F.3 NeuArchitect API Reference

NeuArchitect v3.0

6 Armstrong Road, Shelton, Connecticut 06484 USA
www.neuvis.com

Contents

12 Interface Request

13 Interface Response

14 Interface Server

15 Interface Session

1 Context

The class, Context extends java.lang.Object and is a public abstract class. It defines a platform neutral object representing the NeuArchitect runtime context.

CloseDBConnection

Signature:

public void **CloseDBConnection**

Use/Purpose:

Deprecated. Closes the JDBC connection and pops it from the stack

Considerations:

Replaced by **CloseDBConnections**

CloseDBConnections

Signature:

public void **CloseDBConnections**

Use/Purpose:

Closes all JDBC connections

Considerations:

Closes all JDBC connections and frees associated resources. Please use with care.

ExceptionDump

Signature:

private java.lang.String **ExceptionDump**(java.lang.Exception ex)

Use/Purpose:

Returns stack dump for the exception as java String object

Considerations:

None

getConnectionProperties

Signature:

public java.util.Properties **getConnectionProperties**()

Use/Purpose:

Deprecated. Returns the properties specified for the last JDBC connection opened.

This method will return the properties specified in the last call to **OpenDBConnection().** It will not return the properties for any connection pushed onto the stack by **PushDBConnection().**

Considerations:

Replaced by getConnectionProperties(String descriptor)

getConnectionProperties

Signature:

public java.util.Properties **getConnectionProperties**(java.lang.String descriptor)

Use/Purpose:

Returns the properties for JDBC connection opened on the specified descriptor

This method will return the properties specified in the last call to **OpenDBConnection().**

Considerations:

None

GetDBConnection

Signature:

public java.sql.Connection **getDBConnection**() throws java.util.EmptyStackException

Use/Purpose:

Deprecated. *Returns the Connection object at the top of the stack without removing it from the stack*

Considerations:

Replaced by **getDBConnection(String descriptor)**

GetDBConnection

Signature:

public java.sql.Connection **getDBConnection**(java.lang.String descriptor) throws java.lang.Exception

Use/Purpose:

Returns the Connection object for the specified descriptor

Considerations:

Parameter **descriptor** - A String representing the database descriptor

Returns: the Connection object

Throws - **NoSuchElementException** if the descriptor was not loaded

GetLastError

Signature:

public NeuError **getLastError()** throws java.util.EmptyStackException

Use/Purpose:

Returns the last error the application generated. The error is expressed in the form of a NeuError object.

Considerations:

Please refer to NeuError documentation for it's definition and usage.

getNeuVisUtil

Signature:

public NeuVisUtilX **getNeuVisUtil()**

Use/Purpose:

Returns the NeuVisUtilX object.

Considerations:

None

getRequest

Signature:

public **Request getRequest()**

Use/Purpose:

Returns a handle to the Request object.

Considerations:

Please refer to the Request object API section for more details on this object.

getResponse

Signature:

public **Response getResponse()**

Use/Purpose:

Returns a handle to the Response object.

Considerations:

Please refer to the Response section for a definition of this object.

getSecurity

Signature:

public com.neuvis.security.NeuSecurityAdmin getSecurity()

Use/Purpose:

Returns a handle to the NeuSecurity object.

Considerations:

Please refer to the NeuSecurity section for a definition of this object.

getServer

Signature:

public Server **getServer()**

Use/Purpose:

Returns a handle to the Server object.

Considerations:

Please refer to the Server section for a definition of this object.

getSession

Signature:

public Session **getSession()**

Use/Purpose:

Returns a handle to the Session object.

Considerations:

Please refer to the Session section for a definition of this object.

InTransaction

Signature:

public boolean **InTransaction()**

Use/Purpose:

Deprecated! Please use getConnectionProperties(String descriptor) instead!

LoadDBDescriptors

Signature:

public void **LoadDBDescriptors**(java.lang.String fileName)

throws java.io.IOException,

java.util.NoSuchElementException

Use/Purpose:

Loads the Database Descriptors defined in the specified file

Considerations:

The specified file must exist somewhere along the Classpath.

OpenDBConnection

Signature:

public void **OpenDBConnection**(java.lang.String descriptor)
throws java.lang.LinkageError,
java.lang.ClassNotFoundException,
java.lang.InstantiationException,
java.lang.IllegalAccessException,
java.lang.ExceptionInInitializerError,
java.sql.SQLException,
java.util.NoSuchElementException

Use/Purpose:

Opens a JDBC connection based on the specified database descriptor

Considerations:

Parameters: descriptor - A String representing the database descriptor

OpenDBConnection

Signature:

public void OpenDBConnection(java.lang.String descriptor)
throws java.lang.LinkageError,
java.lang.ClassNotFoundException,
java.lang.InstantiationException,
java.lang.IllegalAccessException,
java.lang.ExceptionInInitializerError,
java.sql.SQLException,
java.util.NoSuchElementException

Use/Purpose:

Opens a JDBC connection based on the specified database descriptor

Considerations:

Parameters: descriptor - A String representing the database descriptor

OpenDBConnection

Signature:

public void **OpenDBConnection**(java.lang.String driver,

java.lang.String url,
java.util.Properties info)
throws java.lang.LinkageError,
java.lang.ClassNotFoundException,
java.lang.InstantiationException,
java.lang.IllegalAccessException,
java.lang.ExceptionInInitializerError,
java.sql.SQLException

Use/Purpose:

Deprecated. Opens a JDBC connection to the given database url and pushes it onto the stack

Considerations:

Replaced by **OpenDBConnection**(String descriptor)

Parameters:

driver - A String representing the JDBC driver

url - A String representing a database url of the form jdbc:subprotocol:subname

info - A Properties object a list of arbitrary string tag/value pairs as connection arguments; normally at least a "user" and "password" property should be included

OpenDBConnection

Signature:

public void OpenDBConnection(java.lang.String driver,
java.lang.String url,
java.lang.String user,
java.lang.String password)
throws java.lang.LinkageError,
java.lang.ClassNotFoundException,
java.lang.InstantiationException,
java.lang.IllegalAccessException,
java.lang.ExceptionInInitializerError,
java.sql.SQLException

Use/Purpose:

Deprecated. Opens a JDBC connection to the given database url and pushes it onto the stack

Considerations:

Replaced by **OpenDBConnection(String descriptor)**

Parameters:

driver - A String representing the JDBC driver

url - A String representing a database url of the form jdbc:subprotocol:subname

user - A String representing the username

password - A String representing the user's password

OpenDBConnections

Signature:

public void **OpenDBConnections()**

throws java.lang.LinkageError,
java.lang.ClassNotFoundException,
java.lang.InstantiationException,
java.lang.IllegalAccessException,
java.lang.ExceptionInInitializerError,
java.sql.SQLException

Use/Purpose:

Opens JDBC connections for all database descriptors loaded

Considerations:

Database descriptors must be loaded before this method is called.

PopDBConnection

Signature:

public java.sql.Connection **PopDBConnection()**

throws java.util.EmptyStackException

Use/Purpose:

Deprecated.

Considerations:

Not replaced

ProcessException

Signature:

public void ProcessException(java.lang.Exception ex)

Use/Purpose:

Dumps stack of java exception into response object

Considerations:

Ability to write stack dump into log file will be provide later. This method dumps the stack into the response object.

