

Table of Contents

Table of Contents	1
Including the DipZoom Package	2
Placing the package correctly – Command Line	2
Placing the package correctly – Eclipse software	4
Importing the package’s contents	5
DipzoomClientLibrary Class	6
login	6
getMeasuringPointList	7
sendRequest	8
sendRequest2	11
getTicketStatus	11
requestResult	12
getFinishedMeasurements	13
clearFinishedTicketXML	14
DipzoomConstants Class	15
Relevant constants:	15
statusCodeToString	15
Measurement Class	16
toString	16
equals	16
getNonce	17
getTransactionStatus	18
getResult	19
MeasurementRequest Class	20
Dipzoom Exceptions	22
Currently Supported Measurement Types	24
Possible User Errors (Troubleshooting):	25

Including the DipZoom Package

As with any other Java packages, in order to utilize the Dipzoom Client Library, its package must be imported into the program that intends to use the library. This import is handled in two steps:

1. Placing the Client Library package in a location that will be correctly recognized by the compile/run environment.
2. The actual import statement within the program (and the subsequent instantiation of the desired class within the package).

Placing the package correctly

The Java compile/run environment must know where to find the package. Thus, either it must be placed in a default location, or the compiler and run-time must be told explicitly where to find the package. This walkthrough will examine two cases; compiling from the command line, and from the Eclipse platform. It will also consider Windows and Linux operating systems.

Placing the package correctly – Command Line

Java Virtual machine uses the CLASSPATH environment variable to find any third-party or user-defined classes that may be used by the current program. Normally, CLASSPATH is set to point both to the current directory and to the /bin subfolder of the Java program folder. As such, any .class files that are moved to these locations will automatically be discovered by the JVM during regular compilation/execution.

The current setting for the CLASSPATH environment variable can be checked using the command (in Windows and Linux, respectively)

```
echo %CLASSPATH%  
echo $CLASSPATH
```

which will return all locations the CLASSPATH variable is currently tied to. A single period indicates the current directory.

To use one of these default locations, one needs to extract the class files from the dipzoom_client_lib.jar file and place a folder with these classes in that location. The .class files can be extracted using the command

```
jar -xf dipzoom_client_lib.jar
```

executed from the directory that contains the above .jar file. Then, the edu folder (which contains the necessary .class files in the cwrw\dipzoom\lib subfolder) must be copied to the desired default location.

If we do not wish to extract classes from the .jar file, we must add the pathname of this file to the CLASSPATH variable. This can be done by either using the command line option `-classpath` when invoking the `javac` and `java` commands, or modifying the CLASSPATH variable explicitly.

Let's say our downloaded package is located in the folder `C:\dipzoom>window`. We want to import the class files stored with the file `dipzoom_client_lib.jar`. Our program which imports this package is named `test.java`. In order to compile our program correctly from the command line, our command should look like this (using Windows in this example):

```
javac -classpath "C:\dipzoom>window\dipzoom_client_lib.jar" test.java
```

and our command to run will look like so:

```
java -classpath "C:\dipzoom>window\dipzoom_client_lib.jar" ;. test
```

Note that the classpath argument in the "java" command contains the extra characters `;. -` – this is so that the Java Virtual Machine will look for classes from both the specified jar file and the relative directory (as designated by the period). Any extra classpath locations can be specified by placing semicolon between each listed location. Note that in a Linux-based machine, the colon (`:`) is used in lieu of a semicolon.

In order to avoid having to specify the classpath of our library's jar file on every compile/execution, we can temporarily add this location to the CLASSPATH environment variable. This is performed through the command, in Windows,

```
set CLASSPATH= %CLASSPATH%;C:\dipzoom>window\dipzoom_client_lib.jar
```

or, in Linux,

```
export CLASSPATH=$CLASSPATH: /dipzoom/window/dipzoom_client_lib.jar
```

