NtcpHelper Reference
(NTCP Client)

Laura Pearlman¹, Mike D’Arcy¹

¹USC Information Sciences Institute, Marina del Rey, CA

Feedback on this document should be directed to neesgrid-si@neesgrid.org

Acknowledgment: This work was supported primarily by the George E. Brown, Jr. Network for Earthquake Engineering Simulation (NEES) Program of the National Science Foundation under Award Number CMS-0117853.
1  NtcpHelper ........................................................................................................ 3
1.1  Initiating a connection: activateNtcpServer .............................................. 3
1.2  Methods corresponding to NTCP protocol Requests ............................... 4
    1.2.1  openSession .................................................................................. 4
    1.2.2  Propose .......................................................................................... 5
    1.2.3  Execute ......................................................................................... 5
    1.2.4  getTransaction ............................................................................ 5
    1.2.5  getControlPoint .......................................................................... 5
    1.2.6  Cancel .......................................................................................... 6
    1.2.7  getParameter ................................................................................ 6
    1.2.8  setParameter ................................................................................ 6
    1.2.9  getParameters .............................................................................. 6
    1.2.10 closeSession ................................................................................. 6
    1.2.11 getParameter ................................................................................ 6
    1.2.12 proposeAndExecute ..................................................................... 6
    1.2.13 Reset ............................................................................................ 7
1.3  Utility Methods ......................................................................................... 7
    1.3.1  getParameter ................................................................................ 7
    1.3.2  getCP ............................................................................................ 7
    1.3.3  getCPGeomParamType ................................................................ 7
    1.3.4  Other utility methods ................................................................... 8
2  Other Classes Related to NtcpHelper ............................................................ 8
2.1  The ControlPointType Class ................................................................ 8
    2.1.1  ControlPointGeomParameterType ............................................... 9
2.2  TransactionType ..................................................................................... 9
3  Acknowledgements ..................................................................................... 10
1 NtcpHelper

NtcpHelper is an NTCP client class. The NTCP protocol itself is described in the NTCP protocol Document\(^1\); we assume the reader is familiar with that document and with the NTCP protocol. The NtcpHelper class requires that the following be imported:

```java
import org.nees.ntcp.ntcpServer.ParameterType;
import org.nees.ntcp.ntcpServer.ControlPointType;
import org.nees.ntcp.ntcpServer.ControlPointParameterNameType;
import org.nees.ntcp.ntcpServer.ControlPointGeomParameterType;
import org.nees.ntcp.ntcpServer.GeomAxisType;
import org.nees.ntcp.ntcpServer.TransactionType;
import org.nees.ntcp.ntcpServer.TransactionStateType;
import org.globus.ogsa.impl.security.authentication.Constants;
import org.nees.ntcp.server.util.NtcpHelper;
import org.nees.ntcp.ntcpServer.NtcpServer;
import java.math.BigInteger;
```

1.1 Initiating a connection: activateNtcpServer

The `activateNtcpServer` call is used to initiate a connection to an NTCP server. The `serverURL` is the URL of the container in which the NTCP server is running (typically something like “http://hostname:port/ogsa/services/nees/ntcp”, and `instanceName` is the name of the NTCP instance within that container (typically “NTCPServer”).

If present, the `isSecure` or `securityMechanism` argument determines what mechanism (if any) will be used to authenticate to the NTCP server.

| activateNtcpServer(serverURL, instanceName) | Authentication is attempted using the XML signature mechanism. This mechanism can handle more network interruptions, for many applications, yields better performance, than the secure conversation mechanism. |
| activateNtcpServer(serverURL, instanceName, Constants.GSI_XML_SIGNATURE) | Authentication is attempted using the XML signature mechanism. This mechanism can handle more network interruptions, for many applications, yields better performance, than the secure conversation mechanism. |
| activateNtcpServer(serverURL, instanceName, false) | No authentication is attempted. |
| activateNtcpServer(serverURL, instanceName, null) | No authentication is attempted. |
| activateNtcpServer(serverURL, instanceName) | Authentication is attempted using the |
The result of a successful `activateNtcpServer` call is an `NtcpServer` object which can be used to communicate with an NTCP server. The values of two system properties, `NtcpAuthType` and `NtcpAuthDetail`, control whether or not the client program will verify the identity of the NTCP server:

<table>
<thead>
<tr>
<th>NtcpAuthType</th>
<th>NtcpAuthDetail</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>“host”</td>
<td>null</td>
<td>Host authorization is performed: the client will communicate only with a server that authenticates itself with a host certificate corresponding to the host named in the <code>serverURL</code> argument to <code>activateNtcpServer</code>.</td>
</tr>
<tr>
<td>“host”</td>
<td>serviceName</td>
<td>Service authorization is performed: the client will communicate only with a server that authenticates itself with a service certificate corresponding to the service named in <code>serviceName</code> and the host named in the <code>serverURL</code> argument to <code>activateNtcpServer</code>.</td>
</tr>
<tr>
<td>“identity”</td>
<td>distinguishedName</td>
<td>The client will communicate only with a server that authenticates itself with the identity <code>distinguishedName</code>.</td>
</tr>
<tr>
<td>“self”</td>
<td></td>
<td>The client will communicate only with a server that authenticates with an identity identical to the client’s identity.</td>
</tr>
<tr>
<td>Not present</td>
<td></td>
<td>The client will communicate with any server, regardless of the server’s identity.</td>
</tr>
</tbody>
</table>

*Note:* the activateNtcpServer does not actually send any requests to the NTCP server; it simply creates a data object that can be used later to send requests to the server.