PushDBConnection

Signature:

public void PushDBConnection(java.sql.Connection conn)

Use/Purpose:

Deprecated. Pushes the Connection onto the top of the stack

Considerations:

None

putLastError

Signature:

public void putLastError(NeuError err)

Use/Purpose:

Places an error object (NeuError) on the top of the error list.

Considerations:

Please refer to the NeuError section for more details on this object.

RollbackTransaction

Signature:

public void RollbackTransaction()

Use/Purpose:

Rolls back a transaction. Must be preceded somewhere by a BeginTransaction. For COM, this executes a SetAbort().

Considerations:

Forces a roll back of all elements of the transaction.

2 EnumRecord

The class, EnumRecord extends java.lang.Object. An EnumRecord object is used to hold a single record from an enumeration table. An enumeration table is created for each attribute in an entity that is defined as type Enum.

EnumRecord

Signature:

public EnumRecord(int ID, String Name, String ImageName)

Use/Purpose:

This method is used to store an single enumeration record with an associated image file. The record is stored in the database and contains three(3) fields: ID, Name and ImageName. The ID field maps to the ID parameter and is an integer used to uniquely identify the record. The Name field maps to the Name parameter and contains the text entered for the particular enumeration. The ImageName field maps to the ImageName parameter and contains the filename for the image associated with the enumeration.

Considerations:

EnumRecord objects are typically added to an ObjectSet, which is a general purpose container.

Method Examples:

// This example creates two EnumRecord objects and then adds them to an ObjectSet container.

EnumRecord color1 = new **EnumRecord(1, “Red”, “red.png”)**;

EnumRecord color2 = new **EnumRecord(2, “Blue”, “blue.png”)**;

ObjectSet colors = new ObjectSet();

colors.Add(color1);

colors.Add(color2);

3 NeuError

The class, NeuError extends java.lang.Object. This class represents a NeuError Object. The NeuError Object can be used to manage any errors that the application needs to handle. It consists of two data members: an integer to hold the error code and a String to hold the associated error message text. It also provides a means of segregating database errors by using a dbtype identifier.

findErrorMessage

Signature:

public java.lang.String findErrorMessage(int errorCode, int dbType)

Use/Purpose:

This method finds the error message text that is associated with the error code that is passed into this function in the parameter errorCode. The corresponding error message is returned as a string.

Considerations:

If the error code is valid, this function will return the associated error message text. The dbtype can be any of the following constants:

SQLSERVER

ORACLE

INFORMIX

SYBASE

IMS

Method Examples:

NeuError errormsg = new NeuError();

errormsg.setErrorCode(1);

String errormsgtxt = errormsg.**findErrorMessage**(1, NeuError.ORACLE);
errormsg.setErrorMessage(errormsgtxt);

getErrorCode

Signature:

public int getErrorCode()

Use/Purpose:

This method is used to retrieve the currently set error code as an integer value.

Considerations:

None

Method Examples:

NeuError errormsg = new NeuError();

errormsg.setErrorCode(1);

int errorCode = errormsg.**getErrorCode()**;//get current error code

getErrorMessage

Signature:

public java.lang.String getErrorMessage()

Use/Purpose:

This method is used to retrieve the currently set error message as a text string.

Considerations:

None

Method Examples:

NeuError errormsg = new NeuError();

errormsg.setErrorCode(1);

errormsg.setErrorMessage("A really BAD error has occurred!");

String errormsgtxt = errormsg.**getErrorMessage()**;

setErrorCode

Signature:

public void setErrorCode(int code)

Use/Purpose:

This method is used to set the error code to the value assigned to the parameter 'code'.

Considerations:

Needs to be a valid error code.

Method Examples:

NeuError errormsg = new NeuError();

errormsg.**setErrorCode(1)**; //Set the error code to 1

setErrorMessage

Signature:

public void setErrorMessage(java.lang.String message)

Use/Purpose:

This method is used to set the error message text associated with the error code set using setErrorCode.

Considerations:

Must be a valid qoute-deliminted string.

Method Examples:

NeuError errormsg = new NeuError();

errormsg.setErrorCode(1);

errormsg.**setErrorMessage(“A really BAD error has occurred!”)**;

4 NeuErrorList

The class, NeuErrorList extends java.lang.Object. This class represents a list of NeuError objects. NeuError objects can be pushed onto the top of the list or popped off the top of the list.

PushLastError

Signature:

public void PushLastError()

Use/Purpose:

This method is used to push a NeuError object onto the top of the list.

Considerations:

The list works like any stack construct. An object is pushed onto the top of the stack. When PopLastError is called, the object on the top of the stack will be returned and removed from the list.

Method Examples:

NeuError err = new NeuError(); //Create an error object

err.setErrorCode(10); //set the error code

err.setErrorMessage("Fatal Error. Resource not found");

NeuErrorList errorList = new NeuErrorList(); //create error list

errorList.PushLastError(err); //place error into list

PopLastError

Signature:

public NeuError PopLastError() throws java.util.EmptyStackException

Use/Purpose:

This method is used to pop a NeuError object of the top of the stack.

Considerations:

If there are no objects in the list when this method is called, a java.util.EmptyStackException is thrown. The list works like any stack construct.

Method Examples:

```
NeuError err = new NeuError(); //Create an error object
err.setErrorCode(10); //set the error code
err.setErrorMessage("Fatal Error. Resource not found");
NeuErrorList errorList = new NeuErrorList(); //create error list
errorList.PushLastError(err); //place error into list
// Catch proper exception for empty list
NeuError myError;
```

```
try
{
   myError = errorList.PopLastError();
}
catch(java.util.EmptyStackException e)
{
   String test = "Oh No!, Nothing left in the list";
}
```

5 NeuVisBlob

The class, NeuVisBlob extends java.lang.Object. The NeuVisBlob class is used to facilitate the upload and download of binary large objects (BLOB).

close

Signature:

public void close()

Use/Purpose:

To close the InputStream represented as iStream member variable in the Object. NeuVisBlob is an object that represents a file as a blob. Close() closes the iStream pointer. Always call the Close() function to close the InputStream pointer after you are done.

It will be used in the Blob operation (trying to upload a file as a blob in the database)

Considerations:

None.

Method Examples:

Sample code from the Insert function generated if you are uploading a blob.

```
// Resume is a blob type variable
// call the LoadStream to Create the input sream pointer from the file name.
// after executing the SQL Query call close to close the Input Sream
pointer.
if ((this.Resume != null) && (this.Resume.LoadStream()) )
{
   FieldList.append( " , Resume " ) ;
   ValueList.append( " ,? " ) ;
}

SqlStmt += FieldList + " ) " + ValueList + " ) " ;
pstmt  = con.prepareStatement(SqlStmt) ;
pstmt.setInt(index++,this.Customer_TS) ;
pstmt.setString(index++,this.Resume_Desc) ;

if ((this.Resume != null) && (this.Resume.LoadStream()) )
   pstmt.setBinaryStream(index++,this.Resume.getStream(),this.Resume.getLeng
th()) ;

pstmt.execute() ;
// call close to close the InputStream

if (this.Resume != null)
{
```

```
   this.Resume.close() ;


}

pstmt.close() ;
```

getFileName

Signature:

public java.lang.String getFileName()

Use/Purpose:

getFileName returns the FileName to which the NeuVisBlob is associated. Whe the file is uploaded , a file is created on the server and getFileName() returns the File name on the server.