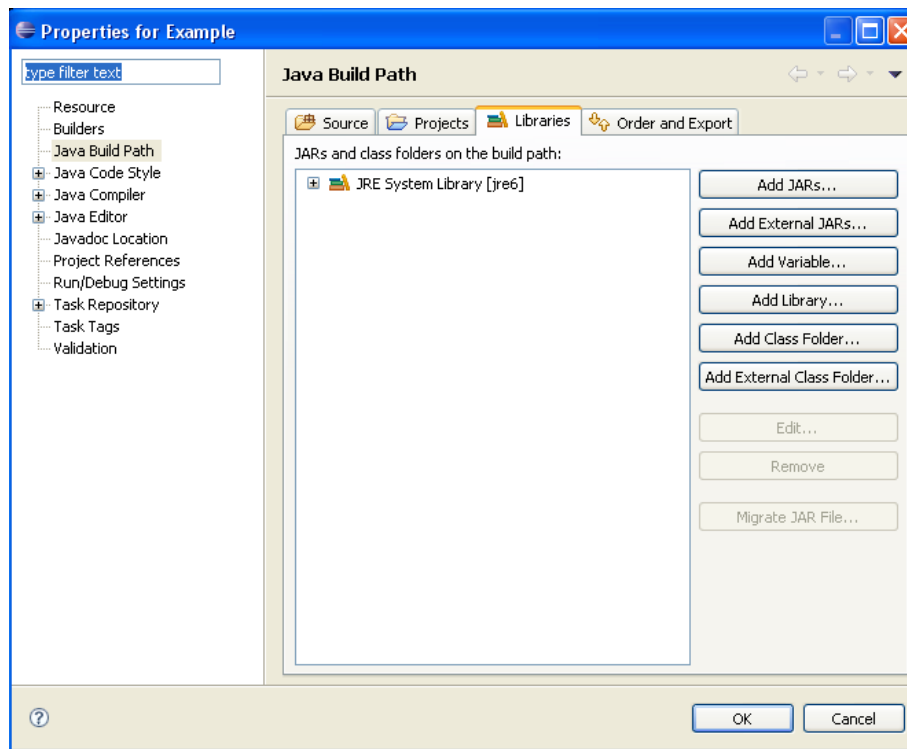
Note that these assignments are not permanent, and will not be maintained once the current shell is exited. Steps to permanently add our library location to the CLASSPATH variable vary by operating system. In a Windows system, for example, the general path for setting the classpath variable should be `Start->My Computer (Right-Click) -> Properties -> Advanced -> Environment Variables -> Edit (With CLASSPATH selected)`.

Placing the package correctly – Eclipse software

(This guide is written with respect to Eclipse version 3.4.1)

Instead of using the CLASSPATH environment variable during its compilation and execution, the Eclipse platform allows the user to specify a different build path for each project. This build path includes any and all locations from which classes can be imported.

In order to edit the build path of a project, we right-click the desired project in the Package Explorer, and select Build Path -> Configure Build Path (Alternately, we can select the desired project, and select the menu button Project -> Properties). This should take us to this screen:



Make sure the Libraries tab is selected. Note that the Java library is automatically included in every project.

From this point, adding the Dipzoom Client Library is a simple matter – select “Add External JARs...” and locate/select the dipzoom_client_lib.jar file. Now, the Client Library will be included in the build path for this project and can be imported for use by any program created in this project folder. Note that when creating a new project, these steps must be repeated for that project’s build path.

At this point, when running the Java compiler or attempting to run our program, the Virtual Machine will be able to locate the class variables within the DipzoomClientLibrary package. However, references to these classes will still result in errors. In order to use these classes, we must now specifically state our intention to import them within our program's code.

Importing the package's contents

Importing packages should be a familiar routine to all Java programmers. To utilize the Random class, one must import java.util.Random; for JOptionPane methods, javax.swing.JOptionPane must first be imported. The same is true of the classes contained within the DipzoomClientLibrary.

Within the dipzoom_client_lib.jar file included in the Dipzoom Client Library package, the DipzoomClientLibrary classes are located in the subfolder edu\cwru\dipzoom\lib. As such, in order to import the class files within the package, we must use these import commands at the beginning of our program:

```
import edu.cwru.dipzoom.lib.DipZoomClientLibrary;  
import edu.cwru.dipzoom.lib.DipZoomConstants;  
...
```

or import all classes in the package using the single command

```
import edu.cwru.dipzoom.lib.*;
```

Now, we can freely use any of the classes contained within the Dipzoom Client Library. Documentation for these classes and their methods can be found [here](#).

