID
- A CIFS server NETBIOS name
- A share name

The following examples show a request to query all shares in the system and the corresponding response.

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <Query>
            <CifsShareQueryParams/>
        </Query>
    </Request>
</RequestPacket>
```

Response:

Retrieve CIFsShare object example

CIFS active management

Using the XML API, you can do the following:

- Enable and disable CIFS service on a mover or VDM
- Modify the configuration of CIFS service on a mover or VDM
- Create, modify, and delete CIFS servers
- Create, modify or delete CIFS shares

Some rules for CIFS server creation follow:

- The default CIFS server is the server that results from a "New" operation with an empty list of the interfaces.
- The default CIFS server uses all interfaces not assigned to other CIFS servers on the Data Mover.
- If the default CIFS server already exists, all "New" operations need to specify the interface; otherwise, the command fails. Such a server can use the interfaces currently used by the default CIFS server. However, if you create a CIFS server with an interface being used by another CIFS server, that interface is removed from the preexisting CIFS server and all clients using the interface are disconnected.

You can create a CIFS server on a VDM only when the VDM is in the "loaded" state. If the VDM changes from the loaded state to another state, the CIFS server shuts down. You cannot create a default CIFS server on a VDM; therefore, for a VDM, the list of the interfaces should not be empty.

Enabling and disabling CIFS service

Enable or disable CIFS service on a mover (on a VDM the service is always enabled because of VDM semantics) by starting a request to submit the task ModifyCifsEnabled. The following example shows a request to enable CIFS on a mover:

```
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <ModifyCifsEnabled mover="1" cifsEnabled="true" />
    </StartTask>
  </Request>
</RequestPacket>
```

Creating a Windows 2000 CIFS server

Create a Windows 2000 CIFS server on a mover or a VDM by starting a request to submit the task NewW2KCifsServer. The following example shows a request to create a Windows 2000 server on a mover:

```
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <StartTask>
            <NewW2KCifsServer name="IVAN" compName="IVAN" domain="w2k.acme.com"
localAdminPassword="lapassword">
                <MoverOrVdm mover="1" moverIdIsVdm="false" />
                <Aliases>
                    <li>IVAN_PRIV</li>
                    <li>IVAN_ENT</li>
```

```
                </Aliases>
                <JoinDomain userName="administrator" password="qwerty123"/>
            </NewW2KCifsServer>
        </StartTask>
    </Request>
</RequestPacket>
```

Creating an NT4.0 CIFS server

Create an NT4.0 CIFS server on a mover or a VDM by starting a request to submit the task NewNT40CifsServer. The following example shows a request to create an NT4.0 server on a mover:

```
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <NewNT40CifsServer name="FIT_SERVER" domain="NT4"
       localAdminPassword="lapassword" interfaces="10.173.0.166">
        <MoverOrVdm mover="1" moverIdIsVdm="true" />
        <Aliases>
          <li>FIT_ALIAS1</li>
          <li>FIT_ALIAS2</li>
        </Aliases>
      </NewNT40CifsServer>
    </StartTask>
</Request>
  </RequestPacket>
```

Creating a stand-alone CIFS server

Create a stand-alone CIFS server on a mover or a VDM by starting a request to submit the task NewStandaloneCifsServer. The following example shows a request to create a stand-alone server on a mover:

```
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <NewStandaloneCifsServer name="XYZ_1" workgroup="workgroup"
       localAdminPassword="lapassword" interfaces="10.173.0.165">
        <MoverOrVdm mover="1" moverIdIsVdm="false" />
        <Aliases>
          <li>XYZ_1A</li>
          <li>XYZ_1B</li>
        </Aliases>
      </NewStandaloneCifsServer>
    </StartTask>
  </Request>
</RequestPacket>
```

Modifying a Windows 2000 CIFS server

Windows 2000 CIFS servers are allowed to join or unjoin the domain and to manage local administration. Modify a Windows 2000 CIFS server on a mover or a VDM by starting a request to submit the task ModifyW2KCifsServer. The following example shows a request to modify a Windows 2000 server on a mover:

```
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <ModifyW2KCifsServer mover="3" name="SERVER_1" moverIdIsVdm="true" >
        <DomainSetting userName="adminitrator" password="qwerty123"
joinDomain="true"/>
      </ModifyW2KCifsServer>
    </StartTask>
  </Request>
</RequestPacket>
```

Modifying an NT4.0 CIFS server

The only modify operation allowed for NT4.0 servers is local administrative management. Modify an NT4.0 CIFS server on a mover or a VDM by starting a request to submit the task ModifyNT40CifsServer. The following example shows a request to modify an NT4.0 server on a mover:

```
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <ModifyNT40CifsServer mover="1" name="SERVERNT4" moverIdIsVdm="true">
        <LocalAdmin enabled="true" password="nasadmin"/>
      </ModifyNT40CifsServer>
    </StartTask>
  </Request>
</RequestPacket>
```

Deleting a CIFS server

Delete a CIFS server on a mover or a VDM by starting a request to submit the task DeleteCifsServer. The following example shows a request to delete a CIFS server:

```
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <StartTask>
    <DeleteCifsServer mover="1" name="SERVER_1" moverIdIsVdm="false"/>
  </StartTask>
</RequestPacket>
```

Creating a CIFS share

Create a CIFS share by starting a request to submit the task NewCifsShare. The following example shows a request to create a CIFS share:

```
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <NewCifsShare name="Marketing" path="/acme_fs1/market" comment="Marketing
department">
        <MoverOrVdm mover="3" moverIdIsVdm="false"/>
        <CifsServers>
          <li>ACME_SERVER1</li>
        </CifsServers>
      </NewCifsShare>
    </StartTask>
  </Request>
</RequestPacket>
```

Modifying a CIFS share

Modify a CIFS share by starting a request to submit the task ModifyCifsShare. The following example shows a request to modify a CIFS share:

```
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <ModifyCifsShare mover="3" moverIdIsVdm="false" name="Marketing" >
        <CifsServers>
          <li>ACME_SERVER1</li>
        </CifsServers>
        <NewCifsServers>
          <li>ACME_SERVER2</li>
        </NewCifsServers>
      </ModifyCifsShare>
    </StartTask>
  </Request>
</RequestPacket>
```

Deleting a CIFS share

Delete a CIFS share by starting a request to submit the task DeleteCifsShare. The following example shows a request to delete a CIFS share:

```
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request>
    <StartTask>
      <DeleteCifsShare mover="3" moverIdIsVdm="false" name="Marketing">
        <CifsServers>
          <li>ACME_SERVER2</li>
        </CifsServers>
      </DeleteCifsShare>
    </StartTask>
  </Request>
</RequestPacket>
```

Component management

The schema file Component.xsd defines data structures and queries that reflect physical components of the system.

Component data objects

Components are not necessarily entities that have only physical data associated with them. In this release, some components also contain software settings. The ControlStation component contains associated software configuration information. The mover configuration has been split into a software part, represented by the Mover object and residing in the mover package, and the physical part, represented by MoverHost object and residing in this package.

Celerra systems, storage, and mover hosts have numeric string IDs unique in the context of a specific Celerra system. A Control Station is identified by the slot number in which it is installed. Local Celerra systems (the one that client application is attached to), always have an ID of "0". Currently, viewing remote Celerra systems is not supported. A Celerra system also has a

worldwide, unique identification attribute and a serial number, which, when the system is installed properly, must be present. The worldwide, unique identification should be used by customer applications that need to manage multiple Celerra systems.

Celerra system object

The object CelerraSystem contains the top-level data about the Celerra system. The following example illustrates a typical Celerra system:

```
<CelerraSystem serial="APM00052608291" type="system" version="5.5.15-2"
wwCid="0001901004210055" id="0">
        <Status maxSeverity="ok"/>
</CelerraSystem>
```

Storage system object

The object StorageSystem describes a storage system at the top level. The following example illustrates a typical storage system:

```
<StorageSystem cacheSize="0" diskCount="240" name="000190100421"
serial="000190100421" type="storageSystem" id="1">
        <Status maxSeverity="ok"/>
        <SymmetrixSystemData local="true" model="DMX3-24"
softwareVersion="5771.69" upTime="1192186"/>
</StorageSystem>
```

Control Station object

The object ControlStation describes a Celerra Control Station. There can be no more that two Control Stations per Celerra system. The following example illustrates a typical ControlStation object:

```
<ControlStation address="172.24.147.11" dnsDomain="unix.dvt"
dnsServers="172.24.152.171 172.24.153.171" gateway="172.24.144.173"
hostname="etonic" netmask="255.255.240.0" time="1134638902"
timeZone="America/New_York" type="controlStation" version="5.5.15-2" slot="0">
        <Status maxSeverity="ok"/>
</ControlStation>
```

MoverHost aspect of a mover host

The aspect MoverHost contains the status of the host and the slot number of the Celerra backplane in which it is inserted. The following example illustrates a MoverHost aspect:

```
<MoverHost mover="2" slot="3" id="2">
         <Status maxSeverity="ok"/>
</MoverHost>
```

MoverMotherboard aspect of a mover host

The aspect MoverMotherboard of the mover host contains the data about the motherboard of a mover host. The following example illustrates a MoverMotherboard aspect:

```
<MoverMotherboard boardType="CMB-Hammerhead" busSpeed="800" cpuSpeed="3400"
cpuType="Intel Pentium 4" memorySize="4094" id="2"/>
```

Physical device object

The object PhysicalDevice describes a mover host I/O physical device. Devices provide an interface to the outside world (network devices), and devices attach to storage (normally Fibre Channel devices).

The following example illustrates a network device:

```
<PhysicalDevice description="Broadcom Gigabit" irq="30" portNumber="1"
type="ethernet" moverHost="2" name="cge0">
          <NetworkDeviceData allowedSpeeds="FD1000 HD1000 FD100 HD100 FD10 HD10
auto"/>
</PhysicalDevice>
```

The following example illustrates a Fibre Channel device:

```
<PhysicalDevice description="Agilent" irq="32" portNumber="0" type="fc"
moverHost="2" name="fcp-1">
          <FibreChannelDeviceData portWWN="5006016939a003d0"/>
</PhysicalDevice>
```

PhysicalDevice is not an aspect of a mover host; however, the entire set of physical devices is an aspect of a mover host. In this version of the XML API, you cannot query a specific PhysicalDevice object on a mover.

Fibre Channel descriptor aspect

The FcDescriptor aspect of a mover host describes the back-end Fibre Channel connections of a mover host to Symmetrix director ports or to CLARiiON storage processors. In combination with the PhysicalDevice object, it allows you to trace the data flow paths from disk type volumes to storage devices. It also allows Fibre Channel management applications to match the data gathered by means other than the XML API to the appropriate Celerra Fibre Channel ports.

The following example illustrates an FcDescriptor aspect:

```
<FcDescriptor moverHost="2">
    <FcConnectionSet portWWN="5006016839a003d0">
       <StorageEndpoint directorNumber="46" directorType="FC" port="0"
portWWN="5006048ad52cf14c" storage="1"/>
       <StorageEndpoint directorNumber="29" directorType="FC" port="0"
portWWN="50060482bfd00a1b" storage="2"/>
       <StorageEndpoint directorNumber="20" directorType="FC" port="0"
portWWN="50060482bfd00a12" storage="2"/>
       <StorageEndpoint directorNumber="4" directorType="FC" port="0"
portWWN="50060482bfd00a02" storage="2"/>
       <StorageEndpoint directorNumber="37" directorType="FC" port="0"
portWWN="5006048ad52cf143" storage="1"/>
       <ScsiRange scsiChainStart="176" scsiNumChains="16" />
    </FcConnectionSet>
    <FcConnectionSet portWWN="5006016939a003d0">
       <StorageEndpoint directorNumber="47" directorType="FC" port="0"
portWWN="5006048ad52cf14d" storage="1"/>
       <StorageEndpoint directorNumber="36" directorType="FC" port="0"
portWWN="5006048ad52cf142" storage="1"/>
```

```
       <StorageEndpoint directorNumber="31" directorType="FC" port="0"
portWWN="50060482bfd00a1d" storage="2"/>
       <StorageEndpoint directorNumber="22" directorType="FC" port="0"
portWWN="50060482bfd00a14" storage="2"/>
       <StorageEndpoint directorNumber="13" directorType="FC" port="0"
portWWN="50060482bfd00a0b" storage="2"/>
       <ScsiRange scsiChainStart="128" scsiNumChains="16" />
    </FcConnectionSet>
    <FcConnectionSet portWWN="5006016a39a003d0">
       <ScsiRange scsiChainStart="144" scsiNumChains="16"/>
    </FcConnectionSet>
    <FcConnectionSet portWWN="5006016b39a003d0">
       <ScsiRange scsiChainStart="160" scsiNumChains="16" />
    </FcConnectionSet>
</FcDescriptor
```

Component queries

The following section describes the functions of system parameters.

Retrieving Celerra objects

The CelerraSystemQueryParams object specifies parameters for retrieving CelerraSystem objects. A Query filter is the system ID.

Note: Querying remote Celerra systems is not supported in this release.

The following examples show how to retrieve Celerra objects and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
     <Request>
         <Query>
             <CelerraSystemQueryParams />
         </Query>
     </Request>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Response>
        <QueryStatus maxSeverity="ok"/>
        <CelerraSystem productName="NSX" serial="APM00052608291" type="system"
version="5.5.18-2 " wwCid="0001901004210055" celerra="0">
            <Status maxSeverity="ok"/>
        </CelerraSystem>
    </Response>
</ResponsePacket>
```

Retrieving Control Station objects

The ControlStationQueryParams object specifies parameters for retrieving ControlStation objects. A Query filter is the Control Station ID (physical slot number).

The following examples show how to retrieve Control Station objects and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <Query>
            <ControlStationQueryParams/>
        </Query>
    </Request>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Response>
        <QueryStatus maxSeverity="ok"/>
        <ControlStation address="172.24.147.11" dnsDomain="unix.dvt"
dnsServers="172.24.152.171 172.24.153.171" gateway="172.24.144.173"
hostname="etonic" netmask="255.255.240.0" time="1139847795"
timeZone="America/New_York" type="controlStation" version="5.5.18-2 " slot="0">
            <Status maxSeverity="ok"/>
        </ControlStation>
    </Response>
</ResponsePacket>
```

Retrieving storage system objects

The StorageSystemQueryParams object specifies parameters for retrieving StorageSystem objects.

Query filter is the ID of the storage system on this Celerra system.

The following examples show how to retrieve storage system objects and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <Query>
            <StorageSystemQueryParams/>
        </Query>
    </Request>
```

```
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Response>
        <QueryStatus maxSeverity="ok"/>
        <StorageSystem cacheSize="0" diskCount="240" name="000190100421"
serial="000190100421" type="storageSystem" storage="1">
            <Status maxSeverity="ok"/>
            <SymmetrixSystemData local="true" model="DMX3-24"
softwareVersion="5771.74" upTime="2938828"/>
        </StorageSystem>
        <StorageSystem cacheSize="0" diskCount="96" name="000184500264"
serial="000184500264" type="storageSystem" storage="2">
            <Status maxSeverity="ok"/>
            <SymmetrixSystemData local="true" model="8430"
softwareVersion="5568.67" upTime="6406266"/>
        </StorageSystem>
    </Response>
</ResponsePacket>
```

Retrieving mover host aspects

The MoverHostQueryParams object specifies parameters for retrieving all aspects associated with mover host component.