### 1.2 Methods corresponding to NTCP protocol Requests

The methods in this section are used to send requests to an NTCP server. Each request takes an `NtcpServer` object as its first argument; this object should be the result of a prior call to `activateNtcpServer`.

#### 1.2.1 openSession

```java
public static void openSession(NtcpServer ntcp, ParameterType[] parameters) throws Exception
```

The `openSession` method is used to send an NTCP openSession request. The `parameters` argument is an array of parameters as described in the NTCP protocol document.
ParameterType objects are created using the getParameter utility method described in section 1.3.1.

1.2.2 Propose

```
public static TransactionStateType propose (NtcpServer ntcp, String transactionName, BigInteger stepNumber, ControlPointType[] controlPoint, int proposeTimeout, int transactionTimeout, int transactionRememberedUntil) throws Exception
```

The propose method sends a propose request to an NTCP server. The transactionName, stepNumber, and controlPoint arguments are as described in the NTCP protocol document. The three timeout arguments (proposeTimeout, transactionTimeout, and transactionRememberedUntil) specify the corresponding timeout values as in the NTCP protocol; however, each of these arguments should be expressed as a number of seconds from the current time, rather than as an absolute time value.

The ControlPointType class is described in section 2.1.

If the proposal is accepted by the NTCP server, the propose call will return the value org.nees.ntcp.ntcpServer.TransactionStateType.accepted. If the proposal is rejected, the propose call will return the value org.nees.ntcp.ntcpServer.TransactionStateType.terminated.

1.2.3 Execute

```
public static void execute(NtcpServer ntcp, String transactionName) throws Exception
```

The execute method is used to send an execute request to the NTCP server (the results of a transaction can be queried by calling the getTransaction method described in section 1.2.4).

1.2.4 getTransaction

```
static TransactionType getTransaction(NtcpServer ntcp, java.lang.String transactionName)
```

The getTransaction method polls the server for the status of the named transaction; when that transaction is terminated, it returns a TransactionType object corresponding to the state of that transaction. TransactionType objects are described in section 2.2

1.2.5 getControlPoint

```
public static ControlPointType getControlPoint(NtcpServer ntcp, String name) throws Exception
```
the `getControlPoint` method sends a `getControlPoint` request to an NTCP server. If successful, it returns a `ControlPointType` object representing the current measured (or calculated) state of the requested control point. The `ControlPointType` class is described in section 2.1.

### 1.2.6 Cancel

```java
cancel(JNIEnv* env, jobject javaNtcpServer, jobject javaTransactionName, jobject javaInterruptWhileExecuting)
```

The `cancel` method sends an NTCP `cancel` request.

### 1.2.7 getParameter

```java
getParameter(JNIEnv* env, jobject javaNtcpServer, jobject javaName)
```

The `getParameter` method sends a `getParameter` request to an NTCP server and, if successful, returns the parameter value. This should not be confused with the `getParameter` utility method described in section 1.3.1.

### 1.2.8 setParameter

```java
setParameter(JNIEnv* env, jobject javaNtcpServer, jobject javaName, jobject javaValue)
```

The `setParameter` method sends a `setParameter` request to an NTCP server.

### 1.2.9 getParameters

```java
getParameters(JNIEnv* env, jobject javaNtcpServer)
```

The `getParameters` method queries the server for the names and values of all parameters known to the server.

### 1.2.10 closeSession

```java
closeSession(JNIEnv* env, jobject javaNtcpServer)
```

The `closeSession` method sends a closeSession request to an NTCP server.

### 1.2.11 getParameter

```java
getParameter(JNIEnv* env, jobject javaNtcpServer, jobject javaName) throws Exception
```

The `getParameter` call sends a `getParameter` call to the NTCP server to get the value of an experiment parameter.

### 1.2.12 proposeAndExecute

```java
proposeAndExecute(JNIEnv* env, jobject javaNtcpServer, jobject javaTransactionName, jlong stepNumber, jobject javaControlPointTypes)
```

The `proposeAndExecute` method sends a proposeAndExecute call to the NTCP server to propose and execute a control point change.
controlPoint, int proposeTimeout, int transactionTimeout, 
int transactionRememberedUntil) throws java.lang.Exception, 
GridServiceException, InterruptedException, NtcpHelperException, 
java.rmi.RemoteException

The proposeAndExecute method sends a proposeAndExecute request to the NTCP server 
to propose and execute a transaction. The return value is the resulting transaction state.

1.2.13 Reset

public static void reset(NtcpServer ntcp) throws 
java.rmi.RemoteException, java.lang.InterruptedException, 
NtcpHelperException

The reset method sends a reset request to the NTCP server. The reset request causes the 
server to clear its state related to all current and previous transactions and should only be 
used by administrators who are sure that none of this information will be needed.