Considerations:

None

Method Examples:

```
// Resume is the NeuVisBlob object
String sFileName = this.Resume.getFileName();
```

getLength

Signature:

public int getLength()

Use/Purpose:

getLength() returns the length of the file.

Considerations:

To get the length first call the LoadStream to load the stream other wise it will return 0.

Method Examples:

```
if ((this.Resume != null) && (this.Resume.LoadStream()) )
{
   FieldList.append( " , Resume " ) ;
   ValueList.append( " ,? " ) ;
}

SqlStmt += FieldList + " ) " + ValueList + " ) " ;
pstmt  = con.prepareStatement(SqlStmt) ;
// getLength() is passed when setting the BinaryStream
if ((this.Resume != null) && (this.Resume.LoadStream()) )
```

```
   pstmt.setBinaryStream(index++,this.Resume.getStream(),this.Resume.getLeng
th()) ;
```

GetStatus

Signature:

public boolean GetStatus()

Use/Purpose:

getStatus() returns the Status of the NeuVisBlob whether it is loaded or not.

When a call to LoadStream() is made the InputStream is attached to the FileInputStream and on success the Status is set to true.

Considerations:

None

Method Examples:

getStream

Signature:

public java.io.InputStream getStream()

Use/Purpose:

getStream() returns the InputStream. If the file is already attached to the InputStream then it returns the Input Stream other wise it opens the File and returns the InputStream.

Considerations:

None

Method Examples:

```
if ((this.Resume != null) && (this.Resume.LoadStream()) )
{
   FieldList.append( " , Resume " ) ;
   ValueList.append( " ,? " ) ;
}

SqlStmt += FieldList + " ) " + ValueList + " ) " ;
pstmt  = con.prepareStatement(SqlStmt) ;
// getStream() is passed when setting the BinaryStream
if ((this.Resume != null) && (this.Resume.LoadStream()) )
   pstmt.setBinaryStream(index++,this.Resume.getStream(),this.Resume.getLeng
th()) ;
```

LoadStream

Signature:

public boolean LoadStream()

Use/Purpose:

To load the InputStream with the file name associated with the NeuVisBlob Object.

Returns true if the InputStream is loaded successfully otherwise returns false.

Considerations:

None.

Method Examples:

```
// before calling the setBinaryStream () check whether the InputStream is
available or not
// if not than don't call the LoadStream()
if ((this.Resume != null) && (this.Resume.LoadStream()) )
   pstmt.setBinaryStream(index++,this.Resume.getStream(),this.Resume.getLeng
th()) ;
```

setFileName

Signature:

public void setFileName(java.lang.String FileName)

Use/Purpose:

setFileName() sets the FileName for the NeuVisBlob object.

Considerations:

None

Method Examples:

setLength

Signature:

public void setLength(int lLength)

Use/Purpose:

Sets the length of the NeuVisBlob Object.

Considerations:

None

Method Examples:

setStream

Signature:

public void setStream(java.io.InputStream iStream)

Use/Purpose:

setStream sets the inputStream from external source.

Considerations:

Method Examples:

6 NeuVisField

The class, NeuVisField extends java.lang.Object and implements java.io.Serializable.It is a Value Class Of NeuVis Generated Object . This class is used mostly as variant.

getBinaryStream

Signature:

public NeuVisBlob getBinaryStream()

Use/Purpose:

Returns NeuVisBlob object .use this function if Field is blob

Considerations:

NeuvisField must be set with setBinaryStream

Method Examples:

```
Class Product
{
public void customMethod()
{
   NeuVisBlob nb = this.productImage.getBinaryStream() ;
   …
}
}
```

getBlob

Signature:

public NeuVisBlob getBlob()

Use/Purpose:

Returns NeuVisBlob object .use this function if Field is blob

Considerations:

NeuvisField must be set with setBinaryStream

Method Examples:

```
Class Product
{
public void customMethod()
{
   NeuVisBlob nb = this.productImage.getBinaryStream() ;
   …
}
}
```

getBoolean

Signature:

public boolean getBoolean()

Use/Purpose:

Returns true or false associated with this field

Considerations:

None

Method Examples:

```
NeuVisField nf = new NeuVisField(ctx.getRequest().getParameter("IsList") ;
If(nf.getBoolean())
{
…
}
```

getDate

Signature:

public java.sql.Date getDate()

throws java.lang.Exception

Use/Purpose:

Get date Value associated with this field

Considerations:

none

Method Examples:

```
NeuVisField nf = new NeuVisField(ctx.getRequest().getParameter("ListDate") ;
java.sql.Date d = nf.getDate() ;
```

getDate

Signature:

public java.sql.Date getDate(int ai_format)

throws java.lang.Exception

Use/Purpose:

Get date Value associated with this field

Format 1 : returns date in GMT Format

Format 2 : returns date in us format

Considerations:

None

Method Examples:

```
NeuVisField nf = new NeuVisField(ctx.getRequest().getParameter("ListDate") ;
java.sql.Date d = nf.getDate(2) ;
```

getDouble

Signature:

public double getDouble() throws java.lang.Exception

Use/Purpose:

Get double Value associated with this field

Considerations:

Value associated with this field should be double

Method Examples:

```
NeuVisField nf = new NeuVisField(ctx.getRequest().getParameter("price") ;
Double d = nf.getDouble() ;
```

getFloat

Signature:

public float getFloat() throws java.lang.Exception

Use/Purpose:

Get float Value associated with this field

Considerations:

Value associated with this field should be double

Method Examples:

```
NeuVisField nf = new NeuVisField(ctx.getRequest().getParameter("distance"))
;
float f = nf.getFloat() ;
```

getInt

Signature:

public int getInt()

Use/Purpose:

Get integer value associated with this field

Considerations:

Value hold by field should be integer

Method Examples:

```
NeuVisField nf = new NeuVisField(ctx.getRequest().getParameter("age")) ;
Int age = nf.getInt() ;
```

getJDBCDate

Signature:

public java.sql.Date getJDBCDate()

Use/Purpose:

Get date value associated with this field

Considerations:

Value hold by field should be date

Method Examples:

```
NeuVisField nf = new NeuVisField(ctx.getRequest().getParameter("birthdate"))
;
Java.sql.Date  bDate = nf.getJDBCDate() ;
```

getJDBCTime

Signature:

public java.sql.Time getJDBCTime()

Use/Purpose:

Get time value associated with this field

Considerations:

Value hold by field should be time

Method Examples:

```
NeuVisField nf = new NeuVisField(ctx.getRequest().getParameter("showtime"))
;
Java.sql.Time  sTime = nf.getJDBCTime() ;
```

getJDBCTimestamp

Signature:

public java.sql.Timestamp getJDBCTimestamp()

Use/Purpose:

Get timestamp value associated with this field

Considerations:

Value hold by field should be timestamp

Method Examples:

```
NeuVisField nf = new NeuVisField(ctx.getRequest().getParameter("shiptime"))
;
Java.sql.Timestamp shipTime = nf.getJDBCTimestamp() ;
```

getShort

Signature:

public short getShort() throws java.lang.Exception

Use/Purpose:

Get short value associated with this field

Considerations:

Value hold by field should be short

Method Examples:

```
NeuVisField nf = new
NeuVisField(ctx.getRequest().getParameter("purchasequantiy")) ;
short qty = nf.getShort() ;
```

getString

Signature:

public java.lang.String getString()

Use/Purpose:

Get String value associated with this field

Considerations:

Value hold by field should be String

Method Examples:

```
NeuVisField nf = new
NeuVisField(ctx.getRequest().getParameter("employeename")) ;
String eName = nf.getString() ;
```

getTime

Signature:

public java.sql.Time getTime() throws java.lang.Exception

Use/Purpose:

Deprecated.