Query filters include:

- An AspectSelection that selectively retrieves MoverHost, MoverMotherboard, PhysicalDevice set, and FcDescriptor aspects
- A mover host ID

The following examples show how to retrieve mover host aspects and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xsi:schemaLocation='http://www.emc.com/schemas/celerra/xml_api
file:///C:/PROJECTS/BURG1/API_SCHEMA/xsd/APISchema.xsd'>
    <Request>
        <Query>
           <MoverHostQueryParams>
              <AspectSelection moverHosts="true" moverMotherboards="true"
physicalDevices="true" fibreChannelConnections="true"/>
           </MoverHostQueryParams>
        </Query>
    </Request>
</RequestPacket>
```

Response:

Retrieve mover host aspects example

CLARiiON management

The schema file Clariion.xsd defines data structures and operations related to CLARiiON backend configuration and statistics information.

CLARiiON data bjects

CLARiiON data objects are divided into two major categories: configuration and statistics

CLARiiON configuration data object

This object describes the CLARiiON configuration. A ClariionConfig object is identified by the CLARiiON storage ID. A special object, CelerraClariionRef, represents a reference to a generic ClariionConfig.

The ClariionConfig object describes the basic generic CLARiiON attributes like the storage ID, name, UID, model number, model type, number of CLARiiON devices, number of physical disks, number of visible devices, number of RAID groups, number of storage groups, snapshot, highwaterMark, and so on.

```
<ClariionConfig cachePageSize="8" clariionDevices="9" highWaterMark="90.0"
lowWaterMark="70.0" modelNumber="Model600" modelType="rackmount"
name="WRE00460100045" physicalDisks="14" raidGroups="2" snapshot="1159309074"
storageGroups="0" uid="50060160806000340000000000000000"
unassignedCachePages="0" visibleDevices="1" clariion="WRE00460100045"/>
```

CLARiiON storage processor (SP) configuration aspect

This object describes the CLARiiON storage processor (SP) configuration aspect. The Clariion SPConfig object describes a CLARiiON storage processor configuration using the attributes CLARiiON storage ID, SP ID, list of ports having port number, port UID, switch ID information, signature, serial number, PromRev, agentRev, read/write cache, and so on.

```
<ClariionSPConfig agentRev="6.16.0 (4.80)" freeMemorySize="0"
microcodeRev="2.16.600.5.004" physicalMemorySize="2048" promRev="0.00.00"
raid3MemorySize="0" readCacheSize="1057" serialNumber="LKE00020500881"
signature="545649" systemBufferSize="551" writeCacheSize="440" id="A"
storage="1"/>
```

CLARiiON storage processor (SP) status aspect

This object describes the CLARiiON storage processor (SP) status aspect. The ClariionSpStatus object describes CLARiiON storage processor status using the attributes CLARiiON storage ID, SP ID, list of port statuses, state, and read/write cache state.

```
<ClariionSpStatus readCacheState="enabled" writeCacheState="enabled" id="A"
storage="1"/>
```

CLARiiON disk configuration aspect

This object describes the CLARiiON disk configuration aspect. The ClariionDiskConfig object is identified by the CLARiiON storage ID and disk name. A special object, ClariionDiskRef, represents a reference for this object.

The ClariionDiskConfig object describes CLARiiON disk configurations using the attributes CLARiiON storage ID, disk name, bus number, enclosure number, disk number, state, vendor ID, product ID revision, serial number, capacity, and so on.

```
<ClariionDiskConfig bus="0" capacity="139751424" diskNumber="0"
enclosureNumber="0" productId="ST373405 CLAR72" remappedBlocks="-1"
revision="4A24" serialNumber="3EK1503Z" state="enabled"
usedCapacity="101321689" vendorId="SEAGATE" name="0_0_0" storage="1"/>
```

CLARiiON RAID group configuration data object

This object describes the CLARiiON RAID group configuration. The ClariionRaidGroupConfig object is identified by CLARiiON storage ID and CLARiiON RAID group ID. A special object, ClariionRaidGroupRef, represents a reference for this object.

The ClariionRaidGroupConfig object describes CLARiiON RAID group configurations using the attributes CLARiiON storage ID, CLARiiON RAID group ID, RAID type, state, raw capacity, logical capacity, used capacity, disk, and devices list.

```
<ClariionRaidGroupConfig devices="0 1 2 3 4 5 16 17" disks="0_0_0, 0_0_1,
0_0_2, 0_0_3, 0_0_4" logicalCapacity="506609984" raidType="raid-5"
rawCapacity="633262480" state="other" usedCapacity="506608640" id="0000"
storage="1"/>
```

CLARiiON device configuration aspect

This object describes the ClariionDeviceConfig. The ClariionDeviceConfig object describes CLARiiON device configurations using the attributes CLARiiON storage ID, device ID, user defined name, device UID, RAID group ID, type, capacity, first stripe number, current owner, default owner, and so on.

```
<ClariionDeviceConfig capacity="23068672" currentOwner="B" defaultOwner="A"
deviceUid="60060160715308008ce52f5373cfda11" firstStripeNumber="0"
idleDelayTime="20" idleThreshold="0" maxPrefetch="4096" prefetchDisable="4097"
prefetchIdleCount="40" raidGroupId="0000" stripeElemSize="128"
usersDefinedName="LUN 0" device="0" storage="1"/>
```

CLARiiON device status aspect

This object describes the CLARiiON device status aspect. The ClariionDeviceStatus object describes CLARiiON device statuses using the attributes CLARiiON storage ID, device ID, autoTrspass, autoAssignment, Read/write cache, variable length Prefetch, state, and isPrivate.

```
<ClariionDeviceStatus autoAssignment="disabled" autoTrespass="disabled"
isPrivate="disabled" readCache="other" state="bound"
variableLengthPrefetch="other" writeCache="other" device="0" storage="1"/>
```

CLARiiON storage processor (SP) statistics aspect

This object describes the CLARiiON storage processor (SP) statistics aspect. The ClariionSpStats aspect describes the storage processor statistics using the attributes CLARiiON storage ID, storage processor ID, read/write requests, block read/written, dirtypages, and queuedArrivals.

```
<ClariionSpStats clariion="1">
           <Sample time="1157944559">
               <SPStat blocksRead="1528855" blocksWritten="142945706"
dirtyPages="1" id="A" queuedArrivals="32624444" readReqs="36185"
writeReqs="7836086"/>
           </Sample>
</ClariionSpStats>
```

CLARiiON device statistics object

The ClariionDeviceStats object describes CLARiiON logical device statistics using the attributes CLARiiON storage ID, read/write histograms, device name, read/write requests, read/write blocks, read/write cache hits, read cache misses, prefetchedBlocks, and forcedFlushes.

```
<ClariionDeviceStats clariion="1">
           <Sample time="1159321990">
               <DevStat deviceName="0000" forcedFlushes="0"
prefetchedBlocks="189568" readBlocks="915590" readCacheHits="45982"
readCacheMisses="0" readReqs="49249" writeBlocks="2607005"
writeCacheHits="188671" writeReqs="393961">
                   <readHistogram h0="13248" h1="17" h2="7320" h3="15440"
h4="2635" h5="38" h6="10544" h7="1" h8="0" h9="0"/>
                   <writeHistogram h0="270167" h1="32194" h2="5235" h3="10309"
h4="67483" h5="38" h6="1051" h7="7402" h8="2" h9="80"/>
               </DevStat>
               <DevStat deviceName="0001" forcedFlushes="0"
prefetchedBlocks="132160" readBlocks="132134" readCacheHits="802"
readCacheMisses="0" readReqs="1082" writeBlocks="465923" writeCacheHits="85"
writeReqs="10466">
                   <readHistogram h0="37" h1="0" h2="0" h3="0" h4="14" h5="4"
h6="0" h7="1027" h8="0" h9="0"/>
                   <writeHistogram h0="2782" h1="4158" h2="0" h3="0" h4="382"
h5="6" h6="0" h7="3079" h8="9" h9="25"/>
               </DevStat>
           </Sample>
</ClariionDeviceStats>
```

CLARiiON disk statistics object

The ClariionDiskStats object describes the disks statistics using the attributes CLARiiON storage ID, disk name, and read/write requests.

```
<ClariionDiskStats clariion="1">
           <Sample time="1159336235">
               <DiskStat diskName="0_0_0" readReqs="4743293"
writeReqs="2965383"/>
               <DiskStat diskName="0_0_1" readReqs="4739103"
writeReqs="2954059"/>
```

```
                <DiskStat diskName="0_0_2" readReqs="4765592"
writeReqs="1558773"/>
                <DiskStat diskName="0_0_3" readReqs="4717613"
writeReqs="1485449"/>
                <DiskStat diskName="0_0_4" readReqs="4748425"
writeReqs="1510821"/>
                <DiskStat diskName="0_0_5" readReqs="0" writeReqs="0"/>
                <DiskStat diskName="0_0_6" readReqs="2632587" writeReqs="4"/>
                <DiskStat diskName="0_0_7" readReqs="2632585" writeReqs="4"/>
                <DiskStat diskName="0_0_8" readReqs="2632589" writeReqs="4"/>
                <DiskStat diskName="0_0_9" readReqs="2632585" writeReqs="4"/>
                <DiskStat diskName="0_0_10" readReqs="2632585" writeReqs="4"/>
                <DiskStat diskName="0_0_11" readReqs="2" writeReqs="2"/>
                <DiskStat diskName="0_0_12" readReqs="2" writeReqs="2"/>
                <DiskStat diskName="0_0_13" readReqs="2" writeReqs="2"/>
            </Sample>
</ClariionDiskStats>
```

CLARiiON SP statistics object

The ClariionSpStats object describes the storage processor statistics using the attributes CLARiiON storage ID, storage processor ID, read/write requests, block read/written, dirtypages, and queuedArrivals.

```
<ClariionSpStats clariion="1">
            <Sample time="1157944559">
                <SPStat blocksRead="1528855" blocksWritten="142945706"
dirtyPages="1" id="A" queuedArrivals="32624444" readReqs="36185"
writeReqs="7836086"/>
            </Sample>
</ClariionSpStats>
```

CLARiiON queries

The following section describes CLARiiON queries.

Querying a generic CLARiiON configuration

The ClariionGeneralConfigQueryParams object specifies parameters for retrieving the ClariionConfig object.

The following examples show a request that queries a CLARiiON generic configuration and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
    <RequestEx>
        <Query>
<ClariionGeneralConfigQueryParams
clariion="1"></ClariionGeneralConfigQueryParams>
        </Query>
     </RequestEx>
```

```
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx>
       <ClariionConfig cachePageSize="8" clariionDevices="9"
highWaterMark="90.0" lowWaterMark="70.0" modelNumber="Model600"
modelType="rackmount" name="WRE00460100045" physicalDisks="14" raidGroups="2"
snapshot="1159309074" storageGroups="0" uid="50060160806000340000000000000000"
unassignedCachePages="0" visibleDevices="1" clariion="WRE00460100045"/>
   </ResponseEx>
</ResponsePacket>
```

Querying a CLARiiON storage processor (SP) configuration

The ClariionSpQueryParams object specifies parameters for retrieving all aspects of a CLARiiON storage processor. Objects returned are either of type ClariionSpConfig or ClariionSpStatus, depending on the aspect of the query.

Query filters include:

- An Aspect selection element specifying the aspect needed by the application program
- Storage processor ID

If the filter, storage processor ID, is provided in the query, the configuration for that particular storage processor is returned.

The following examples show a request that queries a CLARiiON storage processor configuration and status and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
     <RequestEx>
         <Query>
<ClariionSpQueryParams clariion="1">
     <AspectSelection  status="true" config="true"></AspectSelection>
  </ClariionSpQueryParams>
           </Query>
       </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx>
       <ClariionSPConfig agentRev="6.16.0 (4.80)" freeMemorySize="0"
microcodeRev="2.16.600.5.004" physicalMemorySize="2048" promRev="0.00.00"
raid3MemorySize="0" readCacheSize="1057" serialNumber="LKE00020500881"
```