1.3 Utility Methods

These methods are used to convert data types used by NTCP.

1.3.1 getParameter

static ParameterType getParameter(java.lang.String name, 
java.lang.String value)

This method is used to create a ParameterType object from a name and value. This 
should not be confused with the getParameter method described in section 1.2.7, which 
queries an NTCP server for the value of a parameter.

1.3.2 getCP

static ControlPointType getCP(java.lang.String name, 
ControlPointGeomParameterType[] elements)

This method creates a ControlPointType object from a name and an array of 
ControlPointGeomParameterType objects.

1.3.3 getCPGeomParamType

static ControlPointGeomParameterType getCPGeomParamType(String name, 
String axis, double value) 
static ControlPointGeomParameterType getCPGeomParamType(String name, 
String axis, Float value)

This method creates a ControlPointGeomParameterType object from a name, an axis, and 
a value.
1.3.4 Other utility methods

static java.util.Vector getControlPointArrayAsVector(ControlPointType[] controlPoints)
static java.util.Vector getObjectArrayAsVector(java.lang.Object[] objects)
static java.util.Vector getParameterArrayAsVector(ParameterType[] parameters)

These methods convert arrays to vectors.

2 Other Classes Related to NtcpHelper

2.1 The ControlPointType Class

A ControlPointType object is used to specify values associated with a control point; these may be values representing an action requested on a control point, or measured/calculated values representing the state of a control point. A control point can be thought of as having a name and an array of (zero or more) values, each of which corresponds to (for example) a force or displacement along some axis. The methods within ControlPointType are described here.

public ControlPointType()

The ControlPointType constructor takes no arguments and creates an “empty” ControlPointType object (with no name or control points associated with it).

public void setControlPointName(java.lang.String controlPointName)
public java.lang.String getControlPointName()

The setControlPointName sets the control point’s name; getControlPointName gets the control point’s name (i.e., returns the name that was set by the most recent call to setControlPointName). Generally, setControlPointName will be called only once during the life of a ControlPointType object.

public void setControlPointType(ControlPointGeomParameterType[] controlPointType)
public void setControlPointType(int i, ControlPointGeomParameterType value)

The setControlPointType methods set the values associated with the control point (ControlPointGeomParameterType is described below). The first form sets the entire array; the second is used to set one value at a time.

public ControlPointGeomParameterType[] getControlPointType()
public ControlPointGeomParameterType getControlPointType(int i)

The getControlPointType methods get the values associated with the control point. The first form returns the entire array; the second returns the \( i \)th entry in the array.
2.1.1 ControlPointGeomParameterType

The ControlPointGeomParameterType object is used to represent a geometric parameter (such as “2 cm. displacement along the X axis”). The methods belonging to this type are described here:

```java
public ControlPointGeomParameterType()
```

The constructor takes no arguments and creates an “empty” ControlPointGeomParameterType object.

```java
public void setName(ControlPointParameterNameType name)
public ControlPointParameterNameType getName()
```

The setName method sets the name of the parameter (that is, the name describing what kind of parameter this object represents); name should be one of these statically-defined objects:

```java
ControlPointParameterNameType.force
ControlPointParameterNameType.moment
ControlPointParameterNameType.displacement
ControlPointParameterNameType.rotation
```

The getName method returns the parameter’s name (the name set by setName).

```java
public void setAxis(GeomAxisType axis)
public GeomAxisType getAxis()
```

The setAxis method sets the axis associated with this parameter; axis should be one of these three statically-defined objects:

```java
GeomAxisType.x
GeomAxisType.y
GeomAxisType.z
```

The getAxis method returns the parameter’s axis (the axis set by setAxis).

```java
public void setValue(java.lang.Float value)
public java.lang.Float getValue()
```

The setValue method sets the parameter’s value; getValue returns the parameter’s value.

2.2 TransactionType

A TransactionType object represents the state of a transaction. The following methods are provided to examine the values of the various TransactionType fields (see the definition of the TransactionType XML object in the NTCP protocol document for the meaning of each of these fields):

```java
java.lang.String getName()
```
ControlPointType[] getRequestedControlPoints()
ControlPointType getRequestedControlPoints(int i)
ControlPointType[] getResultingControlPoints()
ControlPointType getResultingControlPoints(int i)
TransactionStateType getState()
org.gridforum.ogsi.ExtendedDateTimeType
  getTransactionExecutionBeginTime()
java.lang.String getTransactionProposerName()
org.gridforum.ogsi.ExtendedDateTimeType
  getTransactionRememberedUntil()
org.gridforum.ogsi.ExtendedDateTimeType
  getTransactionTerminationTime()
org.gridforum.ogsi.ExtendedDateTimeType getTransactionTimeout()

3 Acknowledgements
Erik Johnson provided a great deal of useful feedback during the development of the
NtcpHelper class and contributed some of the utility routines. Paul Hubbard provided
helpful feedback on earlier versions of this document.

1 L. Pearlman, M. D’Arcy, E. Johnson, C. Kesselman, P. Plaszczak. NEESgrid Teleoperation Control