Considerations:

Replaced by NeuVisUtil.getTime(String sDateTime ,String format);

Method Examples:

getTime

Signature:

public java.sql.Time getTime(int ai_format) throws java.lang.Exception

Use/Purpose:

Get time value associated with this field

Ai_format 1 : JDBC Time Format

Considerations:

Value hold by field should be Time

Method Examples:

```
NeuVisField nf = new NeuVisField(ctx.getRequest().getParameter("showtime"))
;
Java.sql.Time  sTime = nf.getTime() ;
```

getTimestamp

Signature:

public java.sql.Timestamp getTimestamp() throws java.lang.Exception

Use/Purpose:

Deprecated.

Considerations:

Replaced by NeuVisUtil.getTimestamp(String sDateTime ,String format);

getTimestamp

Signature:

public java.sql.Timestamp getTimestamp(int ai_format) throws java.lang.Exception

Use/Purpose:

Get timestamp value associated with this field

Ai_format 1 : JDBC Timestamp Format

Considerations:

Value hold by field should be Timestamp

Method Examples:

```
NeuVisField nf = new NeuVisField(ctx.getRequest().getParameter("showtime"))
;
Java.sql.Time stamp sTime = nf.getTime() ;
```

getType

Signature:

public int getType()

Use/Purpose:

Returns type of value hold by NeuVisFieldd

Return value : Basic Type : integer value

NeuVisField.NF_INT : integer:1

NeuVisField.NF_SHORT :short:2

NeuVisField.NF_DOUBLE :double:3

NeuVisField.NF_FLOAT :float:4

NeuVisField.NF_BOOLEAN :Boolean:5

NeuVisField.NF_DATE :date:6

NeuVisField.NF_TIME :time:7

NeuVisField.NF_TIMESTAMP :timestamp:8

NeuVisField.NF_STRING :string:9

NeuVisField.NF_BLOB :blob:10

NeuVisField.NF_BIGDECIMAL :bigdecimal:11

Considerations:

None

Method Examples:

```
NeuVisField nf = new NeuVisField() ;
Nf.setInt(5) ;
…
…
if(nf.getType() == nf.NF_INT)
{
   int I = nf.getInt() ;
…
}
else if(nf.getType() == nf.NF_STRING)
{
String iStr = nf.getString() ;
}
```

setBinaryStream

Signature:

public void setBinaryStream(NeuVisBlob in)

Use/Purpose:

Set Value of this NeuVisField as Blob

Considerations:

None

Method Examples:

```
NeuVisBlob nBlob = new NeuVisBlob() ;
NBlob.setFileName("..\blue.gif") ;
NeuVisField nf = new NeuVisField() ;
Nf.setBinaryStream(nBlob) ;
```

setBoolean

Signature:

public void setBoolean(boolean ab)

Use/Purpose:

Set Value of this NeuVisField as boolean

Considerations:

None

Method Examples:

```
NeuVisField nf = new NeuVisField() ;
Nf.setBoolean(true) ;
```

setDate

Signature:

public void setDate(java.sql.Date ad)

Use/Purpose:

Set Value of this NeuVisField as Date

Considerations:

None

Method Examples:

```
Java.sql.Date date = ctx.NeuVisUtil. GetCurrentDate();
NeuVisField nf = new NeuVisField() ;
Nf.setDate(date) ;
```

setDouble

Signature:

public void setDouble(double adb)

Use/Purpose:

Set Value of this NeuVisField as Double

Considerations:

none

Method Examples:

```
NeuVisField nf = new NeuVisField() ;
Nf.setDouble(5.5) ;
```

setFloat

Signature:

public void setFloat(float af)

Use/Purpose:

Set Value of this NeuVisField as Float

Considerations:

None

Method Examples:

```
Float f ;
F = 5.005 ;
NeuVisField nf = new NeuVisField() ;
Nf.setFloat(f) ;
```

setInt

Signature:

public void setInt(int ai)

Use/Purpose:

Set Value of this NeuVisField as int

Considerations:

None

Method Examples:

```
Int I = 3 ;
NeuVisField nf = new NeuVisField() ;
Nf.setInt(i) ;
```

setShort

Signature:

public void setShort(short as)

Use/Purpose:

Set Value of this NeuVisField as Short

Considerations:

None

Method Examples:

```
Short s = 3 ;
NeuVisField nf = new NeuVisField() ;
Nf.setShort(s) ;
```

setString

Signature:

public void setString(java.lang.String as)

Use/Purpose:

Set Value of this NeuVisField as String

Considerations:

None

Method Examples:

```
NeuVisField nf = new NeuVisField() ;
Nf.setString(“Devang Parikh”) ;
```

setTimestamp

Signature:

public void setTimestamp(java.sql.Timestamp at)

Use/Purpose:

Set Value of this NeuVisField as TimeStamp

Considerations:

None

Method Examples:

```
Java.sql.TimeStamp ts = ctx.NeuVisUtil.getCurrentTimeStamp() ;
NeuVisField nf = new NeuVisField() ;
Nf.setTimestamp(ts) ;
```

7 NeuVisObject

The class, NeuVisObject extends java.lang.Object and is the base class of all generated business objects. All generated Business Objects are derived from NeuVisObject**.**

ExecuteMethod

Signature:

public boolean ExecuteMethod(java.lang.String MethodName) throws java.lang.Exception

Use/Purpose:

Use this method to execute any Boolean method

Considerations:

Methos return type mut be boolean

Method Examples:

```
If(this.Products.GetLength() > 0)
{
   if(this.GetAttribute(“Products”,0).ExecuteMethod(“procBuy”))
{
…
}
}
```

GetAttribute

Signature:

public NeuVisObject GetAttribute(java.lang.String AttrName, int index)

throws java.lang.Exception

Use/Purpose:

Use this function to retrieve NeuVisObject in any contained ObjectSet at specified index

Considerations:

None

Method Examples:

```
If(this.Products.GetLength() > 0)
{
   if(this.GetAttribute(“Products”,0).ExecuteMethod(“procBuy”))
{
…
}
}
```

GetAttrType

Signature:

public int GetAttrType(java.lang.String AttrName)

throws java.lang.Exception

Use/Purpose:

This method returns type of given attribute name

returns

0 : for all basic attributes

1: for referenced and single ObjectSet

2:for multi ObjectSet

Considerations:

None

Method Examples:

```
If(this.GetAttrType("Product") == 1)
{
this.Product.procBuy() ;
}
else if(this.GetAttrType("Product") == 2)
{
int len = this.Product.GetLength() ;
}
```

GetPageAttributeValue

Signature:

public NeuVisField GetPageAttributeValue(java.lang.String FieldName)

throws java.lang.Exception

Use/Purpose:

Call this function to query actual value enterd in client browser for basic projected attribute in page.