```
signature="545649" systemBufferSize="551" writeCacheSize="440" id="A"
storage="1"/>
       <ClariionSpStatus readCacheState="enabled" writeCacheState="enabled"
id="A" storage="1"/>
       <ClariionSPConfig agentRev="6.16.0 (4.80)" freeMemorySize="0"
microcodeRev="2.16.600.5.004" physicalMemorySize="1023" promRev="3.42.00"
raid3MemorySize="0" readCacheSize="32" serialNumber="LKE00021202058"
signature="575498" systemBufferSize="551" writeCacheSize="440" id="B"
storage="1"/>
       <ClariionSpStatus readCacheState="enabled" writeCacheState="enabled"
id="B" storage="1"/>
   </ResponseEx>
</ResponsePacket>
```

Querying a CLARiiON disk configuration

The ClariionDiskQueryParams object specifies parameters for retrieving all CLARiiON disks. Objects returned are of type ClariionDiskConfig.

Query filters include: CLARiiON disk name.

If the filter, CLARiiON disk name, is provided in the query, the configuration for that particular disk is returned.

The following examples show a request that queries a CLARiiON disk configuration and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
     <RequestEx>
          <Query>
  <ClariionDiskQueryParams clariion="1"></ClariionDiskQueryParams>
          </Query>
    </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx>
       <ClariionDiskConfig bus="0" capacity="139751424" diskNumber="0"
enclosureNumber="0" productId="ST373405 CLAR72" remappedBlocks="-1"
revision="4A24" serialNumber="3EK1503Z" state="enabled"
usedCapacity="101321689" vendorId="SEAGATE" name="0_0_0" storage="1"/>
       <ClariionDiskConfig bus="0" capacity="139751424" diskNumber="1"
enclosureNumber="0" productId="ST373405 CLAR72" remappedBlocks="-1"
revision="4A24" serialNumber="3EK1545B" state="enabled"
usedCapacity="101321689" vendorId="SEAGATE" name="0_0_1" storage="1"/>
       <ClariionDiskConfig bus="0" capacity="139751424" diskNumber="2"
enclosureNumber="0" productId="ST373405 CLAR72" remappedBlocks="-1"
revision="4A24" serialNumber="3EK14AK4" state="enabled"
usedCapacity="101321689" vendorId="SEAGATE" name="0_0_2" storage="1"/>
```

```
       <ClariionDiskConfig bus="0" capacity="139751424" diskNumber="3"
enclosureNumber="0" productId="ST373405 CLAR72" remappedBlocks="-1"
revision="4A24" serialNumber="3EK12W7G" state="enabled"
usedCapacity="101321689" vendorId="SEAGATE" name="0_0_3" storage="1"/>
       <ClariionDiskConfig bus="0" capacity="139751424" diskNumber="4"
enclosureNumber="0" productId="ST373405 CLAR72" remappedBlocks="-1"
revision="4A24" serialNumber="3EK0Z737" state="enabled"
usedCapacity="101321689" vendorId="SEAGATE" name="0_0_4" storage="1"/>
       <ClariionDiskConfig bus="0" capacity="139751424" diskNumber="5"
enclosureNumber="0" productId="ST373405 CLAR72" remappedBlocks="-1"
revision="4A24" serialNumber="3EK14M4V" state="hot-spare"
usedCapacity="139681792" vendorId="SEAGATE" name="0_0_5" storage="1"/>
       <ClariionDiskConfig bus="0" capacity="139751424" diskNumber="6"
enclosureNumber="0" productId="ST373405 CLAR72" remappedBlocks="-1"
revision="4A24" serialNumber="3EK151M3" state="unbound" usedCapacity="0"
vendorId="SEAGATE" name="0_0_6" storage="1"/>
       <ClariionDiskConfig bus="0" capacity="139751424" diskNumber="7"
enclosureNumber="0" productId="ST373405 CLAR72" remappedBlocks="-1"
revision="4A24" serialNumber="3EK15ALE" state="unbound" usedCapacity="0"
vendorId="SEAGATE" name="0_0_7" storage="1"/>
       <ClariionDiskConfig bus="0" capacity="139751424" diskNumber="8"
enclosureNumber="0" productId="ST373405 CLAR72" remappedBlocks="-1"
revision="4A24" serialNumber="3EK11TLQ" state="unbound" usedCapacity="0"
vendorId="SEAGATE" name="0_0_8" storage="1"/>
       <ClariionDiskConfig bus="0" capacity="139751424" diskNumber="9"
enclosureNumber="0" productId="ST373405 CLAR72" remappedBlocks="-1"
revision="4A24" serialNumber="3EK14PV4" state="unbound" usedCapacity="0"
vendorId="SEAGATE" name="0_0_9" storage="1"/>
       <ClariionDiskConfig bus="0" capacity="139751424" diskNumber="10"
enclosureNumber="0" productId="ST373405 CLAR72" remappedBlocks="-1"
revision="4A24" serialNumber="3EK142VY" state="unbound" usedCapacity="0"
vendorId="SEAGATE" name="0_0_10" storage="1"/>
   <ClariionDiskConfig bus="0" capacity="139751424" diskNumber="11"
enclosureNumber="0" productId="ST373405 CLAR72" remappedBlocks="-1"
revision="4A24" serialNumber="3EK11F0P" state="unbound" usedCapacity="0"
vendorId="SEAGATE" name="0_0_11" storage="1"/>
       <ClariionDiskConfig bus="0" capacity="139751424" diskNumber="12"
enclosureNumber="0" productId="ST373405 CLAR72" remappedBlocks="-1"
revision="4A24" serialNumber="3EK11PRW" state="unbound" usedCapacity="0"
vendorId="SEAGATE" name="0_0_12" storage="1"/>
       <ClariionDiskConfig bus="0" capacity="139751424" diskNumber="13"
enclosureNumber="0" productId="ST373405 CLAR72" remappedBlocks="-1"
revision="4A24" serialNumber="3EK156A3" state="unbound" usedCapacity="0"
vendorId="SEAGATE" name="0_0_13" storage="1"/>
   </ResponseEx>
</ResponsePacket>
```

Querying a CLARiiON RAID group configuration

The ClariionRaidGroupQueryParams object specifies parameters for retrieving all CLARiiON RAID groups. Objects returned are of type ClariionRaidGroupConfig.

Query filters include: CLARiiON RAID group ID.

If the filter, CLARiiON RAID group ID, is provided in the query, the configuration for that particular RAID group is returned if present.

The following examples show a request that queries a CLARiiON RAID group configuration and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
     <RequestEx>
         <Query>
             <ClariionRaidGroupQueryParams clariion="1" raidGroup="200"/>
         </Query>
    </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx>
       <ClariionRaidGroupConfig devices="200" disks="0_0_5"
logicalCapacity="139681874" raidType="spare" rawCapacity="139681874"
state="other" usedCapacity="139681792" id="0200" storage="1"/>
   </ResponseEx>
</ResponsePacket>
```

Querying CLARiiON device configurations

The ClariionDeviceQueryParams object specifies parameters for retrieving all aspects of a CLARiiON device configuration. Objects returned are either of type ClariionDeviceConfig or ClariionDeviceStatus, depending on the aspect of the query.

Query filters include:

- An Aspect selection element specifying the aspect needed by the application program
- CLARiiON device range

The following examples show a request that queries a CLARiiON device configuration and status and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
    <RequestEx>
         <Query>
 <ClariionDeviceQueryParams clariion="1">
      <AspectSelection status="true" config="true"></AspectSelection>
           <deviceRange start="A" size="10"></deviceRange>
 </ClariionDeviceQueryParams>
         </Query>
     </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx>
      <ClariionDeviceConfig capacity="955898880" currentOwner="B"
defaultOwner="B" deviceUid="6006016001b71a00e2c527c8d906db11"
firstStripeNumber="0" idleDelayTime="20" idleThreshold="0" maxPrefetch="4096"
prefetchDisable="4097" prefetchIdleCount="40" raidGroupId="0000"
stripeElemSize="128" type="other" usersDefinedName="LUN 16" device="16"
storage="1"/>
      <ClariionDeviceStatus autoAssignment="disabled" autoTrespass="disabled"
isPrivate="disabled" readCache="other" state="bound"
variableLengthPrefetch="other" writeCache="other" device="16" storage="1"/>
      <ClariionDeviceConfig capacity="955898880" currentOwner="A"
defaultOwner="A" deviceUid="6006016001b71a00e3c527c8d906db11"
firstStripeNumber="0" idleDelayTime="20" idleThreshold="0" maxPrefetch="4096"
prefetchDisable="4097" prefetchIdleCount="40" raidGroupId="0000"
stripeElemSize="128" type="other" usersDefinedName="LUN 17" device="17"
storage="1"/>
      <ClariionDeviceStatus autoAssignment="disabled" autoTrespass="disabled"
isPrivate="disabled" readCache="other" state="bound"
variableLengthPrefetch="other" writeCache="other" device="17" storage="1"/>
      <ClariionDeviceConfig capacity="1125628928" currentOwner="B"
defaultOwner="B" deviceUid="6006016001b71a0086eeb8d3d906db11"
firstStripeNumber="0" idleDelayTime="20" idleThreshold="0" maxPrefetch="4096"
prefetchDisable="4097" prefetchIdleCount="40" raidGroupId="0010"
stripeElemSize="128" type="other" usersDefinedName="LUN 18" device="18"
storage="1"/>
      <ClariionDeviceStatus autoAssignment="disabled" autoTrespass="disabled"
isPrivate="disabled" readCache="other" state="bound"
variableLengthPrefetch="other" writeCache="other" device="18" storage="1"/>
      <ClariionDeviceConfig capacity="1125628928" currentOwner="A"
defaultOwner="A" deviceUid="6006016001b71a0087eeb8d3d906db11"
firstStripeNumber="0" idleDelayTime="20" idleThreshold="0" maxPrefetch="4096"
prefetchDisable="4097" prefetchIdleCount="40" raidGroupId="0010"
stripeElemSize="128" type="other" usersDefinedName="LUN 19" device="19"
storage="1"/>
      <ClariionDeviceStatus autoAssignment="disabled" autoTrespass="disabled"
isPrivate="disabled" readCache="other" state="bound"
variableLengthPrefetch="other" writeCache="other" device="19" storage="1"/>
   </ResponseEx>
</ResponsePacket>
```

Symmetrix management

Symemetrix data objects

Symmetrix data objects are divided into two major categories: configuration and statistics.

Symmetrix overall configuration object

This object describes the SymmetrixConfig data object. A SymmetrixConfig is identified by the Symmetrix storage ID. A special object, CelerraSymmRef, represents a reference to the generic SymmetrixConfig.

The SymmetrixConfig object describes the basic generic Symmetrix attributes such as the storage ID, name, model, serial number, mcVersion, mcVersionNum, mcPatchLevel, mcPatchDate, mcDate,pwronTime, iplTime, and so on.

```
<SymmetrixConfig cacheSize="2048" cacheSlots="53937"
fastIplTime="1135753981000" ident="Symm4" iplTime="1118652132000"
logicalDisks="35" maxDaWPS="21595" maxDeviceWPS="10160" maxWPS="43191"
mcDate="1078203600000" mcPatchDate="1078203600000" mcPatchLevel="44"
mcVersion="5267" mcVersionNum="1493AA01" model="3300" name="000182600982"
permaCacheSlots="0" physicalDisks="16" powerPathDevices="0"
pwronTime="1135753981000" serialNumber="000182600982"
snapShotAt="1148094877337" symmetrix="1"/>
```

Physical disk object

This object describes the PhysicalDisk data object. A PhysicalDisk is identified by a director number, interface, and scsiId. A special object, PhysicalDiskRef, represents a reference to a Symmertix physical disk.

The physical disk object describes the Symmetrix physical disk attributes such as the director number, interface, scsiId, productId, revisionId, rpm, disk capacity, blockSize, hypers, and so on.

```
<PhysicalDisk blockSize="512" diskCapacity="69838" hypers="6"
productId="SX173404LC" productRevision="CHET_73" rpm="10033" symmetrix="1"
usedCapacity="143028400" vendorId="SEAGATE " director="1" interface="C"
scsiId="1"/>
```

Symmetrix director configuration aspect

The SymmDirectorConfig aspect describes the basic configuration of the director using attributes such as director type, number of ports, and director type specific information. A special object, SymmDirectorRef, represents a reference to a director configuration object.

```
<SymmDirectorConfig ports="2" type="channel" director="1" symmetrix="1">
        <DAData hyperVolumes="76" scsiCapability="narrow"/>
 </SymmDirectorConfig>
 <SymmDirectorConfig ports="2" type="channel" director="2" symmetrix="1">
        <DAData hyperVolumes="78" scsiCapability="narrow"/>
</SymmDirectorConfig>
```

Symmetrix Director status aspect

The SymmDirectorStatus aspect describes the status of the director using attributes such as status, list of ports each element describes, port number, and port status information. A special object, SymmDirectorRef, represents a reference to a director status object.

```
<SymmDirectorStatus dirStatus="on-line" director="2" symmetrix="1">
       <portStatus>
           <li port="0" status="ready"/>
           <li port="1" status="ready"/>
       </portStatus>
</SymmDirectorStatus>
<SymmDirectorStatus dirStatus="on-line" director="3" symmetrix="1">
       <portStatus>
            <li port="0" status="ready"/>
```

```
       </portStatus>
</SymmDirectorStatus>
```

Symmetrix device frontend configuration aspect

The SymmDeviceFrontendConfig aspect describes device access information using attributes such as director number, port, LUN, virtual bus, target. A special object, SymmDeviceRef, represents a reference to a frontend configuration object.

```
<SymmDeviceFrontendConfig device="0F" symmetrix="1">
    <DeviceAccessInfo director="4" lun="18" port="1" virtualBus="0"/>
     <DeviceAccessInfo director="15" lun="18" port="1" virtualBus="0"/>
     <DeviceAccessInfo director="16" lun="18" port="1" virtualBus="0"/>
     <DeviceAccessInfo director="17" lun="18" port="1" virtualBus="0"/>
     <DeviceAccessInfo director="20" lun="18" port="0" virtualBus="0"/>
    <DeviceAccessInfo director="31" lun="18" port="0" virtualBus="0"/>
    <DeviceAccessInfo director="32" lun="18" port="0" virtualBus="0"/>
    <DeviceAccessInfo director="33" lun="18" port="0" virtualBus="0"/>
</SymmDeviceFrontendConfig>
```

Symmetrix device hyper disk configuration aspect

The SymmDeviceHyperDiskConfig aspect describes the information of a logical disk using attributes such as hyper type, status, RAID groups, mirror number, hyper number, member number first block address, and so on. A special object, SymmDeviceRef, represents a reference to a hyper disk configuration object.

```
<SymmDeviceHyperDiskConfig device="00" symmetrix="1">
      <hyper mirrorNum="2" raidGroup="0" status="ready" type="other"
hyperNumber="0"/>
</SymmDeviceHyperDiskConfig>
<SymmDeviceHyperDiskConfig device="01" symmetrix="1">
    <hyper mirrorNum="2" raidGroup="0" status="ready" type="other"
hyperNumber="0"/>
</SymmDeviceHyperDiskConfig>
```

Symmetrix device configuration aspect

The SymmetrixDeviceConfig aspect describes the information about the Symmetrix logical device configuration using attributes such as device configuration, capacity, cylinders, block size, bcv device list, and attributes specific to device types. A special object, SymmDeviceRef, represents a reference to a device configuration object.

```
<SymmetrixDeviceConfig bcvDevices="" blockSize="512" capacity="2077"
config="mirror-2" cylinders="4430" device="0A" symmetrix="1"/>
<SymmetrixDeviceConfig bcvDevices="" blockSize="512" capacity="2077"
config="mirror-2" cylinders="4430" device="0B" symmetrix="1"/>
<SymmetrixDeviceConfig bcvDevices="" blockSize="512" capacity="2077"
config="mirror-2" cylinders="4430" device="0C" symmetrix="1"/>
<SymmetrixDeviceConfig bcvDevices="" blockSize="512" capacity="3"
config="mirror-2" cylinders="7" device="0D" symmetrix="1"/>
```

Symmetrix device status aspect

The SymmetrixDeviceStatus aspect describes the information about Symmetrix logical device status. A special object, SymmDeviceRef, represents a reference to a device status object.

```
<SymmetrixDeviceStatus status="ready" device="0A" symmetrix="1">
     <BcvDevice bcvStatus="invalid"/>
</SymmetrixDeviceStatus>
<SymmetrixDeviceStatus status="ready" device="0B" symmetrix="1">
     <BcvDevice bcvStatus="invalid"/>
</SymmetrixDeviceStatus>
```