All methods defined in the Dipzoom Client Library are members of the class DipzoomClientLibrary. Thus, to use any of these methods, we must first initialize an object of the class DipzoomClientLibrary, which we can then use to invoke these methods. This can be done, for example, using the following lines of code within our program:

```
DipzoomClientLibrary library = new DipzoomClientLibrary(); //creates a new  
                                                                DipzoomClientLibrary object  
  
library.login() //we can now invoke any of the methods defined in the  
                DipzoomClientLibrary documentation as members of the "library"  
                object.
```

DipzoomClientLibrary Class

Description: The DipzoomClientLibrary class contains all relevant methods that can be performed by the user in order to gain measurements from the Dipzoom network. The supporting classes such as DipzoomConstants, MeasurementRequest, and Measurement, are used by this class in order to package and interpret its interactions with the Dipzoom Core.

Constructors:

```
DipzoomClientLibrary()
```

The default constructor.

Methods:

login

```
public void login(java.lang.String filename) throws DipzoomException
```

Attempts to log into the Dipzoom Core(Server) using the address, Client ID, and encryption key specified in login.xml (included in the downloaded package). Client ID and encryption key will vary between packages, and should not be altered.

Note: If the login.xml file is lost/deleted, the function will attempt to establish a new ID and encryption key; this attempt will be successful as long as the client's JRE has been updated to version 1.5 or later.

Throws:

LoginException – thrown if login fails for any of three reasons: invalid login information from login.xml; invalid encryption method in KeyUtil.java; JRE not updated to version 1.5 or later.

logout

```
public void logout() throws DipzoomException
```

Logs out from the server.

Throws:

DipZoomException - Usually thrown due to internal system errors. Simple retry is suggested.

getMeasuringPointList

```
public MeasuringPoint[]  
getMeasuringPointList(java.util.Hashtable<java.lang.String, java.lang.St  
ring> measuringParameters) throws DipzoomException
```

Queries the DipZoom server for a list of measuring points that match certain parameters, as specified in the supplied hashtable.

Parameters:

HashTable<java.lang.String, java.lang.String> measuringParameters – A HashMap which contains key/value pairs for some or all of the following filters:

- String country
- String region
- String city
- String asnum (Autonomous System number)
- String operatingSystem
- String measurementType
- String measurementNumber
- String bandwidth

These filters are passed into the HashMap using the command format measuringParameters.put(filter, filter_value). For example, to query for measuring points capable of performing a ping measurement, one would need to use the command measuringParameters.put(“measurementType”, “ping”) before passing measuringParameters into the getMeasuringPointList() function.

All values besides measurementType can be left blank or “any” to signify no limitation on that filter. However, measurementType must contain exactly one type of measurement that the user wants to perform. A list of currently supported measurement types can be found [here](#).

In order to create a list of measuring points that can each handle multiple types of measurements, one should collect and compare individual lists for each type of measurement desired.

Returns: A MeasuringPoint[] array which contains all points that match the specified criteria, in the order returned by the server.

Throws:

LoginException – Thrown if the function is called before the user logs in (via the login() method).

MultipleLoginException – Thrown if another user attempts to log in using the same Client ID as the current user. A simple re-try is recommended. If problem persists, a new ID can be generated by deleting the login.xml file included in the package, then re-attempting to log in; however, the user’s JRE version must be updated to 1.5 or later.

DipZoomException - Usually thrown due to internal system errors (though certain error messages may imply user fault). Simple retry is suggested.

sendRequest

```
public java.util.ArrayList<Measurement>  
sendRequest(MeasurementRequest[] measurementRequests,  
MeasuringPoint[] measuringPoints) throws DipzoomException
```

Sends a request to the DipZoom server that each measuring point listed performs all measurement requests (with x requests and y measuring points, a total of xy measurements will be performed).

Note that this method only sends requests; the user must utilize other functions such as `getTicketStatus()` and `requestResult()` to monitor the status and process the results of these requests.

Parameters:

`MeasurementRequest[] measurementRequests` – contains the set of requests to be performed.

`MeasuringPoint[] measuringPoints` – contains the set of measuring points, each of which should perform all requests.

Returns: An `ArrayList<Measurement>` that contains all measurements that have been sent to the server by the function. This list can be used during future steps which track the progress on these measurements.

Throws:

`LoginException` – Thrown if the function is called before the user logs in (via the `login()` method).

`MultipleLoginException` – Thrown if another user attempts to log in using the same Client ID as the current user. A simple re-try is recommended. If problem persists, a new ID can be generated by deleting the login.xml file included in the package, then reattempting to log in; however, the user’s JRE version must be updated to 1.5 or later.