Considerations:

If attribute is not projected in page in editable form function will return null

Method Examples:

```
String quantity =  this.GetPageAttributeValue("quantityToOrder")
.getString() ;
```

SetAttributeValue

Signature:

public void SetAttributeValue(java.lang.String FieldName, NeuVisField FieldValue)

throws java.lang.Exception

Use/Purpose:

Use this function to set basic attribute value of this object

FieldName : Name of Attribute To set

FieldValue : Value of Attribute

Considerations:

None

Method Examples:

```
NeuVisField nf = new NeuVisField();
Nf.setInt(5) ;
This.SetAttributeValue("quantityToOrder",nf) ;
This.Update() ;
```

SetAttributeValue

Signature:

public void SetAttributeValue(java.lang.String FieldName, NeuVisField FieldValue, java.lang.String NeuFormatStr)

throws java.lang.Exception

Use/Purpose:

Use this function to set basic attribute value of this object

FieldName : Name of Attribute To set

FieldValue : Value of Attribute

NeuFormatStr : Format to use for parsing

Considerations:

Use this function to set date and datetime attribute values

Method Examples:

```
NeuVisField nf = new NeuVisField();
Nf.setDate(ctx.NeuVisUtil. GetCurrentDate()) ;
This.SetAttributeValue("shipdate",nf,"mm/dd/yyyy") ;
This.Update() ;
```

8 NeuVisUtilX

The class, NeuVisUtilX extends java.lang.Object and implements java.io.Serializable. NeuVisUtilX provides the applications developer with some standard utility functions to format dates, time, strings as well as other functionality.

ConvertStringToDate

Signature:

public java.util.Date ConvertStringToDate(java.lang.String dateStr, java.lang.String fmtStr)

Use/Purpose:

Convert String into java.util.Date object based on the format string provided in the fmtStr parameter.

Considerations:

The format string must comply with the definitions as described below.

Parameters:

dateStr - A String representing the date

fmtStr - A String representing parsing format. To specify the parsing format, use a time pattern string. In this pattern, all ASCII letters are reserved as pattern letters, which are defined as the following:

Symbol	Meaning	Presentation	Example
G	era designator	(Text)	AD
y	year	(Number)	1996
M	month in year	(Text & Number)	July & 07
d	day in month	(Number)	10
h	hour in am/pm (1~12)	(Number)	12
H	hour in day (0~23)	(Number)	0
m	minute in hour	(Number)	30
s	second in minute	(Number)	55
S	millisecond	(Number)	978
E	day in week	(Text)	Tuesday
D	day in year	(Number)	189
F	day of week in month	(Number)	2 (2nd Wed in July)
w	week in year	(Number)	27
W	week in month	(Number)	2
a	am/pm marker	(Text)	PM
k	hour in day (1~24)	(Number)	24
K	hour in am/pm (0~11)	(Number)	0

Symbol	Meaning	Presentation	Example
z	time zone	(Text)	Pacific Standard Time
'	escape for text		
''	single quote	'	

(**Text**): 4 or more pattern letters--use full form, < 4--use short or abbreviated form if one exists.

(**Number**): the minimum number of digits. Shorter numbers are zero-padded to this amount. Year is handled specially; that is, if the count of 'y' is 2, the Year will be truncated to 2 digits.

(**Text & Number**): 3 or over, use text, otherwise use number.

Any characters in the pattern that are not in the ranges of ['a'..'z'] and ['A'..'Z'] will be treated as quoted text. For instance, characters like ':', '.', ' ', '#' and '@' will appear in the resulting time text even they are not embraced within single quotes.

Examples :

Format Pattern _____ Date String

"yyyy.MM.dd G 'at' hh:mm:ss z"-->> 1996.07.10 AD at 15:08:56 PDT

"EEE, MMM d, "yy"---------->> Wed, July 10, '96

"h:mm a"---------->> 12:08 PM

"hh 'o'"'clock' a, zzzz"---------->> 12 o'clock PM, Pacific Daylight Time

"K:mm a, z"---------->> 0:00 PM, PST

"yyyyy.MMMMM.dd GGG hh:mm aaa"->> 1996.July.10 AD 12:08 PM

Returns: the java.util.Date object or null if dateStr could not be converted

Decrypt

Signature:

public java.lang.String Decrypt(java.lang.String s_String) throws java.lang.Exception

Use/Purpose:

Decrypts a string that was encrypted using the Encrypt(String) method. This method uses the session key to decrypt the string.

Considerations:

Can only be used on a string that was encrypted using the Encrypt(String) method. The encrypt/decrypt algorithm used is called Blowfish. Blowfish is a symmetric block cipher that can be used as a drop-in replacement for DES or IDEA. It takes a variable-length key, from 32 bits to 448 bits, making it ideal for both domestic and exportable use.

Decrypt

Signature:

public java.lang.String Decrypt(java.lang.String s_String, java.lang.String sKey)

throws java.lang.Exception

Use/Purpose:

Decrypts a string based on the key provided in the sKey parameter.

Considerations:

This method should only be used to decrypt strings that have been encrypted using the method Encrypt(String s, String sKey). The encrypt/decrypt algorithm used is called Blowfish. Blowfish is a symmetric block cipher that can be used as a drop-in replacement for DES or IDEA. It takes a variable-length key, from 32 bits to 448 bits, making it ideal for both domestic and exportable use.

Encrypt

Signature:

public java.lang.String Encrypt(java.lang.String s_String) throws java.lang.Exception

Use/Purpose:

This method encrypts the string passed in as the parameter. The method uses a session key value as the key for encrypting the string.

Considerations:

Strings encrypted using this method can only be decrypted using the Decrypt(String s) method. The encrypt/decrypt algorithm used is called Blowfish. Blowfish is a symmetric block cipher that can be used as a drop-in replacement for DES or IDEA. It takes a variable-length key, from 32 bits to 448 bits, making it ideal for both domestic and exportable use.

Encrypt

Signature:

public java.lang.String Encrypt(java.lang.String s_String, java.lang.String sKey)

throws java.lang.Exception

Use/Purpose:

This method encrypts the string ‘s’ using the key value passed in with the parameter sKey.

Considerations:

Strings encrypted with this method can only be decrypted using the Decrypt(String s, String sKey) method. The encrypt/decrypt algorithm used is called Blowfish. Blowfish is a symmetric block cipher that can be used as a drop-in replacement for DES or IDEA. It takes a variable-length key, from 32 bits to 448 bits, making it ideal for both domestic and exportable use.

FormatDate

Signature:

public java.lang.String FormatDate(java.sql.Date timeObj, java.lang.String fmtStr)

Use/Purpose:

Converts java.sql.Date object into a String. The format of the string depends upon the value of fmtStr.

Considerations:

If an empty or null string is passed as the format, the method will use the default format of MM/dd/yyyy.