Symmetrix device statistics data object

The SymmDevicesStats object describes the status of the Symmetrix logical devices using the attributes Symmetrix storage ID and statistics information for each device, including total request, read/write requests, read/write hits, prefetchedTracks, destagedTracks, and so on.

```
<SymmDevicesStats symmetrix="1">
           <Sample time="1157940209">
               <DeviceStat deferredWrites="2142293" delayedDeferredWrites="0"
destagedTracks="675354" formatPendingTracks="0" hits="3192654"
prefetchedTracks="105022" readHits="1050962" readRequests="11331988480"
sequentialReadRequests="37052" totalRequests="44733292544" writeHits="2141692"
writePendingLimit="8248" writePendingTracks="1" writeRequests="33401304064"
device="0"/>
               <DeviceStat deferredWrites="2142293" delayedDeferredWrites="0"
destagedTracks="675354" formatPendingTracks="0" hits="3192654"
prefetchedTracks="105022" readHits="1050962" readRequests="11331988480"
sequentialReadRequests="37052" totalRequests="44733292544" writeHits="2141692"
writePendingLimit="8248" writePendingTracks="1" writeRequests="33401304064"
device="1"/>
           </Sample>
 </SymmDevicesStats>
```

Symmetrix director frontend statistics aspect

The SymmFrontEndStats object describes all statistics or specific director statistics using the attributes Symmetrix ID, ios, total request, read/write request, hit, read misses, slot collisions, and director number (in case of specific director statistics).

```
<SymmFrontEndStats symmetrix="1">
       <Sample time="1157939999">
          <FrontEnd deviceWPDisconnects="107976" hits="1008128041"
ios="1061913627" permaCacheRequests="0" readMisses="4976791"
readRequests="440111124" slotCollisions="615903" systemWPDisconnects="1"
totalRequests="1014574187" writeRequests="574463063"/>
   </Sample>
</SymmFrontEndStats>
```

Symmetrix director backend statistics aspect

The SymmBackEndStats object describes all statistics or specific director statistics using the attributes Symmetrix ID, ios, total request, read/write request, prefetchTask, prefetchShortMiss,

prefatchLongMiss, prefetchTracksUsed, prefetchTrackNotUsed, and director number (in case of specific director statistics).

```
<SymmBackEndStats symmetrix="1">
      <Sample time="1157939896">
           <BackEnd director="1" ios="55575912" prefetchLongMiss="9091"
prefetchMismatches="0" prefetchRestarts="17478" prefetchShortMiss="11129"
prefetchTasks="0" prefetchTracksNotUsed="104159" prefetchTracksUsed="860057"
readRequests="979438" totalRequests="59062707" writeRequests="58083269"/>
      </Sample>
</SymmBackEndStats>
```

Symmetrix director port statistics aspect

The SymmPortStats object describes all statistics or specific director statistics using the attributes Symmetrix ID, ios, port, and blocks transferred.

```
<SymmPortStats symmetrix="1">
     <Sample time="1157940059">
           <DirectorPortStats director="3">
                <PortStats blocks="0" ios="0" port="1"/>
           </DirectorPortStats>
           <DirectorPortStats director="4">
               <PortStats blocks="386061907" ios="224151556" port="1"/>
           </DirectorPortStats>
      </Sample>
</SymmPortStats>
```

Symmetrix queries

The following section describes Symmetrix queries.

Querying generic Symmetrix configuration

SymmGeneralConfigQueryParams object specifies parameters for retrieving a SymmetrixConfig object.

The following examples show a request to query for a generic Symmetrix configuration and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
<RequestEx>
<Query>
<SymmGeneralConfigQueryParams symmetrix="1"></SymmGeneralConfigQueryParams>
</Query>
</RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx>
       <SymmetrixConfig cacheSize="2048" cacheSlots="53937"
fastIplTime="1135753981000" ident="Symm4" iplTime="1118652132000"
logicalDisks="35" maxDaWPS="21595" maxDeviceWPS="10160" maxWPS="43191"
mcDate="1078203600000" mcPatchDate="1078203600000" mcPatchLevel="44"
mcVersion="5267" mcVersionNum="1493AA01" model="3300" name="000182600982"
permaCacheSlots="0" physicalDisks="16" powerPathDevices="0"
pwronTime="1135753981000" serialNumber="000182600982"
snapShotAt="1148094877337" symmetrix="1"/>
   </ResponseEx>
</ResponsePacket>
```

Querying Symmetrix director objects

The SymmDirectorQueryParams object specifies parameters for retrieving all aspects of Symmetrix directors. Objects returned are either of type SymmDirectorConfig or SymmDirectorStatus depending on the aspect of the query.

Query filters are:

- An Aspect selection element specifying the aspect needed by the application program
- Director type
- Director number

The following examples show a request to query for a Symmetrix director configuration and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
    <RequestEx>
         <Query>
<SymmDirectorQueryParams symmetrix="1">
     <AspectSelection config="true" status=”true”></AspectSelection>
</SymmDirectorQueryParams>
         </Query>
    </RequestEx>
</RequestPacket>
```

Response:

Retrieve Symmetrix director configuration example

A user can query for any type of director using type filter and/or any specific director using the director number.

Querying Symmetrix physical disks

The SymmPhysicalDiskQueryParams object specifies parameters for retrieving Symmetrix physical disks. Objects returned are of type PhysicalDisk.

The following examples show a request to query for a specific physical disk and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
     <RequestEx>
<Query>
        <SymmPhysicalDiskQueryParams symmetrix="1"><physicalDisk interface="C"
scsiId="1" director="1"></physicalDisk></SymmPhysicalDiskQueryParams>
</Query>
    </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx>
       <PhysicalDisk blockSize="512" diskCapacity="69838" hypers="6"
productId="SX173404LC" productRevision="CHET_73" rpm="10033" symmetrix="1"
usedCapacity="143028400" vendorId="SEAGATE " director="1" interface="C"
scsiId="1"/>
   </ResponseEx>
</ResponsePacket>
```

If a user does not apply a filter, all disks are returned.

Querying Symmetrix device objects

The SymmDeviceQueryParams object specifies parameters for retrieving all aspects of Symmetrix devices. Objects returned are of type SymmDeviceFrontendConfig, SymmDeviceHyperDiskConfig, SymmetrixDeviceConfig, or SymmetrixDeviceStatus, depending on the aspect of the query.

Query filters include:

- An Aspect selection element specifying the aspect needed by the application program
- Device range
- Device configuration type

The following examples show a request to query for a Symmetrix frontend configuration and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
     <RequestEx>
         <Query>
<SymmDeviceQueryParams symmetrix="1" configType="bcv">
      <AspectSelection  DeviceAccessInfo="true"></AspectSelection>
      <deviceRange start="A" size="10"></deviceRange>
</SymmDeviceQueryParams>
         </Query>
      </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx>
       <SymmDeviceFrontendConfig device="0F" symmetrix="1">
           <DeviceAccessInfo director="4" lun="18" port="1" virtualBus="0"/>
           <DeviceAccessInfo director="15" lun="18" port="1" virtualBus="0"/>
           <DeviceAccessInfo director="16" lun="18" port="1" virtualBus="0"/>
           <DeviceAccessInfo director="17" lun="18" port="1" virtualBus="0"/>
           <DeviceAccessInfo director="20" lun="18" port="0" virtualBus="0"/>
           <DeviceAccessInfo director="31" lun="18" port="0" virtualBus="0"/>
           <DeviceAccessInfo director="32" lun="18" port="0" virtualBus="0"/>
           <DeviceAccessInfo director="33" lun="18" port="0" virtualBus="0"/>
       </SymmDeviceFrontendConfig>
       <SymmDeviceFrontendConfig device="10" symmetrix="1">
           <DeviceAccessInfo director="4" lun="19" port="1" virtualBus="0"/>
           <DeviceAccessInfo director="15" lun="19" port="1" virtualBus="0"/>
           <DeviceAccessInfo director="16" lun="19" port="1" virtualBus="0"/>
           <DeviceAccessInfo director="17" lun="19" port="1" virtualBus="0"/>
           <DeviceAccessInfo director="20" lun="19" port="0" virtualBus="0"/>
           <DeviceAccessInfo director="31" lun="19" port="0" virtualBus="0"/>
           <DeviceAccessInfo director="32" lun="19" port="0" virtualBus="0"/>
           <DeviceAccessInfo director="33" lun="19" port="0" virtualBus="0"/>
       </SymmDeviceFrontendConfig>
       <SymmDeviceFrontendConfig device="11" symmetrix="1">
           <DeviceAccessInfo director="4" lun="1A" port="1" virtualBus="0"/>
           <DeviceAccessInfo director="15" lun="1A" port="1" virtualBus="0"/>
           <DeviceAccessInfo director="16" lun="1A" port="1" virtualBus="0"/>
           <DeviceAccessInfo director="17" lun="1A" port="1" virtualBus="0"/>
           <DeviceAccessInfo director="20" lun="1A" port="0" virtualBus="0"/>
           <DeviceAccessInfo director="31" lun="1A" port="0" virtualBus="0"/>
           <DeviceAccessInfo director="32" lun="1A" port="0" virtualBus="0"/>
           <DeviceAccessInfo director="33" lun="1A" port="0" virtualBus="0"/>
       </SymmDeviceFrontendConfig>
       <SymmDeviceFrontendConfig device="12" symmetrix="1">
           <DeviceAccessInfo director="4" lun="1B" port="1" virtualBus="0"/>
           <DeviceAccessInfo director="15" lun="1B" port="1" virtualBus="0"/>
           <DeviceAccessInfo director="16" lun="1B" port="1" virtualBus="0"/>
```

```
            <DeviceAccessInfo director="17" lun="1B" port="1" virtualBus="0"/>
            <DeviceAccessInfo director="20" lun="1B" port="0" virtualBus="0"/>
            <DeviceAccessInfo director="31" lun="1B" port="0" virtualBus="0"/>
            <DeviceAccessInfo director="32" lun="1B" port="0" virtualBus="0"/>
            <DeviceAccessInfo director="33" lun="1B" port="0" virtualBus="0"/>
        </SymmDeviceFrontendConfig>
        <SymmDeviceFrontendConfig device="13" symmetrix="1">
            <DeviceAccessInfo director="4" lun="1C" port="1" virtualBus="0"/>
            <DeviceAccessInfo director="15" lun="1C" port="1" virtualBus="0"/>
            <DeviceAccessInfo director="16" lun="1C" port="1" virtualBus="0"/>
            <DeviceAccessInfo director="17" lun="1C" port="1" virtualBus="0"/>
            <DeviceAccessInfo director="20" lun="1C" port="0" virtualBus="0"/>
            <DeviceAccessInfo director="31" lun="1C" port="0" virtualBus="0"/>
            <DeviceAccessInfo director="32" lun="1C" port="0" virtualBus="0"/>
            <DeviceAccessInfo director="33" lun="1C" port="0" virtualBus="0"/>
        </SymmDeviceFrontendConfig>
    </ResponseEx>
</ResponsePacket>
```

The following examples show a request that queries a hyper disk configuration and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
     <RequestEx>
         <Query>
<SymmDeviceQueryParams symmetrix="1">
     <AspectSelection hyperDisk="true"></AspectSelection>
</SymmDeviceQueryParams>
         </Query>
     </RequestEx>
</RequestPacket>
```

Response:

Retrieve Symmetrix hyper disk configuration example

The following examples show a request that queries device configuration and status and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
     <RequestEx>
          <Query>
    <SymmDeviceQueryParams symmetrix="1">
                <AspectSelection config="true" status="true"></AspectSelection>
           <DeviceRange start="A" size="10"></DeviceRange>
     </SymmDeviceQueryParams>
          </Query>
     </RequestEx>
```

```
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx>
       <SymmetrixDeviceConfig bcvDevices="" blockSize="512" capacity="2077"
config="mirror-2" cylinders="4430" device="0A" symmetrix="1"/>
       <SymmetrixDeviceConfig bcvDevices="" blockSize="512" capacity="2077"
config="mirror-2" cylinders="4430" device="0B" symmetrix="1"/>
       <SymmetrixDeviceConfig bcvDevices="" blockSize="512" capacity="2077"
config="mirror-2" cylinders="4430" device="0C" symmetrix="1"/>
       <SymmetrixDeviceConfig bcvDevices="" blockSize="512" capacity="3"
config="mirror-2" cylinders="7" device="0D" symmetrix="1"/>
       <SymmetrixDeviceConfig bcvDevices="" blockSize="512" capacity="11619"
config="mirror-2" cylinders="24788" device="0E" symmetrix="1"/>
       <SymmetrixDeviceConfig bcvDevices="" blockSize="512" capacity="11619"
config="mirror-2" cylinders="24788" device="0F" symmetrix="1"/>
       <SymmetrixDeviceConfig bcvDevices="" blockSize="512" capacity="2077"
config="mirror-2" cylinders="4430" device="10" symmetrix="1"/>
       <SymmetrixDeviceConfig bcvDevices="" blockSize="512" capacity="2077"
config="mirror-2" cylinders="4430" device="11" symmetrix="1"/>
       <SymmetrixDeviceConfig bcvDevices="" blockSize="512" capacity="2077"
config="mirror-2" cylinders="4430" device="12" symmetrix="1"/>
       <SymmetrixDeviceConfig bcvDevices="" blockSize="512" capacity="2077"
config="mirror-2" cylinders="4430" device="13" symmetrix="1"/>
       <SymmetrixDeviceStatus status="ready" device="0A" symmetrix="1">
           <BcvDevice bcvStatus="invalid"/>
       </SymmetrixDeviceStatus>
       <SymmetrixDeviceStatus status="ready" device="0B" symmetrix="1">
           <BcvDevice bcvStatus="invalid"/>
       </SymmetrixDeviceStatus>
       <SymmetrixDeviceStatus status="ready" device="0C" symmetrix="1">
           <BcvDevice bcvStatus="invalid"/>
       </SymmetrixDeviceStatus>
       <SymmetrixDeviceStatus status="ready" device="0D" symmetrix="1">
           <BcvDevice bcvStatus="invalid"/>
       </SymmetrixDeviceStatus>
       <SymmetrixDeviceStatus status="ready" device="0E" symmetrix="1">
           <BcvDevice bcvStatus="invalid"/>
       </SymmetrixDeviceStatus>
       <SymmetrixDeviceStatus status="write-disabled" device="0F"
symmetrix="1">
           <BcvDevice bcvStatus="invalid"/>
       </SymmetrixDeviceStatus>
       <SymmetrixDeviceStatus status="ready" device="10" symmetrix="1">
           <BcvDevice bcvStatus="invalid"/>
       </SymmetrixDeviceStatus>
       <SymmetrixDeviceStatus status="ready" device="11" symmetrix="1">
           <BcvDevice bcvStatus="invalid"/>
       </SymmetrixDeviceStatus>
       <SymmetrixDeviceStatus status="ready" device="12" symmetrix="1">
           <BcvDevice bcvStatus="invalid"/>
       </SymmetrixDeviceStatus>
       <SymmetrixDeviceStatus status="ready" device="13" symmetrix="1">
           <BcvDevice bcvStatus="invalid"/>
       </SymmetrixDeviceStatus>
   </ResponseEx>
```

```
</ResponsePacket>
```

Statistics management

Statistics are collected in the JServer database. For each set of related metrics, JServer polls the appropriate component of the system using a constant time interval. (The polling interval for movers is set by default to five minutes, but it can be changed using the Celerra Manager. The polling interval of file systems usage is 10 minutes, but samples are saved in the database only once per hour. This cannot be changed.) Each sample is recorded in the database together with the time at which the sample was taken. The time is the Control Station time. Therefore, when the application submits a request for statistics or when it receives back samples, it needs to adjust the time to the time as it is set on the client's machine.