`TicketLimitException` – The user has requested more than his/her personal ticket limit, which is currently set to 2000. To fix this problem, the user should either collect any

finished results or wait for results to return / old outstanding requests to be cleared automatically by the DipZoom server.

GlobalTicketLimitException – The total number of outstanding requests to the server exceeds the global limit, which is currently 10000. The user should try again at a later time to ease the server load.

DipZoomException - Usually thrown due to internal system errors (though certain error messages may imply user fault). Simple retry is suggested.

sendRequest

```
public Measurement sendRequest(MeasurementRequest measurementRequest,  
MeasuringPoint measuringPoint) throws DipzoomException
```

Sends a request to the DipZoom server that the specified measuring point performs a single measurement request.

Note that this method only sends the request; the user must utilize other functions such as `getTicketStatus()` and `requestResult()` to monitor the status and process these results of this request.

Parameters:

`MeasurementRequest measurementRequest` – The request the user wishes to perform.
`MeasuringPoint measuringPoint` – The measuring point which should perform the request.

Returns: A single `Measurement` which can be used as a parameter in other functions to track the status of the original request.

Throws:

See `sendRequest(MeasurementRequest[] measurementRequests, MeasuringPoint[] measuringPoints)` for details.

sendRequest

```
public java.util.ArrayList<Measurement>  
sendRequest(MeasurementRequest measurementRequest,  
MeasuringPoint[] measuringPoints) throws DipzoomException
```

Sends a request to the DipZoom server that the specified measurement request be performed by all listed measuring points. The total number of measurements to be performed will equal the number of measuring points passed in.

Note that this method only sends the requests; the user must utilize other functions such as `getTicketStatus()` and `requestResult()` to monitor the status and process the results of these requests.

Parameters:

`MeasurementRequest measurementRequest` – The request the user wishes to perform.

`MeasuringPoint[] measuringPoints` – The list of measuring points which should perform the measurement.

Returns: An `ArrayList<Measurement>` that contains all measurements that have been sent to the server by this function. This list can be used during future steps which track the progress on these measurements.

Throws:

See `sendRequest(MeasurementRequest\[\] measurementRequests, MeasuringPoint\[\] measuringPoints)` for details.

sendRequest

```
public java.util.ArrayList<Measurement>  
sendRequest(MeasurementRequest\[\] measurementRequests,  
MeasuringPoint measuringPoint) throws DipzoomException
```

Sends a request to the DipZoom server that the specified measurements be performed by a single measuring point. The total number of measurements to be performed will equal the number of measurement requests passed in.

Note that this method only sends the requests; the user must utilize other functions such as `getTicketStatus()` and `requestResult()` to monitor the status and process the results of these requests.

Parameters:

`MeasurementRequest[] measurementRequests` – The set of requests the user wishes to perform.

`MeasuringPoint measuringPoint` – The measuring point which should perform the measurements.

Returns: An `ArrayList<Measurement>` that contains all measurements that have been sent to the server by this function. This list can be used during future steps which track the progress on these measurements.

Throws:

See `sendRequest(MeasurementRequest\[\] measurementRequests, MeasuringPoint\[\] measuringPoints)` for details.

sendRequest2

```
public java.util.ArrayList<Measurement>  
sendRequest2(MeasurementRequest[] measurementRequests,  
MeasuringPoint[] measuringPoints) throws DipzoomException
```

Sends a request to the DipZoom server that each listed measuring point listed perform its corresponding measurement request (the first measuring point should execute the first request, the second point should execute the second request, and so on). The total number of measurements to be performed will equal the number of measuring points, which must also equal the number of measurement requests.

Note that this method only sends the requests; the user must utilize other functions such as `getTicketStatus()` and `requestResult()` to monitor the status and process the results of these requests.

Parameters:

`MeasurementRequest[] measurementRequests` – The set of requests the user wishes to perform.

`MeasuringPoint[] measuringPoints` – The set of measuring points which should each perform its corresponding measurement.

Returns: An `ArrayList<Measurement>` that contains all measurements that have been sent to the server by this function. This list can be used during future steps which track the progress on these measurements.

Throws:

See `sendRequest(MeasurementRequest[] measurementRequests, MeasuringPoint[] measuringPoints)` for details. In addition, the function will throw a `RequestException` in the case where the sizes of the two parameters passed in are not equal.

getTicketStatus

```
public java.util.ArrayList<Measurement> getTicketStatus()  
throws java.lang.Exception
```

Returns the list of outstanding requests associated with the client during the current session. Each request has a `TransactionStatus` which can be checked to determine if the request has been performed by the time of the function call.