Parameters:

timeObj - the Date value to be formatted into a time string.

fmtStr - A String representing format pattern. If fmtStr value is empty string or null, the function will use default format string of MM/dd/yyyy.

Returns: the String object

FormatDateTime

Signature:

public java.lang.String FormatDateTime(java.sql.Timestamp timeObj, java.lang.String fmtStr)

Use/Purpose:

Converts java.sql.Timestamp object into String

Considerations:

If the format string is empty or null, the default format string (MM/dd/yyyy HH:mm:ss) will be used.

Parameters:

timeObj - the Timestamp value to be formatted into a time string.

fmtStr - A String representing format pattern. If fmtStr value is empty string or null function will use default format string of MM/dd/yyyy HH:mm:ss.

Returns: the String object

FormatTime

Signature:

public java.lang.String FormatTime(java.sql.Time timeObj, java.lang.String fmtStr)

Use/Purpose:

Converts java.sql.Time object into a string.

Considerations:

If the format string is an empty or null, the default format (HH:mm:ss) will be used.

Parameters:

timeObj - the time value to be formatted into a time string.

fmtStr - A String representing format pattern. If fmtStr value is empty string or null function will use default format string (HH:mm:ss).

Returns: the String object

GetCurrentDate

Signature:

public java.sql.Date GetCurrentDate()

Use/Purpose:

Gets the current server date in the form of a java.sql.Date object.

Considerations:

None

Method Examples:

```
java.sql.Date myDate = ctx.getNeuVisUtil().GetCurrentDate();
```

GetCurrentDateTime

Signature:

public java.lang.String GetCurrentDateTime()

Use/Purpose:

Gets the current server date and time as a string in the format of “YYYY-MM-DD HH:MM:SS”.

Considerations:

None

Method Examples:

```
String myString = ctx.getNeuVisUtil().GetCurrentDateTime();
```

GetCurrentTime

Signature:

public java.sql.Time GetCurrentTime()

Use/Purpose:

Returns the current server time in the form of a java.sql.Time object.

Considerations:

None

Method Examples:

```
java.sql.Time myTime = ctx.getNeuVisUtil().GetCurrentTime();
```

GetCurrentTimeStamp

Signature:

public java.sql.Timestamp GetCurrentTimeStamp()

Use/Purpose:

Gets the current timestamp from the server in the form of a java.sql.Timestamp object.

Considerations:

None

Method Examples:

```
java.sql.Timestamp myTimeStamp = ctx.getNeuVisUtil().GetCurrentTimeStamp();
```

getDate

Signature:

public java.sql.Date getDate(java.lang.String dataStr, java.lang.String fmtStr)

Use/Purpose:

Converts String into java.sql.Date object based on format string.

Considerations:

The fmtStr must represent a valid date format.

Parameters:

dateStr - A String representing the date.

fmtStr - A String representing parsing format.

Returns:

The java.sql.Date object or null if dateStr could not be converted

GetDefaultDateFormat

Signature:

public java.lang.String GetDefaultDateFormat()

Use/Purpose:

Returns the Default Date format String, which is "MM/dd/yyyy" unless it was modified.

Considerations:

None

Returns: the String object *

GetDefaultDateTimeFormat

Signature:

public java.lang.String GetDefaultDateTimeFormat()

Use/Purpose:

Returns the Default Date/Time format string, which is "MM/dd/yyyy hh:mm:ss" unless it was modified.

Considerations:

None

Returns: the String object

GetDefaultTimeFormat

Signature:

public java.lang.String GetDefaultTimeFormat()

Use/Purpose:

Returns the Default time format string, which is "HH:mm:ss" unless it was modified.

Considerations:

None

Returns: the String object

GetEnumImage

Signature:

public java.lang.String GetEnumImage(java.lang.String dbDesc, java.lang.String EnumName, int IDValue) throws java.lang.Exception

Use/Purpose:

Retrieves the name and path of an image associated with a particular enumeration value. Enumerations are created in NeuArchitect when defining attributes for classes.

Considerations:

The dbDesc parameter represents the name of the database descriptor set up in NeuArchitect under the Technology Selections/Database tab.

GetEnumValue

Signature:

public java.lang.String GetEnumValue(java.lang.String dbDesc, java.lang.String EnumName, int IDValue) throws java.lang.Exception

Use/Purpose:

Retrieves the name of the Enumeration value based on the database descriptor name and the ID of the particular enumeration value.

Considerations:

If you have an enumeration called shipping_type with two values: Shipping and Billing, NeuArchitect will assign a unique ID to each value. This is the value that must be used for the IDValue parameter. The dbDesc parameter represents the name of the database descriptor set up in NeuArchitect under the Technology Selections/Database tab.

getIncludeDocument

Signature:

public java.lang.String getIncludeDocument(java.lang.String as_path)

Use/Purpose:

Supports server side include functionality. Retrieves the name of an include document to load.

Considerations:

The document must be located in the classpath of the generated application in order to be loaded.

Parameters:

as_path - A String representing the relative file path

Returns: String or "" as default

isDate

Signature:

public boolean isDate(java.lang.String as_Year, java.lang.String as_Month, java.lang.String as_Day)

throws java.lang.Exception

Use/Purpose:

Determine if a year, day, and month combination is a valid date.

Considerations:

Returns true if the values comprise a valid date, or false if they do not.

isEmpty

Signature:

public boolean isEmpty(java.lang.String as_Str)

Use/Purpose:

Checks to see if string parameter as_Str is empty or null.

Considerations:

Returns true if string is empty or null, or false if it is not.

isFloat

Signature:

public boolean isFloat(java.lang.String as_Str, boolean ab_EmptyOk)

Use/Purpose:

Used to validate the passed argument as an unsigned floating point number.

Considerations:

The emptyOK Boolean can be used to allow the passed value to be empty by using true; the default is to disallow empty values.

Valid floats will return a Boolean of true; any other value, such as an alpha character, will return a false value.

Method Examples:

1. String MyFloat = "A";

   boolean FloatReturn = isFloat(MyFloat); **Returns false**

2. FloatReturn = isFloat(13.2); **Returns true**

osInteger1(java.lang.String as_Str, boolean ab_EmptyOk)

Use/Purpose:

Used to validate the passed argument as a positive integer, which is any positive whole number.

Considerations:

Valid integers will return a Boolean of true; any other value, such as an alpha character or negative number, will return a false value.

The emptyOK Boolean can be used to allow the passed value to be empty by using true; the default is to disallow empty values.

Method Examples:

```
boolean IntReturn = isPositiveInteger("-12"); Returns false
String MyInt = "987";

IntReturn = isPositiveInteger(MyInt);.Returns true
```

isValidDate

Signature:

public boolean isValidDate(java.lang.String as_Str)

Use/Purpose:

Checks whether the particular date string is valid or not using context default format pattern .

Considerations:

Parameters:

dateStr - A String representing the date

Returns: boolean

isValidDate

Signature:

public boolean isValidDate (java.lang.String as_Str, java.lang.String format)

Use/Purpose:

Checks whether the particular date string is valid or not for specific format pattern.