The best way to estimate Control Station time is as follows:

- Submit an empty <RequestPacket> (The packet that does not have any <Request> elements in it).
- Get a <ResponsePacket>. This packet will have an attribute time that will have a value of ControlStation time in milliseconds at the time of the reply.

Since the processing of an empty packet takes minimum time, the above time is the best estimate of the Control Station time. For better precision, the application can adjust this time with the response delay divided by two, for this particular packet.

QueryStats request

QueryStats requests are processed synchronously. The list of available metrics are defined in the files QueryStats.xsd, MoverStats.xsd, and FileSystemStats.xsd. In the current release, metrics are classified as follows:

- MoverStats – an extensive set of mover statistics, including memory and CPU usage, CIFS statistics, NFS statistics, and Network statistics
- VolumeStats – a set of I/O statistics for Celerra volumes per mover
- FileSystemUsage – the usage of file system resources, such as space and inodes

Mover statistics – MoverStats

The MoverStats request has two parameters (attributes): mover ID and the name of the set of statistics required. Allowed names are enumerated as follows:

- ResourceUsage – mover CPU and memory usage
- Network-IP – mover IP statistics counters
- Network-TCP – mover TCP statistics counters
- Network-UDP – mover UDP statistics counters
- Network-Devices – mover network card devices performance counters
- Network-All – all network-related performance counters
- CIFS-SMB-Procs – mover CIFS SMB procedure calls counters
- CIFS-Trans2-Procs – mover CIFS Trans2 procedure calls counters
- CIFS-NT-Procs – mover CIFS NT procedure calls counters
- CIFS-SMB-Time – mover CIFS SMB "time spent in call" counters
- CIFS-Trans2-Time – mover CIFS Trans2 "time spent in call" counters
- CIFS-NT-Time – mover CIFS NT "time spent in call" counters

- CIFS-State – mover number of open connections and files
- CIFS-Totals – mover CIFS total procedure calls counters
- CIFS-All – all CIFS-related performance counters
- NFSV2-Procs – mover NFSV2 procedure calls counters
- NFSV2-Time – mover NFSV2 "time spent in call" counters
- NFSV2-Failures – mover NFSV2 procedure call failures counters
- NFSV3-Procs – mover NFSV3 procedure calls counters
- NFSV3-Time – mover NFSV3 "time spent in call" counters
- NFSV3-Failures – mover NFSV3 procedure call failures counters
- NFS-RPC – mover NFS RPC statistics
- NFS-Lookup-Cache – mover NFS Lookup Cache counters
- NFS-All – all NFS-related counters
- Performance-Summary-Distribution – last week performance summary distribution (for the detailed explanation see the appropriate annotation in the schema)

The following example demonstrates how mover statistics requests are coded:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request clientHandle='abcdef'>
    <QueryStats start="1130337900" interval="7200" step="300">
      <MoverStats statsSet="ResourceUsage" mover="2"/>
    </QueryStats>
  </Request>
</RequestPacket>
```

The <QueryStats> element contains up to three attributes:

- *start* – specifies time in seconds, since Jan. 1, 1970 (UTC time), where the interval from which to fetch the statistics starts.
- *interval* – specifies the length of the interval in seconds.
- *step* – specifies the minimum time between two successive samples in seconds. If, for example, the database contains records taken at 14:00:00, 14:01:01, 14:02:00, 14:03:02, and 14:04:01, uses a step of 300 seconds, and if the record 14:00:00 is in the resulting set, the next record in the set would be taken at 14:04:01. In other words, the resolution is approximately five minutes.

If the *start* attribute is missing, the XML API assumes the application needs the statistics in the time period from now_time-interval_value until now_time.

If the *step* attribute is missing or 0, the XML API assumes the application needs all statistics within the specified interval.

If all attributes are missing, the XML API assumes the application needs the latest sample only.

The <MoverStats> element in the previous example defines the subset of metrics taken from a Data Mover. In this case, the request is for the CPU and memory usage of mover ID 2.

The response will be similar to the following example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket time="1130359569228"
xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Response clientHandle="abcdef">
   <MoverResourceUsage mover="2">
     <Sample cpu="12.0" mem="35.33596" stamp="510" time="1130338028"/>
     <Sample cpu="13.0" mem="36.33596" stamp="510" time="1130338329"/>
```

```
      <Sample cpu="1.0" mem="33.33596" stamp="510" time="1130338630"/>
      <Sample cpu="5.0" mem="34.336174" stamp="520" time="1130338931"/>
      <!— etc... -->
    </MoverResourceUsage>
  </Response>
</ResponsePacket>
```

The attribute time is UTC time in seconds for each sample. A special explanation needs the attribute stamp. The polling process on the JServer can be stopped and started again for multiple reasons. For example:

- The Data Mover may panic and reboot or failover.
- The polling interval may change by the operator request, and in this case, the mover stops and starts polling with a new polling interval.
- Polling could be stopped completely and resumed after a while.

In short, the JServer, after polling has been stopped and restarted, cannot guarantee there are no gaps in the statistics and that the counters did not wrap. The stamp attribute aids the application in discovering potential gaps in the natural flow of statistics. The stamp changes each time the JServer restarts polling certain sets of statistics. The value of the stamp is of no importance. The only significance is the change.

The following examples show a request for mover 2 to retrieve the CIFS Totals profile for the last two hours using a step of 10 minutes and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <QueryStats  interval="7200" step="600">
            <MoverStats statsSet="CIFS-Totals" mover="2"/>
        </QueryStats>
    </Request>
</RequestPacket>
```

Response:

Retrieve the CIFS Totals profile example

The following examples show a request for mover 2 to retrieve the CIFS number of SMB calls profile for the interval starting on Dec. 20, 2005 at 3 A.M., and lasting six hours using a step of 40 minutes and the corresponding response.

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <QueryStats start="1135066743" interval="21600" step="2400">
            <MoverStats statsSet="CIFS-SMB-Procs" mover="2"/>
        </QueryStats>
    </Request>
</RequestPacket>
```

Response:

Retrieve the CIFS number of SMB calls profile example

The following examples show a request for mover 2 to retrieve the amount of time NFSV3 spent in calls and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <QueryStats>
            <MoverStats statsSet="NFSV3-Time" mover="2"/>
        </QueryStats>
    </Request>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Response>
        <MoverNfsStats mover="2">
            <Sample stamp="290" time="1135085453">
                <ProcV3Time v3access="142289620" v3commit="237197796"
v3create="485888056" v3fsinfo="4293880" v3fsstat="765008" v3getattr="408529900"
v3link="481796" v3lookup="422006892" v3mkdir="18941760" v3mknod="541020"
v3null="0" v3pathconf="0" v3read="2234653756" v3readdir="7127480"
v3readdirplus="53264540" v3readlink="5649556" v3remove="380575312"
v3rename="2507160" v3rmdir="13286632" v3setattr="99470420" v3symlink="787764"
v3write="4038508296"/>
            </Sample>
        </MoverNfsStats>
    </Response>
</ResponsePacket>
```

Mover statistics indications

Applications can subscribe for mover statistics and receive indications, depending on the statistics type (such as CIFS, Network, NFS, performance summary, and so on). Then, whenever statistics are available, applications receive indications. Applications can unsubscribe for mover indications.

Also, applications can specify mover as a filter.

The following example shows a request that subscribes for all CIFS mover statistics for all movers.

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
<Request>
       <Subscribe>
               <MoverStats statsSet="CIFS-All"></MoverStats>
       </Subscribe>
```

```
</Request>
</RequestPacket>
```

Volume statistics – VolumeStats

Celerra systems can have a large number of volumes. Typically, any disk volume (and consequently any metavolume) can be reached from any mover; however, for I/O operations on a volume and on a specific mover to occur the file system based on this volume must be mounted on the mover. As a result, not every mover keeps I/O counters for a specific volume. Since the application may be interested in I/O for a specific file system, the VolumeStats request can have the additional parameter, volume ID. Three types of requests are possible:

- Volume – mover I/O statistics for a specific volume
- Totals – mover I/O statistics all volumes, totaled
- All – mover I/O statistics totaled and the totals for all volumes on this mover

The following example makes a request to fetch statistics for the last 30 minutes using a step of five minutes for the volume 1566 from mover 1:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <Request clientHandle='abcdef'>
    <QueryStats interval="1800" step="300">
      <VolumeStats statsSet="Volume" mover="1" volume="1566" />
    </QueryStats>
  </Request>
</RequestPacket>
```

The corresponding response follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <Response clientHandle="abcdef">
     <VolumeStats mover="1" volume="1566">
       <Sample bytesRead="6078906368" bytesWritten="216357052416"
reqsRead="742019" reqsWritten="26412328" stamp="310" time="1135307661"/>
       <Sample bytesRead="6101344256" bytesWritten="219944157184"
reqsRead="744758" reqsWritten="26850210" stamp="310" time="1135307961"/>
       <Sample bytesRead="6129401856" bytesWritten="223510986752"
reqsRead="748183" reqsWritten="27285618" stamp="310" time="1135308261"/>
       <Sample bytesRead="6155845632" bytesWritten="227055214592"
reqsRead="751411" reqsWritten="27718266" stamp="310" time="1135308561"/>
       <Sample bytesRead="6184247296" bytesWritten="230581952512"
reqsRead="754878" reqsWritten="28148775" stamp="310" time="1135308861"/>
       <Sample bytesRead="6225739776" bytesWritten="234079289344"
reqsRead="759944" reqsWritten="28575700" stamp="310" time="1135309161"/>
     </VolumeStats>
   </Response>
</ResponsePacket>
```

However, if you attempt to retrieve I/O statistics for the same volume on the mover 2, you receive an empty response (the associated request is not shown for this example):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <Response clientHandle="abcdef">
     <VolumeStats mover="2" volume="1566"/>
```

```
   </Response>
</ResponsePacket>
```

The following examples show a request for mover 2 to retrieve the last two hours of volume I/O totals using a step of five minutes:

Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request>
        <QueryStats interval="7200" step="300">
            <VolumeStats statsSet="Totals" mover="2"/>
        </QueryStats>
    </Request>
</RequestPacket>
```

Response:

Retrieve the last two hours of volume I/O totals example

Volume indications

Applications can subscribe for volume statistics and receive indications depending on statistics type. Then, whenever statistics are available, applications receive an indication. Applications can unsubscribe for statistics indications.

There are three types of indications for volumes:

- All
- Totals
- Volume

Also, applications can use mover and volume as filters to receive indications for specific volumes or volumes on a specific mover.

The following example shows a request that subscribes for all statistics for all volumes.

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
<Request>
       <Subscribe>
               <VolumeStats statsSet="All"></VolumeStats>
       </Subscribe>
</Request>
</RequestPacket>
```

The following example shows a request that subscribes for total statistics for all volumes.

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
<Request>
       <Subscribe>
              <VolumeStats statsSet="Totals"></VolumeStats>
```

```
       </Subscribe>
</Request>
</RequestPacket>
```

The following example shows a request that subscribes for volume statistics for specific volume.

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
<Request>
       <Subscribe>
              <VolumeStats statsSet="Volume"  mover="2"
volume="34"></VolumeStats>
       </Subscribe>
</Request>
</RequestPacket>
```

File system usage statistics – FileSystemUsage

FileSystemUsage, (the used and total capacity of the file system and the used and total inodes (files) count) is not recorded as often as mover statistics (the interval is one hour), but they are kept for longer periods (about six months). You can retrieve statistics for all file systems or for a specific file system.

The following example requests the latest usage data for all file-systems:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <Request clientHandle='abcdef'>
     <QueryStats>
       <FileSystemUsage/>
     </QueryStats>
   </Request>
</RequestPacket>
```

The response follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <Response clientHandle="abcdef">
     <FileSystemSetUsageStats time="1135080439">
       <Item filesTotal="1322494" filesUsed="21" spaceTotal="10777520"
spaceUsed="608" fileSystem="422"/>
       <Item filesTotal="55102" filesUsed="121" spaceTotal="402584"
spaceUsed="6272" fileSystem="638"/>
       <Item filesTotal="55102" filesUsed="18" spaceTotal="402584"
spaceUsed="608" fileSystem="618"/>
       <Item filesTotal="251902" filesUsed="37647" spaceTotal="2015984"
spaceUsed="149672" fileSystem="635"/>
       <!—etc... -->
     </FileSystemSetUsageStats>
   </Response>
</ResponsePacket>
```

File system usage indications

Applications can subscribe for file system usage statistics and receive indications. Then, whenever statistics are available, applications receive an indication. Applications can unsubscribe for file system usage indications.

Applications can use file system as a filter to receive specific file system usage statistics.

The following example shows a request that subscribes for all file system usage statistics.

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
<Request>
       <Subscribe>
              <FileSystemUsage></FileSystemUsage>
       </Subscribe>
</Request>
</RequestPacket>
```

CLARiiON statistics

CLARiiON device, disk, and storage processors statistics are large, so only the most recent statistics are available by means of a query. Query is provided so applications can subscribe for statistics. Then, whenever statistics are available, applications receive an indication. Applications can unsubscribe for statistics indications.

There are three types of indications for CLARiiON:

- Device statistics
- Disk statistics
- SP statistics

Subscribing for CLARiiON device statistics

Applications can subscribe for all CLARiiON device statistics or specific CLARiiON device statistics.