Returns: An `ArrayList<Measurement>` that contains all outstanding requests (those whose results have either not returned or not been collected by `requestResult()`).

Throws:

MultipleLoginException – Thrown if another user attempts to log in using the same Client ID as the current user. A simple re-try is recommended. If problem persists, a new ID can be generated by deleting the login.xml file included in the package, then reattempting to log in; however, the user's JRE version must be updated to 1.5 or later.

requestResult

```
public Measurement requestResult(Measurement availableMeasurement)
throws java.lang.Exception
```

Requests the results for a single outstanding measurement. These results are returned by the function, but also recorded in two places: ticketfinished.xml, which contains records of all requested results, and an individual .txt file. Further information is detailed in the requestResult function entry for multiple measurements.

Parameters: Measurement availableMeasurement – the measurement whose result is desired.

Returns: The finished Measurement, whose result and resultFileName should contain the desired information.

requestResult

```
public java.util.ArrayList<Measurement>
requestResult(java.util.ArrayList<Measurement> availableMeasurements)
throws java.lang.Exception
```

Requests the results for an array list of Measurements. These results are returned by the function, but also recorded in two places: ticketfinished.xml, which contains records of all requested results, and an individual .txt file. These files are located relative to the program directory.

Note that this function will not return results for a measurement under either of two conditions:

- 1) The measurement's results have not been received by the server (this case should be handled by calling getTicketStatus() and only requesting measurements whose Transaction Status is equal to the constant RESULT_RECIEVED).
- 2) The requested measurement does not exist within the set of outstanding measurements as returned by getTicketStatus(). This case implies that either incorrect parameters are

being passed into the method, or the measurement in question has been dropped by the server.

Parameters:

ArrayList<Measurement> availableMeasurements – the list of measurements whose results are desired.

Returns:

An ArrayList<Measurement> which contains all finished measurements. Results for these measurements can be found using the getResult() and getResultFileName() methods on each finished measurement.

Throws:

LoginException – Thrown if the function is called before the user logs in (via the login() method).

MultipleLoginException – Thrown if another user attempts to log in using the same Client ID as the current user. A simple re-try is recommended. If problem persists, a new ID can be generated by deleting the login.xml file included in the package, then reattempting to log in; however, the user's JRE version must be updated to 1.5 or later.

DipzoomException – Usually thrown due to internal system errors (though certain error messages may imply user fault). Simple retry is suggested.

getFinishedMeasurements

```
public java.util.ArrayList<Measurement> getFinishedMeasurements()
```

Returns all finished measurements stored within ticketfinished.xml, which updates itself after every requestResult() call. Thus, the list returned by getFinishedMeasurements() should return the full list of all measurements that have been finished during the program's run history.

Note that this method does not actively extract finished measurements from the Dipzoom server, but rather reports all finished measurements that have been collected previous to the function call. In order to extract newly finished measurements from the server, the requestResult() function should be used.

Returns:

An ArrayList<Measurement> of finished measurements as stored within ticketfinished.xml.

clearFinishedTicketXML

```
public void clearFinishedTicketXML()
```

Clears the contents of finishedticket.xml. Future getFinishedMeasurements() calls will only return the finished results recorded since the last time the clear function was called.

Note that this method does not delete the actual .txt files which record each individual measurement, but merely removes any copies of these measurements which are contained in finishedticket.xml

DipzoomConstants Class

Description: The DipzoomConstants class contains information about the various constants utilized in the Dipzoom Client Library, with each constant given a name which implies its purpose within the library's methods. These constants can be used as a basis for comparison to ensure proper functioning of the library, or proper functioning of user-defined functions which use the library methods.

Relevant constants:

Constant name	Value	Significance
INVALID_TICKET	0	Transaction status code; see Measurement class documentation for more information.
REQUEST_RECEIVED	1	See above.
PICKED_BY_MP	2	See above.
RESULT_RECEIVED	4	See above.
FINISHED	8	See above.
Version	"2.0"	Designates version of DipzoomClientLibrary being used. 2.0 is the most recent version – please update package if using earlier version.

Methods:

statusCodeToString

```
public static java.lang.String  
statusCodeToString(java.lang.String code)
```

Given a transaction status code, returns the corresponding string representation for that code. Used specifically for converting a Measurement's transaction status.