Considerations:

Valid date formats are mm/dd/yy, mm/dd/yyyy, m/d/y, m/d/yyyy

** Only U.S. formatted dates are currently validated, i.e., month/day/year formatting - other formatting, e.g., day/month/year, will fail if the day is greater than 12, as it is tested as being a month

Valid dates will return a Boolean of true; as noted previously, invalid dates, including formats that may be correct in other areas, will return a false value.

Method Examples:

1. False date return, with constant

   boolean DateReturn = isValidDate("13/1/2000"); **Returns false**

3. **True date return, passing variable**

   String MyDate = "1/13/2000";

   DateReturn = isValidDate(MyDate); **Returns true**

RemoveSpace

Signature:

public java.lang.String RemoveSpace(java.lang.String as_Str)

Use/Purpose:

Trims all leading and trailing spaces from the string passed in the parameter as_Str

Considerations:

None

Method Examples:

```
// Trim Test
String myString = ctx.getNeuVisUtil().RemoveSpace(" this  ");
```

ResolveURL

Signature:

public java.lang.String ResolveURL(java.lang.String s_protocol, java.lang.String s_filename)

Use/Purpose:

Returns the full document URL string .

Considerations:

None

Parameters:

s_protocol - A String representing the protocol as "http://" or "https://"

s_protocol - A String representing the document file name

Returns: the String object which represents the full URL of the page or servlet

SetDefaultDateFormat

Signature:

public void SetDefaultDateFormat(java.lang.String aFormat)

Use/Purpose:

Sets the Default Date format String for this context.

Considerations:

Must be a valid date format.

Parameters:

aFormat - - format pattern string for dates, such as DD/MM/YYYY.

SetDefaultDateTimeFormat

Signature:

public void SetDefaultDateTimeFormat(java.lang.String aFormat)

Use/Purpose:

Sets the Default timestamp format String for this context.

Considerations:

Must be a valid date/time format such as “YYYY-MM-DD HH:MM:SS”

Parameters:

aFormat - - format pattern string for timestamp.

SetDefaultTimeFormat

Signature:

public void SetDefaultTimeFormat(java.lang.String aFormat)

Use/Purpose:

Sets the Default time format String for this context.

Considerations:

Must be a valid time format such as “HH:MM:SS”

Parameters:

aFormat - - format pattern string for dates.

trim

Signature:

public java.lang.String trim(java.lang.String as_Str)

throws java.lang.Exception

Use/Purpose:

Performs the same function as RemoveSpace. Both leading and trailing spaces are removed.

Considerations:

None

9 ObjectSet

The class, ObjectSet extends java.lang.Object and implements java.io.Serializable. The ObjectSet class is the standard container class used in the NeuArchitect Runtime. It provides all the standard features of a set collection class.

Add

Signature:

public void Add(java.lang.Object Obj)

Use/Purpose:

This method is used to add objects to the ObjectSet.

Considerations:

None

Method Examples:

```
EnumRecord color1 = new EnumRecord(1, "Red", "red.jpg");
EnumRecord color2 = new EnumRecord(2, "Green", "green.jpg");
ObjectSet myColors = new ObjectSet(); //create ObjectSet myColors
myColors.Add(color1); //add color1 to myColors
myColors.Add(color2); //add color2 to myColors
```

GetAt

Signature:

public java.lang.Object GetAt(int index)

Use/Purpose:

To get object at requested index

Considerations:

Should not request object at index greater than or equal to length of objectset

Method Examples:

```
EnumRecord color1 = new EnumRecord(1, "Red", "red.jpg");
EnumRecord color2 = new EnumRecord(2, "Green", "green.jpg");
ObjectSet myColors = new ObjectSet(); //create ObjectSet myColors
myColors.Add(color1); //add color1 to myColors
myColors.Add(color2); //add color2 to myColors
EnumRecord c1 = (EnumRecord)myColors.GetAt(0) ;
```

GetLength

Signature:

public int GetLength()

Use/Purpose:

Gives length of this objectset

Considerations:

none

Method Examples:

```
EnumRecord color1 = new EnumRecord(1, "Red", "red.jpg");
EnumRecord color2 = new EnumRecord(2, "Green", "green.jpg");
ObjectSet myColors = new ObjectSet(); //create ObjectSet myColors
myColors.Add(color1); //add color1 to myColors
myColors.Add(color2); //add color2 to myColors
for(int I = 0 ; I < EnumRecord.GetLenght();I++)
{
   EnumRecord er1 = (EnumRecord)myColors.GetAt(I)  ;
   Er1.doSomething() ;
}
```

NextAvailable

Signature:

public boolean NextAvailable(int pages)

Use/Purpose:

This Method is used to determine if next set of pages are available in paging

Considerations:

None

Method Examples:

Following expression canbe used in visibility expression of label : Next 10

```
ThisSet.NextAvailable(10)
```

PrevAvailable

Signature:

public boolean PrevAvailable(int pages)

Use/Purpose:

This Method is used to determine if previous set of pages are available in paging

Considerations:

None

Method Examples:

Following expression can be used in visibility expression of label : Previous 10

```
ThisSet.PrevAvailable(10)
```

RemoveAll

Signature:

public void RemoveAll()

Use/Purpose:

Removes all objects from this Set

Considerations:

None

Method Examples:

```
If(ThisSet.GetLength() > 0)
{
   ThisSet.RemoveAll() ;
}
```

RemoveAt

Signature:

public void RemoveAt(int index)

Use/Purpose:

Removes Object at given index

Considerations:

Should not attempt to remove object at index greater than or equal to length of objectset

Method Examples:

```
EnumRecord color1 = new EnumRecord(1, "Red", "red.jpg");
EnumRecord color2 = new EnumRecord(2, "Green", "green.jpg");
ObjectSet myColors = new ObjectSet(); //create ObjectSet myColors
myColors.Add(color1); //add color1 to myColors
myColors.Add(color2); //add color2 to myColors
if(myColors.GetLength() > 1)
{
   myColors.RemoveAt(1) ;
}
```

SetAt

Signature:

public void SetAt(int index, java.lang.Object Obj)

Use/Purpose:

Sets Object at given index and each object at given or greater index is shifted upwards

Considerations:

None

Method Examples:

```
EnumRecord color1 = new EnumRecord(1, "Red", "red.jpg");
EnumRecord color2 = new EnumRecord(2, "Green", "green.jpg");
ObjectSet myColors = new ObjectSet(); //create ObjectSet myColors
myColors.Add(color1); //add color1 to myColors
myColors.Add(color2); //add color2 to myColors
EnumRecord newRecord = new EnumRecord(3,"Blue","Blue.gif") ;
MyColors.SetAt(0,newRecord) ;
```

SetLength

Signature:

public void SetLength(int len)

Use/Purpose:

Used to set length of this ObjectSet

Considerations:

If len is greater than ObjectSet length null objects are inserted.

Method Examples:

```
EnumRecord color1 = new EnumRecord(1, "Red", "red.jpg");
EnumRecord color2 = new EnumRecord(2, "Green", "green.jpg");
ObjectSet myColors = new ObjectSet(); //create ObjectSet myColors
myColors.Add(color1); //add color1 to myColors
myColors.Add(color2); //add color2 to myColors
EnumRecord newRecord = new EnumRecord(3,"Blue","Blue.gif") ;
MyColors.SetAt(0,newRecord) ;
MyColors.SetLength(2) ;
```

10 ObjectSpaceBase

The class, ObjectSpaceBase extends java.lang.Object. It is the Base Class Of All Generated ObjectSpaces.

