Technical Report NEESgrid-2003-20
www.neesgrid.org

(Whitepaper Version: 0.4)
Last modified: September 2, 2003

NtcpHelper Reference
(DRAFT)

Laura Pearlman[1], Mike D'Arcy[1]

[1]USC 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.

TABLE OF CONTENTS

1 Summary

This document is a reference manual for NtcpHelper, the NTCP client API, which provides an interface to the NTCP server. The guide provides descriptions of all NtcpHelper functions and example syntax, as well as functions and syntax of other classes related to NtcpHelper.

2 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:

```
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.nees.ntcp.server.util.NtcpHelper;
import org.nees.ntcp.ntcpServer.NtcpServer;
import java.math.BigInteger;
```

2.1 Initiating a connection: activateNtcpServer

```
public static NtcpServer activateNtcpServer(String serverURL, String
    instanceName) throws Exception
public static NtcpServer activateNtcpServer(String serverURL, String
    instanceName, boolean isSecure) throws Exception
```

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*:8080/" and instance name of the NTCP server to connect to, and instanceName is the name of the NTCP instance within that container (typically "NTCPServer"). The *isSecure* parameter, if present, determines whether or not to attempt to make an authenticated connection (if *isSecure* is not present, it is assumed to be true).

The result of a successful activateNtcpServer call is an NtcpServer object which can be used to communicate with an NTCP server.

2.2 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*.

2.2.1 openSession

```
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 2.3.1.

2.2.2 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 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 3.1.

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`.

2.2.3 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 2.2.4).

2.2.4 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 3.2

2.2.5 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 *ControlPointTyp*e class is described in section 3.1.

2.2.6 Cancel

```
static void cancel(NtcpServer ntcp, java.lang.String transactionName,
    java.lang.Boolean interruptWhileExecuting)
```

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

2.2.7 getParameter

```
static java.lang.String getParameter(NtcpServer ntcp,
    java.lang.String name)
```

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* utililty method described in section 2.3.1.

2.2.8 setParameter

```
static void setParameter(NtcpServer ntcp, java.lang.String name,
    java.lang.String value)
```

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

2.2.9 getParameters

```
static ParameterType[] getParameters(NtcpServer ntcp)
```

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

2.2.10 closeSession

```
static void closeSession(NtcpServer ntcp)
```

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

2.3 Utility Methods

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

2.3.1 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 2.2.7, which queries an NTCP server for the value of a parameter.

2.3.2 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.

3 Other Classes Related to NtcpHelper

3.1 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 *ith* entry in the array.

3.1.1 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:

```
public ControlPointGeomParameterType()
```

The constructor takes no arguments and creates an "empty" *ControlPointGeomParameterType* object.

```
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:

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

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

```
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:

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

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

```
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.

3.2 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.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()
```

4 Acknowledgements

Erik Johnson provided a great deal of useful feedback during the development of the NtcpHelper class.

[1] L. Pearlman, M. D'Arcy, E. Johnson, C. Kesselman, P. Plaszczak. NEESgrid Teleoperation Control Protocol (NTCP). NEESgrid Technical Report 2003-07. September 2003.