Parameters: String code, the transaction status code.

Returns: String status, the represented status.

Measurement Class

Description: Each Measurement object represents an individual measurement that the Dipzoom server has received, and depending on the Transaction Status, is in various stages of execution.

Constructors:

```
Measurement()
```

Default constructor – sets no values.

```
Measurement(Hashtable<java.lang.String, java.lang.String> parameters)
```

Sets measurement values equal to values specified in the parameters Hashtable. Key values that can be set are as follows:

“type” – The type of measurement stored in this object (nslookup, dig, etc.).

“target” – The target host name or address of this measurement.

“number” – The number of measurements performed within this object.

“parameter” – Any special parameters related to this measurement.

Some measurement function entries have small grammatical errors: the get__() methods should be edited to read ‘after successfully submitting the request’ and ‘after successfully downloading the request.’

Methods:

toString

```
public java.lang.String toString()
```

Creates a string representation of the Measurement object, which contains all relevant values associated with the measurement.

Overrides: Inherited method `java.lang.Object.toString()`

Returns: A representative String for the Measurement.

equals

```
public boolean equals(Measurement m2)
```

Returns true if two Measurements objects refer to the same location, or if both have the same nonce (as assigned by the Dipzoom core).

Overrides: Inherited method `java.lang.Object.equals()`

Returns: A boolean value which is true if the two Measurements are equal, and false otherwise.

setParameters

```
public void setParameters (java.util.Hashtable<java.lang.String,  
java.lang.String> parameters)
```

Sets measurement values equal to values specified in the parameters Hashtable. Key values that can be set are as follows:

“type” – The type of measurement stored in this object (nslookup, dig, etc.).

“target” – The target host name or address of this measurement.

“number” – The number of measurements performed within this object.

“parameter” – Any special parameters related to this measurement.

getNonce

```
public java.lang.String getNonce()
```

Returns: String nonce, the unique request serial number, generated by the Dipzoom core after successfully submitting the request.

getId

```
public java.lang.String getId()
```

Returns: String Id, the MP id of the MP running this measurement, available after successfully submitting the request to the Dipzoom core.

getIp

```
public java.lang.String getIp()
```

Returns: String IP, the IP address of the MP running this measurement, available after successfully submitting the request to the Dipzoom core.

getHost

```
public java.lang.String getHost()
```

Returns: String Host, the host name of the MP running this measurement, available after successfully submitting the request to the Dipzoom Core.

getType

```
public java.lang.String getType()
```

Returns: String type, the type of measurement to be performed.

getTarget

```
public java.lang.String getTarget()
```

Returns: String target, the measurement target

getNumber

```
public java.lang.String getNumber()
```

Returns: String number, the number of measurement

getParameter

```
public java.lang.String getParameter()
```

Returns: String parameter, the parameters (if any) specified for this measurement.

getTransactionStatus

```
public int getTransactionStatus()
```

Returns: The status code of the current measurement, which can be interpreted using the `DipZoomConstants.CodetoString()` method. There are currently four possible statuses for a given measurement:

DipzoomConstants.REQUEST_RECIEVED (“requestreceived”) – The measurement has been received by the core server, but has not been sent to the proper MP for measurement.

DipzoomConstants.PICKED_BY_MP (“pickedbymp”) – The measurement has been received by the proper MP, but the results of the MP’s execution have not been received by the server.

DipzoomConstants.RESULTRECEIVED (“resultreceived”) – The measurement has been performed by the proper MP, and the results of the MP’s execution have been received by the server. The results of the measurement are ready to be downloaded by the client.

DipzoomConstants.FINISHED (“finished”) – The results from this measurement have been downloaded by the client; the transaction for this measurement is finished.

All states except “finished” are considered outstanding; all outstanding measurements associated with the client will be returned by the server when using the DipzoomClientLibrary.getTicketStatus() method.

getRequestedTimestamp

```
public long getRequestedTimestamp()
```

Returns: long timestamp, the time (in milliseconds since Jan 1, 1970) at which the result was received from the Dipzoom core. Available after successfully downloading the result from the core.

getResult

```
public java.lang.String getResult()
```

Returns: String result, the raw string of the measurement result, available after successfully downloading the result from the Dipzoom core.

getResultFilename

```
public java.lang.String getResultFilename()
```

Returns: String result filename, the name of the file that contains the results of the measurement. Available after successfully downloading the result from the Dipzoom core.