Query filters include:

- CLARiiON storage ID
- CLARiiON device number

The following example shows a request that subscribes for all device statistics for all available CLARiiONs:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
     <Request>
          <Subscribe>
  <ClarDeviceStats></ClarDeviceStats>
```

```
          </Subscribe>
      </Request>
</RequestPacket>
```

Subscribing for CLARiiON disk statistics

Applications can subscribe for all CLARiiON disk statistics or specific CLARiiON disk statistics.

Query filters include:

- CLARiiON storage ID
- CLARiiON disk name

The following example shows a request that subscribes for all disk statistics for all available CLARiiONs:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
     <Request>
          <Subscribe>
  <ClarDiskStats></ClarDiskStats>
          </Subscribe>
    </Request>
</RequestPacket>
```

Subscribing for CLARiiON SP statistics

Applications can subscribe for all CLARiiON SP statistics or specific CLARiiON SP statistics.

Query filters include:

- CLARiiON storage ID
- CLARiiON storage processor ID

The following example shows a request that subscribes for all storage processor statistics for all available CLARiiONs:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
    <Request>
         <Subscribe>
<ClarSpStats></ClarSpStats>
         </Subscribe>
     </Request>
</RequestPacket>
```

Querying for CLARiiON device statistics

The ClarDeviceStatsQueryParams object specifies parameters for retrieving CLARiiON device statistics. Objects returned are of type ClariionDeviceStats.

Query filters include: CLARiiON device range.

The following examples show a request that queries for CLARiiON device statistics and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
     <RequestEx>
         <QueryStats>
 <ClariionDeviceStats clariion="1">
                   <deviceRange start="0" size="2">
                    </deviceRange>
               </ClariionDeviceStats>
          </QueryStats>
    </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx>
       <ClariionDeviceStats clariion="1">
           <Sample time="1159321990">
               <DevStat deviceName="0000" forcedFlushes="0"
prefetchedBlocks="189568" readBlocks="915590" readCacheHits="45982"
readCacheMisses="0" readReqs="49249" writeBlocks="2607005"
writeCacheHits="188671" writeReqs="393961">
                   <readHistogram h0="13248" h1="17" h2="7320" h3="15440"
h4="2635" h5="38" h6="10544" h7="1" h8="0" h9="0"/>
                   <writeHistogram h0="270167" h1="32194" h2="5235" h3="10309"
h4="67483" h5="38" h6="1051" h7="7402" h8="2" h9="80"/>
               </DevStat>
               <DevStat deviceName="0001" forcedFlushes="0"
prefetchedBlocks="132160" readBlocks="132134" readCacheHits="802"
readCacheMisses="0" readReqs="1082" writeBlocks="465923" writeCacheHits="85"
writeReqs="10466">
                   <readHistogram h0="37" h1="0" h2="0" h3="0" h4="14" h5="4"
h6="0" h7="1027" h8="0" h9="0"/>
                   <writeHistogram h0="2782" h1="4158" h2="0" h3="0" h4="382"
h5="6" h6="0" h7="3079" h8="9" h9="25"/>
               </DevStat>
           </Sample>
       </ClariionDeviceStats>
   </ResponseEx>
</ResponsePacket>
```

Querying for CLARiiON disk statistics

The ClarDiskStatsQueryParams object specifies parameters for retrieving CLARiiON disk statistics. Objects returned are of type ClariionDiskStats.

Query filters include: CLARiiON disk name.

The following examples show a request that queries for CLARiiON disk statistics and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
     <RequestEx>
          <QueryStats>
   <ClariionDiskStats clariion="1"></ClariionDiskStats>
          </QueryStats>
     </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx>
       <ClariionDiskStats clariion="1">
           <Sample time="1159336235">
               <DiskStat diskName="0_0_0" readReqs="4743293"
writeReqs="2965383"/>
               <DiskStat diskName="0_0_1" readReqs="4739103"
writeReqs="2954059"/>
               <DiskStat diskName="0_0_2" readReqs="4765592"
writeReqs="1558773"/>
               <DiskStat diskName="0_0_3" readReqs="4717613"
writeReqs="1485449"/>
               <DiskStat diskName="0_0_4" readReqs="4748425"
writeReqs="1510821"/>
               <DiskStat diskName="0_0_5" readReqs="0" writeReqs="0"/>
               <DiskStat diskName="0_0_6" readReqs="2632587" writeReqs="4"/>
               <DiskStat diskName="0_0_7" readReqs="2632585" writeReqs="4"/>
               <DiskStat diskName="0_0_8" readReqs="2632589" writeReqs="4"/>
               <DiskStat diskName="0_0_9" readReqs="2632585" writeReqs="4"/>
               <DiskStat diskName="0_0_10" readReqs="2632585" writeReqs="4"/>
               <DiskStat diskName="0_0_11" readReqs="2" writeReqs="2"/>
               <DiskStat diskName="0_0_12" readReqs="2" writeReqs="2"/>
               <DiskStat diskName="0_0_13" readReqs="2" writeReqs="2"/>
           </Sample>
       </ClariionDiskStats>
   </ResponseEx>
</ResponsePacket>
```

Querying for CLARiiON storage processor statistics

The ClarSPStatsQueryParams object specifies parameters for retrieving CLARiiON disk statistics. Objects returned are of type ClariionSpStats.

Query filters include: CLARiiON storage processor ID.

The following examples show a request that queries for CLARiiON storage processor statistics and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
     <RequestEx>
          <QueryStats>
   <ClariionSpStats spId="A" clariion="1"></ClariionSpStats>
          </QueryStats>
     </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx>
       <ClariionSpStats clariion="1">
           <Sample time="1157944559">
               <SPStat blocksRead="1528855" blocksWritten="142945706"
dirtyPages="1" id="A" queuedArrivals="32624444" readReqs="36185"
writeReqs="7836086"/>
           </Sample>
       </ClariionSpStats>
   </ResponseEx>
</ResponsePacket>
```

Symmetrix statistics

Symmetrix device statistics are large, so only the most recent statistics are available by means of a query. Query is provided so applications can subscribe for statistics. Then, whenever statistics are available, applications receive an indication. Applications can unsubscribe for statistics indications.

Applications can subscribe for SymmFrontEndStats, SymmBackEndStats, SymmPortStats, or they can subscribe for all of these statistics. They are all aspects of the Symmetrix director subscription.

Query filters include:

- An Aspect selection element specifying the aspect needed by the application program
- Symmetrix storage ID
- Symmetrix director number

The following example shows a request that subscribes for all types of statistics. Applications can also subscribe for a single set of director statistics for a specific Symmetrix storage system.

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
    <Request clientHandle="abc">
       <Subscribe>
            <SymmDirectorStats statsSet="all"></SymmDirectorStats>
       </Subscribe>
    </Request>
```

```
</RequestPacket>
```

Querying Symmetrix director statistics

The SymmDirectorStatsQueryParams object specifies parameters for retrieving all aspects of a Symmetrix director. Objects returned are of type SymmFrontEndStats, SymmBackEndStats, or SymmPortStats, depending on the aspect of the query.

Query filters include:

- An Aspect selection element specifying the aspect needed by the application program
- Director number

If the director number is not specified, the total frontend or backend statistics are returned depending on the aspect.

The following examples show a request that queries total frontend statistics and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
     <RequestEx>
         <QueryStats>
<SymmDirectorStats symmetrix="1">
     <AspectSelection frontStats="true"></AspectSelection>
</SymmDirectorStats>
         </QueryStats>
     </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx>
       <SymmFrontEndStats symmetrix="1">
           <Sample time="1157939999">
               <FrontEnd deviceWPDisconnects="107976" hits="1008128041"
ios="1061913627" permaCacheRequests="0" readMisses="4976791"
readRequests="440111124" slotCollisions="615903" systemWPDisconnects="1"
totalRequests="1014574187" writeRequests="574463063"/>
           </Sample>
       </SymmFrontEndStats>
   </ResponseEx>
</ResponsePacket>
```

The following examples show a request that queries backend statistics and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
     <RequestEx>
          <QueryStats>
  <SymmDirectorStats symmetrix="1" director="1">
       <AspectSelection backStats="true"></AspectSelection>
  </SymmDirectorStats>
          </QueryStats>
     </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx>
       <SymmBackEndStats symmetrix="1">
           <Sample time="1157939896">
               <BackEnd director="1" ios="55575912" prefetchLongMiss="9091"
prefetchMismatches="0" prefetchRestarts="17478" prefetchShortMiss="11129"
prefetchTasks="0" prefetchTracksNotUsed="104159" prefetchTracksUsed="860057"
readRequests="979438" totalRequests="59062707" writeRequests="58083269"/>
           </Sample>
       </SymmBackEndStats>
   </ResponseEx>
</ResponsePacket>
```

The following examples show a request that queries port statistics and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
 apiVersion="V1_1">
    <RequestEx>
         <QueryStats>
  <SymmDirectorStats symmetrix="1">
       <AspectSelection portStats="true"></AspectSelection>
  </SymmDirectorStats>
         </QueryStats>
    </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx>
       <SymmPortStats symmetrix="1">
           <Sample time="1157940059">
               <DirectorPortStats director="3">
                   <PortStats blocks="0" ios="0" port="1"/>
```

```
                </DirectorPortStats>
                <DirectorPortStats director="4">
                    <PortStats blocks="386061907" ios="224151556" port="1"/>
                </DirectorPortStats>
                <DirectorPortStats director="13">
                    <PortStats blocks="0" ios="0" port="1"/>
                </DirectorPortStats>
                <DirectorPortStats director="14">
                    <PortStats blocks="1156460713" ios="733494950" port="1"/>
                </DirectorPortStats>
                <DirectorPortStats director="19">
                    <PortStats blocks="1319387977" ios="61081391" port="1"/>
                </DirectorPortStats>
                <DirectorPortStats director="20">
                    <PortStats blocks="0" ios="0" port="1"/>
                </DirectorPortStats>
                <DirectorPortStats director="29">
                    <PortStats blocks="846688911" ios="43195189" port="1"/>
                </DirectorPortStats>
                <DirectorPortStats director="30">
                    <PortStats blocks="0" ios="0" port="1"/>
                </DirectorPortStats>
            </Sample>
        </SymmPortStats>
    </ResponseEx>
</ResponsePacket>
```

Querying Symmetrix device statistics

The SymmDeviceStatsQueryParams object specifies parameters for retrieving Symmetrix devices for a specified range. Objects returned are of type SymmDeviceStats.

Query filters include: mandatory device range.

The following examples show a request that queries device statistics and the corresponding response:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
apiVersion="V1_1">
     <RequestEx>
 <QueryStats>
             <SymmDeviceStats symmetrix="1">
<DeviceRange start="0000" size="200"></DeviceRange>
      </SymmDeviceStats>
</QueryStats>
     </RequestEx>
</RequestPacket>
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <ResponseEx>
```

```
       <SymmDevicesStats symmetrix="1">
           <Sample time="1157940209">
               <DeviceStat deferredWrites="2142293" delayedDeferredWrites="0"
destagedTracks="675354" formatPendingTracks="0" hits="3192654"
prefetchedTracks="105022" readHits="1050962" readRequests="11331988480"
sequentialReadRequests="37052" totalRequests="44733292544" writeHits="2141692"
writePendingLimit="8248" writePendingTracks="1" writeRequests="33401304064"
device="0"/>
               <DeviceStat deferredWrites="2142293" delayedDeferredWrites="0"
destagedTracks="675354" formatPendingTracks="0" hits="3192654"
prefetchedTracks="105022" readHits="1050962" readRequests="11331988480"
sequentialReadRequests="37052" totalRequests="44733292544" writeHits="2141692"
writePendingLimit="8248" writePendingTracks="1" writeRequests="33401304064"
device="1"/>
               .
               .
               .
               <DeviceStat deferredWrites="9659" delayedDeferredWrites="0"
destagedTracks="12004" formatPendingTracks="0" hits="10415"
prefetchedTracks="69" readHits="756" readRequests="5547520"
sequentialReadRequests="69" totalRequests="242215936" writeHits="9659"
writePendingLimit="8216" writePendingTracks="0" writeRequests="236668416"
device="c6"/>
               <DeviceStat deferredWrites="9659" delayedDeferredWrites="0"
destagedTracks="12004" formatPendingTracks="0" hits="10415"
prefetchedTracks="69" readHits="756" readRequests="5547520"
sequentialReadRequests="69" totalRequests="242215936" writeHits="9659"
writePendingLimit="8216" writePendingTracks="0" writeRequests="236668416"
device="c7"/>
           </Sample>
       </SymmDevicesStats>
   </ResponseEx>
</ResponsePacket
```

API versioning and client application migration strategy

As Celerra software progresses from release to release, the XML API interfaces change as well. EMC recognizes that client applications will not be able to convert to newer versions of the XML API immediately. In addition, if an application manages multiple Celerra systems, where some of the systems remain at the older versions and while some have migrated to newer versions of the software, the application may have difficulties dealing with multiple versions of the protocol. Typically, the application can use only one version of the schema or it should use non-validating XML parsing, which causes difficulties for application developers.

To simplify development, the XML API supports protocol versioning and can reply to clients with the XML objects that comply with the schema version required by the client application. EMC does not commit to support all API versions back to version 1_0, but it will support one or two previous releases of the API.

The supported versions of XML API are enumerated in the type APIVersion defined in the schema file BasicTypes.xsd. Currently, the only supported version is initial version V1_0.

An application declares the API version it understands as the attribute apiVersion of the <RequestPacket> element. If this attribute is not specified, the API uses a value specified in a

previous request packet. If it was never specified, the lowest supported version is assumed. For example:

