NetApp™

Go further, faster™

# NetApp Manageability SDK Training

## Java Programming

# SDK Java Interface - Core Classes

- NaElement
  - Encapsulates one level of an XML element.
  - Elements can be arbitrarily nested.
  - Elements have names, attributes (only used for results), values (always strings) and possibly children
- NaServer
  - Encapsulates an administrative connection to a NetApp filer
- NaAPIFailedException
  - Exception for the API invocation failures.
  - Indicates that server processed the request and returned a failure result including an error code and a reason string.

# SDK Java Interface – Core Methods

- NaElement
  - set*()
    - Provide set operations on an element
  - getChild*()
    - Provide get operations on child elements
  - add*()
    - Provide add operations on the child elements
- NaServer
  - set*()
    - Provide operations for setting connection parameters – port, login style, server type, transport type, admin user etc
  - get*()
    - Provide operations for getting connection parameters – port, login style, server type, transport type, admin user etc
  - invokeElem()
    - Provides operation for sending a command to the server
- NaAPIFailedException
  - getErrno(), getReason()
    - Provide operations to get details about the failure

# Configuring Session Parameters

- Server type
  - Indicates the target appliance for API calls
  - It can be set using the method *NaServer::setServerType()*
  - parameter to *setServerType()* method can take the following values:

    - *SERVER_TYPE_FILER*
      - To send API calls to NetApp Storage system
    - *SERVER_TYPE_DFM*
      - To send API calls to DataFabric Manager server

# Configuring Session Parameters

- Server style
  - Indicates the authentication mechanism to be used for communicating with the Server
  - It can be set using the method *NaServer::setStyle()*
  - parameter to *setStyle()* method can take the following values:
    - *STYLE_LOGIN_PASSWORD*
      - Causes the Server to use HTTP simple authentication with a username and password.
      - User needs to provide login/password details with *set_admin_user()* method
    - *STYLE_HOSTSEQUIV*
      - Causes the Server to authenticate against the /etc/hosts.equiv file on the filer
    - *STYLE_RPC*
      - Causes the Server to use native Windows authentication and authorization

# Configuring Session Parameters

- Transport type
  - Indicates the transport mechanism to be used for sending API commands
  - It can be set using the method *NaServer::setTransportType()*
  - Parameter to *setTransportType()* method can take the following values:
    - TRANSPORT_TYPE_HTTP
      - Uses HTTP as transport protocol
      - This is the default transport type
    - TRANSPORT_TYPE_HTTPS
      - Uses HTTPS as transport protocol
      - Provides secure communication by encrypting the API requests/responses
    - TRANSPORT_TYPE_SSH (Unsupported)
      - Uses SSH transport protocol
      - Provides secure remote login facility

# Configuring Session Parameters

- Server Port
  - Indicates the port on/to which API requests to be sent
  - Depends on ‘Transport type’ and ‘Server type’ settings
  - It can be set using the method *NaServer::setPort()*
  - This setting should be done only after setting ‘Transport type’ and ‘Server type’ values

# Java Programming with SDK – Basic Steps

- STEP 1: Create Server Context on Client
  - Create Server Object with Server Name / IP, Product API version to be used as inputs
    - NaServer s = new NaServer(server-name, majorversion, minorversion)

- STEP 2: Set Session parameters on Client
  - Set server type : *s->setServerType()*
  - Set Authentication style : *s->setStyle()*
  - Set Transport type : *s->setTransportType()*
  - Set Server port : *s->setPort()*

# Java Programming with SDK – Basic Steps

- STEP 3: Send Commands to the Server
  - Create a Command element and use the Core API *invokeElem() of NaServer Class*

    E.g.

    *NaElement elem = new NaElement("system-get-version");*

    *S->invokeElem(elem);*

# Java Programming with SDK – Basic Steps

- STEP 4: Check for API Failures
  - *NaServer* class methods for setting session parameters (*set*()*) throw '*java.lang.IllegalArgumentException'* exception in case of an error
  - *NaElement* class methods for extracting the child element details return 'null' in case of an error
  - Commands to the server through *invokeElem()* method raise the following exceptions in case of errors:

    NaAuthenticationException - if the login was not accepted by the server

    NaAPIFailedException - if the server returned a failure result.

    NaProtocolException - if there is a protocol problem. It means that the server responded with something that we couldn't understand.

    java.io.IOException - if there is a communication problem
  - Product API related failures result in *NaAPIFailedException* exception
    - API failure details can be extracted the using *getErrno() & getReason()* methods of *NaAPIFailedException* class

# Java Programming with SDK – Basic Steps

- STEP 5: Parse the command results
  - The *invokeElem()* Core API always returns *NaElement* object
  - The result of Product API is embedded as a child *NaElement* object in the *NaElement* object returned from *invokeElem()* Core API

E.g.:

```
NaElement quotaElem = new NaElement("quota-list-entries");
NaElement out = s->invokeElem(quotaElem);
NaElement quota_data = out->getChildByName ("quota-entries");
List quotaList = quota_data.getChildren();
Iterator quotaIter = quotaList.iterator();
while(quotaIter.hasNext()){
        NaElement quotaInfo=(NaElement)quotaIter.next();
}
```

# Compiling SDK Java Program

- manageontap.jar  has all the SDK Java library classes
- The full path for this library in the SDK is %sdk-root%/lib/java/classes/manageontap.jar.
- manageontap.jar file should be included in the Java classpath for compiling SDK Java Program

  E.g.

  $javac -classpath <path>/manageontap.jar hello.java

# SDK Java Interface – Differences from other interfaces

- Windows RPC Style authentication mechanism is not supported
- No option to set timeout for the client connection in cases of delayed response from the server
- Java does not support variable length argument lists, so invoke() routine is not available unlike in PERL, C interfaces

# Tutorial Exercises

# Tutorial Exercises

1. Monitor volumes on a NetApp storage system and print message when space usage crosses a threshold

   - Hint:
     - Get list of volumes with the command '*volume-list-info*'
     - Extract values of '*size_total*' and '*size_used*' fields from '*volume_info*' return element

# Tutorial Exercises Contd..

2. Retrieve the quota information of a NetApp storage system in a secure way and print quota information : quota-target,vol name,quota-type

- Hint:
  - Set the transport type as ‘TRANSPORT_TYPE_HTTPS’
  - Get the quota list through the command ‘*quota-list-entries*’

# Tutorial Exercises Contd..

3. Enable NFS server on NetApp storage system and check the status of the NFS server. Get the list of the export rules on the storage system.

- Hint:
  - Use the command '*nfs-enable*' for enabling NFS server
  - Use the command '*nfs-status*' to check the status
  - Use the command '*nfs-exportfs-list-rules*' to get the list of the NFS export rules

NetApp

# Thank You