MeasurementRequest Class

Description: Each MeasurementRequest object represents an individual measurement request that has yet to be sent to the server. These requests are independent of measuring point, and must be assigned to an MP or a set of MPs using a variation of the sendRequest() function before they can be sent to the server.

Constructors:

```
MeasurementRequest()
```

The default constructor – initializes an object with no values.

```
MeasurementRequest(java.util.Hashtable<java.lang.String,  
java.lang.String> parameters)
```

```
MeasurementRequest(java.lang.String type, java.lang.String target)
```

```
MeasurementRequest(java.lang.String type, java.lang.String target,  
java.lang.String number)
```

```
MeasurementRequest(java.lang.String type, java.lang.String target,  
java.lang.String number, java.lang.String parameter)
```

Creates a MeasurementRequest object according to either the parameters passed in, or the contents of the parameters Hashtable. Possible parameters/key values are as follows:

String type – the type of measurement to be performed (ex: ‘ping,’ ‘nslookup’) A full list of currently supported measurement types can be found [here](#).

String target – the measurement target (ex: 129.22.151.190, http://cnn.com)

String number – the number of measurements you want (positive integer, default is one)

String parameter – optional; used to extend functionality of measurements to include command-line options. For example, defining parameter as the string ‘-querytype=MX’ for an nslookup measurement will enable that measurement’s filter-by-type option for MX entries.

Methods:

setParameters

```
public void setParameters(java.util.Hashtable<java.lang.String,  
java.lang.String> parameters)
```

Sets the parameters for the current MeasurementRequest object, based on the contents of the parameters HashTable. Valid key values for the parameters HashTable are as follows:

“type” – the type of measurement to be performed (ex: ‘ping,’ ‘nslookup’) A full list of currently supported measurement types can be found [here](#).

“target” – the measurement target (ex: 129.22.151.190, http://cnn.com)

“number” – the number of measurements you want (positive integer, default is one)

“parameter” – optional; used to extend functionality of measurements to include command-line options. For example, defining parameter as the string ‘-querytype=MX’ for an nslookup measurement will enable that measurement’s filter-by-type option for MX entries.

getType

```
public java.lang.String getType()
```

Returns: String type, the type of measurement being requested. Needs to be set by the user before submitting the request to the Dipzoom core (required).

getTarget

```
public java.lang.String getTarget()
```

Returns: String target, the target host name or address of this request. Needs to be set by the user before submitting the request to the Dipzoom core (required).

getNumber

```
public java.lang.String getNumber()
```

Returns: String number, the number of measurements to be performed in this request. Should be set by the user before submitting the request to the Dipzoom core (optional, default value 1).

getParameter

```
public java.lang.String getParameter()
```

Returns: String parameter, any parameters specified for this request. Should be set by the user before submitting the request to the Dipzoom core (optional, default empty).

Dipzoom Exceptions

The Dipzoom Client Library is designed to throw specific errors in certain cases where its methods cannot successfully complete. These errors are useful during development of programs which utilize the client library, as they can help determine the exact cause of error. Listed below are all relevant exceptions thrown by the Dipzoom Client Library, the conditions that cause them to be thrown, and a suggested resolution for each exception.

Title: DipZoomException

Description: The base class for all the exceptions for the DipZoom system. Errors of this type usually imply either an error on the side of the server, or during communications.

Title: GlobalTicketLimitException

Description: Errors of this type are thrown when the total number of outstanding requests to the DipZoom server exceeds the global limit, which is currently 10000. When facing this error, the user should wait a period of time before reattempting his/her program; this allows the server to become less congested.

Title: LoginException

Description: Errors of this type occur when the user either fails to login successfully, or attempts to call functions dependent on the DipZoom server without first logging in. When facing this error, the user should log in / re-login.

In the specific case of a LoginException of the type "AuthenticationFail on first login," the user should obtain a new version of the login.xml file, either from downloading a new package from the main site or deleting the old file and attempting to log in. Note that for the second method to work, the client must have his/her JRE updated to version 1.5 or later.

Title: MeasuringPointException

Description: Errors of this type occur when the server response to a getMeasuringPointList() function call cannot be properly interpreted. Such errors are uncommon.