```
<RequestPacket apiVersion="V1_2">
......
<Request>
......
</RequestPacket>
```

In the previous example, starting from when the packet is received, the XML API Server starts replying to the client using version 1_2. (Note that this example uses version 1_2, which is not currently supported).

The strategy of migrating schema objects will be on a case-by-case basis. The schema can change in many ways from release to release. Consider, for example, changes in enumerations:

- Values that are new in the new version of the schema are reported as "other" in the output that must be compatible with older version of the schema.
- Values that become obsolete with the new version of the schema are not be deleted, but instead, marked with the global meta attribute `meta:obsolete="true"` in the new schema.

For complex types, considerations are more complicated. Some, but definitely not all, cases include:

- Deleting an attribute or an element that was optional in the old version is handled by ignoring this attribute/element (Note that if the absence of the attribute in the old schema meant something and the application used this fact, this may cause undesirable effects).
- Deleting an attribute or an element required under the old schema introduces a new, similar complex type in the newer version of the schema. For example, if a Volume object lost an attribute, then in the new version of the schema there will be a new object called Volume1_1. When generating an object of the type Volume, this property will probably be faked.
- Adding an optional attribute or an element does not introduce a new type. Under the older version of the schema, this attribute/element is not reported, and under the newer version it is reported.
- Adding a required attribute or an element introduces a new complex type.
- Changing the type and occurrence constraints of an attribute or an element is dealt with on a case-by-case basis. Some cases are backwards-compatible and some are not.
- A new complex type is not visible under the old schema.
- There are cases when adding a new attribute or an element to a complex type, or even adding a new value to an attribute or an element may case the object of this type to totally disappear in the old schema. For example, suppose in the new version there is a volume of a type that has type-dependent information that is completely orthogonal to anything known in the previous version. It, therefore, cannot be reduced to some complex type of the previous version and is not reported when queried by the older version of the software.

XML API fault and error status handling

A client application can experience various fault and error conditions while running. These conditions can be classified as follows:

- Packets faults
- Request faults
- Embedded status elements

Packet faults

Packet faults are generated when the XML API Server cannot process the entire user packet. The most frequent cause occurs when the user request packet is invalid and, therefore, cannot be parsed and understood. The following example contains invalid XML:

```
<?xml version="1.0"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
 <Request>
     <Query>
         <!—Below, the closing angle bracket is missing -->
         <StoragePoolQueryParams
     </Query>
   </Request>
</RequestPacket>
```

The following example shows the resulting response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <PacketFault maxSeverity="error">
      <Problem component="API" facility="Prevalidator" message="User request is
not compliant with XML API schema" messageCode="14227341329" severity="error">
          <Diagnostics>
            Exception tag: 10974d61702
            Exception message: Attribute names must not start with "&lt;"
characters.
          </Diagnostics>
      </Problem>
   </PacketFault>
</ResponsePacket>
```

XML API software bugs are another cause of packet faults. The XML API Server is programmed to handle a failure to perform a valid user request and not result in the total failure of the XML API Server. Instead, only the thread running the request fails. This, in turn, may result in a response similar to the previous example, but with different diagnostics:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <PacketFault maxSeverity="error">
      <Problem component="API" facility="Generic" message="User reply
marshaling error" messageCode="14227144717" severity="error">
          <Diagnostics>
            Exception tag: 10978036408
            Exception message: Null pointer exception.
          </Diagnostics>
      </Problem>
   </PacketFault>
</ResponsePacket>
```

Receiving this type of fault does not mean the XML API Server is totally broken. You need to report the message to an EMC Customer Support Representative and supply the XML API Server log (`/nas/log/cel_api.log*`), but, in general, the application can recover and perform other requests against the server.

Request faults

It is possible that the XML API Server, during the execution of the packet, fails to perform one of the requests. This type of failure results in a fault of this request, but not of the entire packet. Note that after such a failure, the other requests in the same packet are performed as if nothing happened. In the following example, there are two requests: one to query all volumes and another to subscribe for volume statistics:

```
<?xml version="1.0"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
    <Request clientHandle='request_1'>
        <Query>
            <VolumeQueryParams/>
        </Query>
    </Request>
    <Request clientHandle='request_2'>
        <Subscribe>
            <VolumeStats statsSet='Totals'/>
        </Subscribe>
    </Request>
</RequestPacket>
```

In the response, the first request did not execute properly because the APL server was not functioning at the moment, but the second request did not involve APL and it is executed:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <Response clientHandle="request_1">
      <Fault maxSeverity="error">
          <Problem component="API" facility="Generic" message="Unexpected
exception of unknown nature" messageCode="14227144705" severity="error">
             <Diagnostics>
               Exception tag: 10978376d8d
               Exception message: No reply from APL
             </Diagnostics>
          </Problem>
       </Fault>
     </Response>
   <Response clientHandle="request_2"/>
</ResponsePacket>
```

In any case, you need to report failures of this type to an EMC Customer Support Representative since they may indicate additional problems.

Embedded Status elements

A query response, a task response, or an indication can return a <Status> element. This section paraphrases material in the sections about Query and Task. A <Status> element typically appears as shown in the following query response example:

```
<QueryStatus maxSeverity="ok"/>
```

This means there were no problems during the execution. However, in some cases the status may contain (possibly multiple) <Problem> elements:

```
  <QueryStatus maxSeverity=" warning ">
      <Problem component="APL" message="Quota information cannot be obtained
because file system poolfs is not read-write mounted."
messageCode="13691191314" severity="warning"/>
      <Problem component="APL" message="Quota information cannot be obtained
because file system RAW_S08_IPK041 is not read-write mounted."
messageCode="13691191314" severity=" warning "/>
      <Problem component="APL" message="Quota information cannot be obtained
because file system test is not read-write mounted." messageCode="13691191314"
severity=" warning "/>
      <Problem component="APL" message="Quota information cannot be obtained
because file system test_fs1 is not read-write mounted."
messageCode="13691191314" severity=" warning "/>
      <Problem component="APL" message="Quota information cannot be obtained
because file system s1_fs1 is not read-write mounted."
messageCode="13691191314" severity=" warning "/>
  </QueryStatus>
```

A task response appears as follows:

```
<Response>
    <TaskResponse taskId="882">
        <Status maxSeverity="warning">
            <Problem component="APL" message="Service already enabled on
server_2." messageCode="17986748529" severity="warning"/>
        </Status>
    </TaskResponse>
</Response>
```

A severity level of warning usually means nothing is wrong; however, the underlying software discovered some conditions that the application program may want to report to the operator of the program. Queries that are typically valid, but incomplete under present conditions, with few exceptions return only warnings. For example, if a mover is rebooting and can not be queried at the moment, all objects existing in this mover's space cannot be queried and a wildcard query of all movers returns a warning status for this particular mover. Some warnings can indicate real problems. For example, if you try to create a CIFS server and make it join the domain in the same request, there are lots of opportunities for this very complex request to experience problems. In this example, the CIFS server has been created but was not able to join the domain:

```
<TaskResponse taskId="12">
    <Status maxSeverity="warning">
        <Problem component="APL" message="Join failed. System was unable to
join the CIFS server to the domain." messageCode="17986748527"
severity="warning"/>
        <Problem component="APL" message="You may need to enable the CIFS
service if it is not running." messageCode="26576683104" severity="info"/>
    </Status>
</TaskResponse>
```

In all cases the attribute *maxSeverity* of the <Status> element is the worst possible severity among severities of all <Problem> elements.

A severity level of error typically indicates a real problem with either the user application or the server software. The system, in most cases, has no intelligence to differentiate between these two classes of problems. In the following example, it looks like a transient server problem:

```
<TaskResponse taskId="908">
    <Status maxSeverity="error">
        <Problem component="APL" message="Resources currently unavailable."
messageCode="13690601491" severity="error"/>
    </Status>
</TaskResponse>
```

Configuring and starting the XML API Server on the Control Station

Both XML API Servlet and XML API Server share a set of configuration parameters. All parameters are located in the properties file, `$NAS_DB/sys/xml_api.conf`, which typically resolves to `/nas/sys/xml_api.conf`. Client application developers can change some of these parameters. After changing parameters that affect the servlet, the Tomcat server on the Control Station needs to be restarted. If the parameters affect the XML API Server, it needs to be restarted as well. Among the list of properties, there are some debug flags. EMC assumes the current release is intended for the application development and not deployment, so many of the flags are set to true (debug mode).

Properties of interest and a short explanation follow:

- `xml.api.server.log` – the location of the XML API Server log relative to the `$NAS_DB` (usually set to `/nas`) directory. Currently the value is set to `log/cel_api.log`, which normally results in the log file being recorded in the `/nas/log/cel_api.log` file. The value of this property affects XML API Server.
- `xml.api.servlet.log` – the location of the XML API Servlet log relative to the `$NAS_DB` directory. Currently the value is set to `log/webui/cel_api.log`, which normally results in the log file being recorded in the `/nas/log/webui/cel_api.log` file. The value of this property affects Tomcat server.
- `xml.api.servlet.logmask` – switches on and off certain servlet log profiles. The default value for the mask is zero, which means that the servlet does not log anything. You should not change this value, unless you do it temporarily at the request from EMC Support engineers and then reset to zero upon completion of the tests. The value of this property affects Tomcat server.
- `xml.api.user.request.validation.flag` – if true, performs a full user request validation before parsing. In the case of improperly formatted request packets, it returns more meaningful diagnostic messages to the user. This property is set to true in the current release, but EMC suggests you set it to false when the application is deployed since it adds to CPU and memory overhead and slightly increases the response time. The value of this property affects XML API Server.
- `xml.api.enable.indications.ext` – if true, user applications can receive indications for configuration changes. Indications on task completions and statistics are always delivered (regardless of the value of the flag). This property is set to true in the current release and in the future it will be eliminated and the above indications will always be delivered (when the client application subscribes to them). The value of this property affects XML API Server.
- `xml.api.trace.apl.calls` – logs APL requests and responses in the XML API server log (property `xml.api.server.log`). This property is set to true, however, at the time of application deployment, it should be set to false. The value of this property affects XML API Server.
- `xml.api.trace.apl.indications` – logs APL indications in the XML API server log (property `xml.api.server.log`). This property is set to true, however, at the time of application deployment, it should be set to false. The value of this property affects XML API Server.
- `xml.api.trace.user.requests` – records user application requests in the XML API server log (property `xml.api.server.log`). This property is set to true, however, at the time of application deployment it should be set to false. The value of this property affects XML API Server.
- `xml.api.quota.poll.offset` – affects the exact time of the poll. By default, the tree quota cache is populated once a day. The time is specified in minutes, starting at midnight (Control Station local time) when the poll starts. The current value is 120, which means that the poll starts at

2 A.M. every night. The value -240, for example, defines that the poll starts at 8 P.M. The value of this property affects XML API Server.

Starting the XML API Server

By default the XML API is disabled. To start the XML API server, do the following:

As `root`, use a text editor to uncomment the following entry in `/nas/sys/nas_mcd.cfg`:

```
daemon "XML API Server"
    executable "/nas/sbin/start_xml_api_server"
    optional yes
    canexit yes
    autorestart yes
    ioaccess no
```

Restart nas services with the following command:

```
# service nas start
```

The XML API is now started and is controlled by the master control daemon.

Print book

Most information contained in this online book is available via an Adobe PDF file that you can open from here and print the information. Please note, this book is intended for online use. Not all features are available in the printed version. For example, links to XML examples of each request are not included in the printed version.

Open Adobe PDF

In order to view a .pdf, you must have Adobe's Acrobat Reader installed. You can download Adobe's Acrobat Reader free from the following website:

http://www.adobe.com/products/acrobat/readstep.html

Appendix: Session level protocol

Basically, the protocol operates over more than one TCP connection: Some are used for request/reply messaging, and others are used to carry indications.

To send a request and receive a response, an application obtains a TCP connection and uses HTTP POST. That is, the request is sent in the HTTP POST request body and the response is received in HTTP POST response body. Since there can be multiple request/response transactions occurring simultaneously, such connections can coexist in time, although eventually each is torn down.

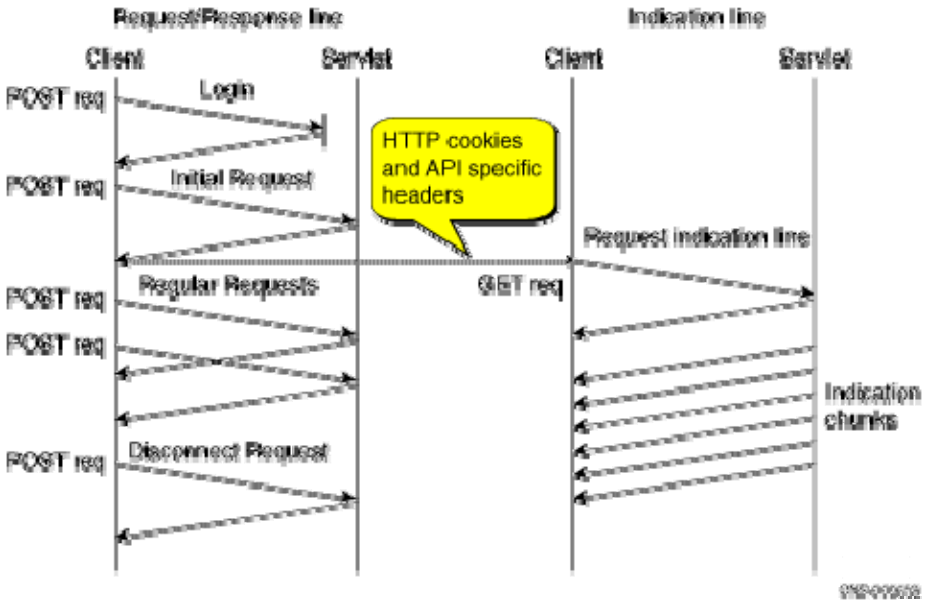
To receive indications, the client obtains one connection and starts a HTTP GET request to the server. The server returns an HTTP1.1 chunked stream of data. Each chunk of the stream corresponds to an indication sent from the server to the client application. The GET request never ends, so effectively it is a permanent TCP connection. This connection is not torn down unless the application exits or the connection is broken.

Therefore, two logical streams coexist: the sequence of exchanges of requests and responses and the sequence of indications. They are called Request/Response line and Indication line.

Initially a client logs in using an HTTP POST request over the Request/Response line. This request is processed by the login script of the HTTP server. The XML API servlet does not participate in the login.

The client then sends an initial request to the servlet using the ticket it received during the login exchange. Any API request can be an initial request. It can be an empty request packet as well. During the processing of this request by the servlet and the API server, an API session is established. The servlet replies with session specific cookies and HTTP headers.

The client can now start an Indication line by sending a GET request, and/or continue sending requests and receiving responses over the Request/Response line. The client can send a Disconnect request to promptly deallocate the session and drop the Indication line. Otherwise, if the client drops the Indication line, the session eventually times out. The following graphic illustrates the Request/Response line and the Indication line.



Request/Response line: initialization

The application first performs the HTTP login and receives the authentication cookie. Login is a totally separate interaction and it goes only as far as HTTP server. This is illustrated in the previous diagram. For login, the application sends a message similar to the following:

```
POST http://172.24.173.31/Login HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: 172.24.173.31
Content-Length: 43

user=nasadmin&password=nasadmin&Login=Login
```

The server responds with a message similar to the following:

```
HTTP/1.1 200 OK
Date: Wed, 21 Sep 2005 13:36:35 GMT
Server: Apache/1.3.27 (Unix) (Red-Hat/Linux) mod_jk/1.2.5 mod_ssl/2.8.12
OpenSSL/0.9.6b mod_perl/1.24_01
Set-Cookie:
Ticket=ip&10.13.3.28&idle&0&persists&&last&1127309795&expires&480&hash&33163d07
49ba813fc7be364c2445b2d0&user&nasadmin&type&User&time&1127309795; path=/
```

The application extracts the cookie for all subsequent calls (including HTTP GET calls). For this example, the initial exchange with the servlet uses an empty request packet:

```
POST /servlets/CelerraManagementServices HTTP/1.1
Host: 172.24.173.31:443
Content-Type: text/xml
Content-Length: 122
Cookie:
Ticket=ip&10.13.3.28&idle&0&persists&&last&1127309795&expires&480&hash&33163d07
49ba813fc7be364c2445b2d0&user&nasadmin&type&User&time&1127309795; $Path=/

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

```
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"/>
```

The response to this request appears as follows:

```
HTTP/1.1 200 OK
Date: Wed, 21 Sep 2005 13:36:36 GMT
Server: Apache/1.3.27 (Unix) (Red-Hat/Linux) mod_jk/1.2.5 mod_ssl/2.8.12
OpenSSL/0.9.6b mod_perl/1.24_01
Set-Cookie: JSESSIONID=00FBDEE3626FF75A8FB6D683FFF4B5E5; Path=/; Secure
CelerraConnector-Sess: 00FBDEE3626FF75A8FB6D683FFF4B5E5
Content-Length: 147
Content-Type: text/xml;charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket
time="1127310027520" xmlns="http://www.emc.com/schemas/celerra/xml_api"/>
```

The response to an empty packet is an empty packet. This is not essential. HTTP headers, however, are important because they contain:

- A JSESSIONID cookie used by Tomcat to identify the session
- A CelerraConnector-Sess header used to identify the API session to the API server

The API uses the first Tomcat session ID as the API session ID. Since Tomcat may deallocate its session, eventually the API session ID can differ from the Tomcat session ID. So, in essence, the API session ID is Tomcat's ID.

Request/Response line: regular transactions

On a regular request, the client application sends both cookies and the HTTP header `CelerraConnector-Sess`, which it received during initial request processing. The following request is an example of a subscription for some indications.

```
POST /servlets/CelerraManagementServices HTTP/1.1
Host: 172.24.173.31:443
Content-Type: text/xml
Content-Length: 681
Cookie:
Ticket=ip&10.13.3.28&idle&0&persists&&last&1127309795&expires&480&hash&33163d07
49ba813fc7be364c2445b2d0&user&nasadmin&type&User&time&1127309795; $Path=/
Cookie: JSESSIONID=00FBDEE3626FF75A8FB6D683FFF4B5E5; $Path=/; $Secure
CelerraConnector-Sess: 00FBDEE3626FF75A8FB6D683FFF4B5E5

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RequestPacket xmlns="http://www.emc.com/schemas/celerra/xml_api"
   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'>
<Request clientHandle='a'>
  <Subscribe>
    <MoverStats statsSet='NFS-RPC' mover="2"/>
    <MoverStats statsSet='CIFS-Totals' mover="2"/>
    <MoverStats statsSet='Network-TCP' mover="2"/>
    <MoverStats statsSet='ResourceUsage' mover="2"/>
    <VolumeStats statsSet='Totals' mover="2"/>
    <FileSystemUsage fileSystem="24" />
  </Subscribe>
```

```
</Request>
</RequestPacket>
```

Here is the response:

```
HTTP/1.1 200 OK
Date: Wed, 21 Sep 2005 13:37:29 GMT
Server: Apache/1.3.27 (Unix) (Red-Hat/Linux) mod_jk/1.2.5 mod_ssl/2.8.12
OpenSSL/0.9.6b mod_perl/1.24_01
CelerraConnector-Sess: 00FBDEE3626FF75A8FB6D683FFF4B5E5
Content-Length: 176
Content-Type: text/xml;charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponsePacket xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <Response clientHandle="a"/>
</ResponsePacket>
```

Indication line: initialization

The client application starts an HTTP GET request in which it sets the previously received cookies and CelerraConnector-Sess HTTP header. To recover from TCP failures or from a firewall abruptly canceling a connection in the middle of session, all indications are sequenced. When the application sends the HTTP GET request, it sends it with the request protocol-specific header: `Sequence-Number`. If this is a new session, its value must be equal to "-1". Otherwise, it is the sequence number of the last indication the application processed. Sequence numbers produced by the servlet start from 0 and increment by 1 each time a new chunk is received. Note, that these sequence numbers have nothing to do with API indication sequence numbers and are used internally by the protocol only.

```
GET /servlets/CelerraManagementServices HTTP/1.1
Host: 172.24.173.31:443
Cookie:
Ticket=ip&10.13.3.28&idle&0&persists&&last&1127309795&expires&480&hash&33163d07
49ba813fc7be364c2445b2d0&user&nasadmin&type&User&time&1127309795; $Path=/
Cookie: JSESSIONID=00FBDEE3626FF75A8FB6D683FFF4B5E5; $Path=/; $Secure
Sequence-Number: -1
CelerraConnector-Sess: 00FBDEE3626FF75A8FB6D683FFF4B5E5
```

The servlet starts the reply by sending the HTTP headers. Immediately after sending the GET request, the application receives HTTP GET response as shown in the following example. The protocol-specific header is `CelerraConnector-RC`. Its value `ACCEPT` says that the servlet accepted the indication connection. If the servlet does not accept the connection, it returns the value `REJECT`. A `REJECT occurs for the following reasons`:

- The session does not exist.
- There is a lack of resources (e.g., too many sessions – maximum number of sessions is set currently to 16).
- The session is unrecoverable (refer to Indication line: recovery).

```
HTTP/1.1 200 OK
Date: Wed, 21 Sep 2005 13:36:37 GMT
Server: Apache/1.3.27 (Unix) (Red-Hat/Linux) mod_jk/1.2.5 mod_ssl/2.8.12
OpenSSL/0.9.6b mod_perl/1.24_01
Transfer-Encoding: chunked
```

```
CelerraConnector-RC: ACCEPT
X-Pad: avoid browser bug
```

Next, the application starts receiving chunks of stream data, where each chunk is a separate indication.

Indication line: indication stream

The first seven lines of an indication stream consist of seven, separate zero-length chunks. These are also known as "keep-hot" messages. They contain no data, but the servlet issues them at least once every 30 seconds to indicate that the connection is alive.

```
0
0
0
0
0
0
0
```

Next, the first real indication arrives. It results from the request to subscribe to certain indications shown in the section Request/Response line: regular transactions. According to the HTTP1.1 standard, the chunk is prefixed by the length in hexadecimal of the incoming chunk. The chunk also contains an application-specific sequence number data: `Sequence-Number`. Sequence numbers increment by one with each incoming chunk. They are used for error recovery.

```
1a6;Sequence-Number=0
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<IndicationPacket sequenceNumber="0" time="1127310038"
xmlns="http://www.emc.com/schemas/celerra/xml_api">
  <StatsIndication>
    <MoverNfs mover="2">
      <Sample time="1127310037">
        <Rpc badAuth="0" badData="0" calls="143861" dupl="0" resends="0"/>
      </Sample>
    </MoverNfs>
  </StatsIndication>
</IndicationPacket>
```

An example of a second indication follows. Note that the sequence number has been incremented.

```
1c0;Sequence-Number=1
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<IndicationPacket sequenceNumber="1" time="1127310038"
xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <StatsIndication>
     <MoverNet mover="2">
       <Sample time="1127310037">
         <Tcp connLing="20" connReq="1" received="60927913" resets="0"
retransm="0" sent="36028041"/>
       </Sample>
     </MoverNet>
   </StatsIndication>
</IndicationPacket>
```

An example of a third indication follows. Additional indications would follow in a similar manner.

```
16c;Sequence-Number=2
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<IndicationPacket sequenceNumber="2" time="1127310039"
xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <StatsIndication>
     <MoverResourceUsage mover="2">
       <Sample cpu="1.0" mem="33.345177" time="1127310037"/>
     </MoverResourceUsage>
   </StatsIndication>
</IndicationPacket>
```

When there are no further indications, keep-hot messages will arrive about every 30 seconds.

Indication line: recovery

The indication connection can be disrupted for various reasons. The API protocol can recover from short disruptions. Longer disruptions (about 90 seconds) cause the servlet to deallocate the session, and the client application will have to create a new session (with all the consequences for its caches). Also, when the servlet pushes a large amount of data to the client, a slow recovery may cause the buffer holding messages for the client to overflow. In this case, an attempt to recover is rejected. The current limit for this type of buffer is 150K.

The following examples illustrate the recovery process. Assume the following example is the last indication received:

```
1c0;Sequence-Number=5
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<IndicationPacket sequenceNumber="6" time="1127397231"
xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <StatsIndication>
     <MoverNet mover="2">
        <Sample time="1127397229">
          <Tcp connLing="20" connReq="1" received="88181961" resets="0"
retransm="3" sent="52196649"/>
        </Sample>
     </MoverNet>
   </StatsIndication>
</IndicationPacket>
```

At this moment, the indication socket is reset. The last received indication packet had sequence number 5. Now the application starts another HTTP GET. This GET differs from the initial GET, because the client sends HTTP header SequenceNumber = 5. This is the last sequence number the client processed.

```
GET /servlets/CelerraManagementServices HTTP/1.1
Host: 172.24.173.31:443
Cookie:
Ticket=ip&10.13.3.71&idle&0&persists&&last&1127397070&expires&480&hash&ed81924d
ecef22181aca5e10eda120f7&user&nasadmin&type&User&time&1127397070; $Path=/
Cookie: JSESSIONID=1AC4C1A009A0A5267534CD9AD58C034F; $Path=/; $Secure
Sequence-Number: 5
CelerraConnector-Sess: 1AC4C1A009A0A5267534CD9AD58C034F
```

The servlet finds that it can recover this session and replies with `ACCEPT`. After that, it continues to send indications in the recovered order starting from indication number 6. If the servlet cannot find an indication with the appropriate sequence number, the client receives a `REJECT` and the `GET` request terminates.

```
HTTP/1.1 200 OK
Date: Thu, 22 Sep 2005 13:53:51 GMT
Server: Apache/1.3.27 (Unix) (Red-Hat/Linux) mod_jk/1.2.5 mod_ssl/2.8.12
OpenSSL/0.9.6b mod_perl/1.24_01
Transfer-Encoding: chunked
CelerraConnector-RC: ACCEPT
Content-Type: text/xml;charset=UTF-8
X-Pad: avoid browser bug
0
0
```

It is normal when the first two keep-hot chunks arrive immediately after the HTTP headers, but after that, the servlet immediately sends the two indications that accumulated in the servlet internal buffer.

```
16c;Sequence-Number=6
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<IndicationPacket sequenceNumber="7" time="1127397232"
xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <StatsIndication>
     <MoverResourceUsage mover="2">
       <Sample cpu="1.0" mem="33.349327" time="1127397229"/>
     </MoverResourceUsage>
   </StatsIndication>
</IndicationPacket>
1c9;Sequence-Number=7
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<IndicationPacket sequenceNumber="8" time="1127397232"
xmlns="http://www.emc.com/schemas/celerra/xml_api">
   <StatsIndication>
     <VolumeSet mover="2">
       <Sample time="1127397229">
         <Totals bytesRead="202407424" bytesWritten="120713632256"
reqsRead="12469" reqsWritten="13176535"/>
       </Sample>
     </VolumeSet>
   </StatsIndication>
</IndicationPacket>


...
```

Prevention of server timeouts

Both Apache HTTP server and Tomcat have built in timeouts. The Apache server invalidates the client ticket in about eight hours. Tomcat invalidates the session in about 20 minutes. To avoid these undesirable events, the client application needs to at least send an empty request about every 15-19 minutes.

Client session control

Normally, if the client drops the indication connection (closes TCP socket of the GET request), after 90 seconds the session times out. If a client repeatedly creates and drops indication connections during these 90 seconds, after the 16th request or so, the next request is rejected with the reason "Too many users". In order for the client to control the session allocation and let the Control Station deallocate the session on time, there is a special HTTP header: `CelerraConnector-Ctl`. It can have two values for the server:

- *DISCONNECT* – The client no longer requires this session, and it can be deallocated immediately. The body of the message should not contain any data.
- *ONE-TIME-REQUEST* – The client wants to perform this request and does not need a session, so it can be deallocated at the completion of the request.

An example of a graceful session deallocation follows:

```
POST /servlets/CelerraManagementServices HTTP/1.1
Host: 172.24.173.31:443
Content-Type: text/xml
Cookie:
Ticket=ip&10.13.3.28&idle&0&persists&&last&1127309795&expires&480&hash&33163d07
49ba813fc7be364c2445b2d0&user&nasadmin&type&User&time&1127309795; $Path=/
CelerraConnector-Sess: 00FBDEE3626FF75A8FB6D683FFF4B5E5
CelerraConnector-Ctl: DISCONNECT
Content-Length: 0
HTTP/1.1 200 OK
Date: Wed, 21 Sep 2005 13:36:36 GMT
Server: Apache/1.3.27 (Unix) (Red-Hat/Linux) mod_jk/1.2.5 mod_ssl/2.8.12
OpenSSL/0.9.6b mod_perl/1.24_01
Set-Cookie: JSESSIONID=00FBDEE3626FF75A8FB6D683FFF4B5E5; Path=/; Secure
CelerraConnector-Sess: 00FBDEE3626FF75A8FB6D683FFF4B5E5
Content-Length: 0
Content-Type: text/xml;charset=UTF-8
```

Technical Support

For technical support, call your local sales office.

If you have a valid EMC service contract, contact Customer Service at:

United States: (800) 782-4362 (SVC-4EMC)
Canada: (800) 543-4782 (543-4SVC)
Worldwide: (508) 497-7901

Your Comments

Your suggestions will help us continue to improve the accuracy, organization, and overall quality of the user publications. Please e-mail us at celerradoc_comments@emc.com to let us know your opinion or any errors concerning this documentation.

For additional information on resolving any technical problems you might encounter, EMC recommends that you follow the problem resolution roadmap described in the Celerra Network Server Problem Resolution Roadmap technical module, which is located on the Celerra Network Server User Information CD.

As part of its effort to continuously improve and enhance the performance and capabilities of the Celerra Network Server product line, EMC from time to time releases new revisions of Celerra hardware and software. Therefore, some functions described in this document may not be supported by all revisions of Celerra software or hardware presently in use. For the most up-to-date information on product features, see your product release notes.

If your Celerra system does not offer a function described in this document, please contact your EMC customer support representative for a hardware upgrade or software update.