Title: MultipleLoginException

Description: Errors of this type occur when another user attempts to log in using the same client ID or IP number as a user currently connected to the server, or the same user attempts to login multiple times. Errors of this type can normally be resolving by re-logging in. If issues persist, users can obtain new client IDs by downloading a new package from the main site or deleting the login.xml file and then re-attempting to log in. Note that for the second method to work, the client must have his/her JRE updated to version 1.5 or later.

Title: RequestException

Description: Errors of this type occur in two cases. The more common case occurs when the `sendRequest2()` function is called using two arrays of different size as parameters. This can be resolved by checking the size of each array parameter before calling the `sendRequest2()` function. The other case occurs when the server response to a `sendRequest()` function call cannot be properly interpreted. Such errors are not common.

Title: TicketLimitException

Description: Errors of this type occur when the number of outstanding requests made by the user exceed the per-client limit, which is currently 2000. When facing this error, the user should either download the results of any requests which have been completed by the DipZoom server, or wait for the server to clear any unwanted outstanding requests automatically.

Currently Supported Measurement Types

iplookup – performs DNS query on the target site and returns limited results. Result from measurement is the corresponding address / domain name of the provided target.

host – performs DNS query on the target site and returns limited results. Results from measurement include corresponding address / domain name of the provided target as well as mailbox entries for the domain.

tracpath – traces the path between a measuring point and the target site. Similar in functionality to traceroute. Results from measurement include total number of hops between sites, average time to reach each intermediary router, and names of intermediary routers.

nslookup – performs DNS query on the target site. Results from measurements include any and all discovered DNS records, such as corresponding alias IP addresses / domain names.

ping – transmits x packets to specified target site, where x is the number of messages specified. Results from measurements include packet loss as well as round-trip time for each packet.

dig – performs DNS query on the target site. Results from measurement include any and all discovered DNS records, such as corresponding alias IP addresses / domain names.

tcptraceroute – traces the path between a measuring point and the target site. Distinct from traceroute in that it utilizes TCP packets rather than the usual ICMP method. May not be supported by all measuring points, as tcptraceroute can only be performed by stations which have root access.

curl – retrieves information from a target URL by simulating the actions of an HTTP command (by default, GET). Results from measurement include information regarding the speed and completeness of download of the site information.

sr-test – currently unsupported

wget – performs download from the target site location; results from measurement include download rate, file length, and start/end time.

traceroute – traces the path between a measuring point and the target site through use of ICMP packets; results from measurement include total number of hops between sites, time to reach each intermediary router, and names of intermediary routers.

Possible User Errors (Troubleshooting):

1) *When I run my program, I get the error "AuthenticationFail on first login."*

The login.xml file is corrupted and contains invalid information for a clientID / encryption key pair. This can be resolved by simply deleting the login.xml file; the next login attempt will initialize a new, valid login.xml file. Note that the client must have his/her JRE updated to version 1.5 or later.

2) *The list I receive when calling getFinishedMeasurements() does not contain the measurements I sent out.*

getFinishedMeasurements calls from ticketfinished.xml, which is only updated during calls to the requestResult() function. In other words, for a measurement to be recorded as finished, its results must first be requested via requestResult().

Alternately, the ClearFinishedTicketXML() function clears the contents of ticketfinished.xml. As such, if getFinishedMeasurements does not contain messages that were expected/received a long time ago, this may be due to a ClearFinishedTicketXML() call within your code.

3) *When I run my program for an extended period of time, eventually it returns a memory overflow error. What has happened?*

The likely cause for this error is the finishedticket.xml file, which is updated after each call of requestResult(). If the file is not cleared regularly, it will reach an unmanageable size; as such, the ClearFinishedTicketXML() function should be called periodically.

4) *When I request a large amount of measurements, sometimes my program hangs indefinitely; then, the program suddenly reports less outstanding requests than I am expecting back. I didn't get the results for those reports back. What happened?*

In some cases, specifically when a large amount of requests are made to the same target, the DipZoom server may decide to purposely ignore those requests. This is to avoid exploitation of the server as the head of a DDoS (Distributed Denial of Service) attack. Maintaining a status for dropped requests is undesirable for the server; as such, tracking dropped/successful requests is a responsibility of the user application. This responsibility can be carried out through intelligent use of the getTicketStatus() function.

5) *There is no Measurement type for curl/ping/traceroute. How do I get the results for these measurements?*

The results for these measurements can be accessed using getResult() method of each individual Measurement. To extract useful information quickly from these results, you should implement your own parsing methods after studying the common format of each type of measurement.