GetAttribute

Signature:

public NeuVisObject GetAttribute(java.lang.String AttrName, int index)

throws java.lang.Exception

Use/Purpose:

Returns NeuVisObject from named ObjectSet at specified index

Considerations:

Index should be less than length of ObjectSet

Method Examples:

Customer c = (Customer)ctx.TheObjectSpace.GetAttribute(“Products”,0) ;

11 Parameters

The class, Parameters extends java.lang.Object. This object is used to manage form parameters that can be used in custom code or visibility expressions on a form or message.

addParameter

Signature:

public void addParameter(java.lang.String KeyObj, boolean Val)

Use/Purpose:

This method is used to add a Boolean value to the parameter list of a form. The two parameters represent a name/value pair. The KeyObj string is the name of the parameter and Val in this case is a Boolean value.

Considerations:

None

Method Examples:

```
ctx.TheParameter.addParameter("isChecked", true);
```

addParameter

Signature:

public void addParameter(java.lang.String KeyObj, double Val)

Use/Purpose:

This method is used to add a double value to the parameter list of a form. The two parameters represent a name/value pair. The KeyObj string is the name of the parameter and Val in this case is a double value.

Considerations:

None

Method Examples:

```
ctx.TheParameter.addParameter("TheTotalVote", 60000000);
```

addParameter

Signature:

public void addParameter(java.lang.String KeyObj, float Val)

Use/Purpose:

This method is used to add a float value to the parameter list of a form. The two parameters represent a name/value pair. The KeyObj string is the name of the parameter and Val in this case is a float value.

Considerations:

None

Method Examples:

```
ctx.TheParameter.addParameter("TotalPrice", 125.95);
```

addParameter

Signature:

public void addParameter(java.lang.String KeyObj, int Val)

Use/Purpose:

This method is used to add a integer value to the parameter list of a form. The two parameters represent a name/value pair. The KeyObj string is the name of the parameter and Val in this case is a integer value.

Considerations:

None

Method Examples:

```
ctx.TheParameter.addParameter("MyAge", 40);
```

addParameter

Signature:

public void addParameter(java.lang.String KeyObj, long Val)

Use/Purpose:

This method is used to add a long value to the parameter list of a form. The two parameters represent a name/value pair. The KeyObj string is the name of the parameter and Val in this case is a long value.

Considerations:

None

Method Examples:

```
ctx.TheParameter.addParameter("MyLongValue", 40000);
```

addParameter

Signature:

public void addParameter(java.lang.String KeyObj, java.lang.Object Val)

Use/Purpose:

This method is used to add an object value such as a date or time or any Java object to the parameter list of a form. The two parameters represent a name/value pair. The KeyObj string is the name of the parameter and Val in this case is an object.

Considerations:

Object must inherit from java.lang.Object.

Method Examples:

```
java.sql.Date mydate;
ctx.TheParameter.addParameter("TheDate", mydate);
```

addParameter

Signature:

public void addParameter(java.lang.String KeyObj, short Val)

Use/Purpose:

This method is used to add a short value to the parameter list of a form. The two parameters represent a name/value pair. The KeyObj string is the name of the parameter and Val in this case is a short value.

Considerations:

None

Method Examples:

```
ctx.TheParameter.addParameter("ShortNumber", 10);
```

addParameter

Signature:

public void addParameter(java.lang.String KeyObj, java.lang.String Val)

Use/Purpose:

This method is used to add a string to the parameter list of a form. The two parameters represent a name/value pair. The KeyObj string is the name of the parameter and Val in this case is a user-defined string.

Considerations:

None

Method Examples:

```
ctx.TheParameter.addParameter("MyTitle", "The Pelican Brief");
```

getHashtable

Signature:

public java.util.Hashtable getHashtable()

Use/Purpose:

Provides the entire parameter list in the form of a java.util.HashTable.

Considerations:

None

getParameterBooleanValue

Signature:

public boolean getParameterBooleanValue(java.lang.Object KeyObj)

Use/Purpose:

This method can be used to return the value associated for a particular form parameter that has been created for an application. This method can be used in custom methods or visibility expressions to retrieve a particular parameter's value. In this particular case, the method is passed a key for an existing Boolean key/value pair and the associated Boolean value is returned.

Considerations:

The key and value must have been added using ***addParameter*** first. If the key is not found a null is returned and a NullPointerException is thrown.

Method Examples:

```
boolean myBool = ctx.TheParameter.getParameterBooleanValue("isChecked");
```

getParameterDoubleValue

Signature:

public double getParameterDoubleValue(java.lang.Object KeyObj)

Use/Purpose:

This method can be used to return the value associated for a particular form parameter that has been created for an application. This method can be used in custom methods or visibility expressions to retrieve a particular parameter's value. In this particular case, the method is passed a key for an existing double key/value pair and the associated double value is returned.

Considerations:

The key and value must have been added using ***addParameter*** first. If the key is not found a null is returned and a NullPointerException is thrown.

Method Examples:

```
double myDbl = ctx.TheParameter.getParameterDoubleValue("TheTotalVote");
```

getParameterFloatValue

Signature:

public float getParameterFloatValue(java.lang.Object KeyObj)

Use/Purpose:

This method can be used to return the value associated for a particular form parameter that has been created for an application. This method can be used in custom methods or visibility expressions to retrieve a particular parameter's value. In this particular case, the method is passed a key for an existing float key/value pair and the associated float value is returned.

Considerations:

The key and value must have been added using ***addParameter*** first. If the key is not found a null is returned and a NullPointerException is thrown.

Method Examples:

```
float myfloat = ctx.TheParameter.getParameterFloatValue("TheTotalPrice");
```

getParameterIntValue

Signature:

public int getParameterIntValue(java.lang.Object KeyObj)

Use/Purpose:

This method can be used to return the value associated for a particular form parameter that has been created for an application. This method can be used in custom methods or visibility expressions to retrieve a particular parameter's value. In this particular case, the method is passed a key for an existing integer key/value pair and the associated integer value is returned.

Considerations:

The key and value must have been added using ***addParameter*** first. If the key is not found a null is returned and a NullPointerException is thrown.

Method Examples:

```
int myInt = ctx.TheParameter.getParameterIntValue("MyAge");
```

getParameterLongValue

Signature:

public long getParameterLongValue(java.lang.Object KeyObj)

Use/Purpose:

This method can be used to return the value associated for a particular form parameter that has been created for an application. This method can be used in custom methods or visibility expressions to retrieve a particular parameter's value. In this particular case, the method is passed a key for an existing long key/value pair and the associated long value is returned.

Considerations:

The key and value must have been added using ***addParameter*** first. If the key is not found a null is returned and a NullPointerException is thrown

Method Examples:

```
long myLong = ctx.TheParameter.getParameterLongValue("MyLongValue");
```

getParameterShortValue

Signature:

public short getParameterShortValue(java.lang.Object KeyObj)

Use/Purpose:

This method can be used to return the value associated for a particular form parameter that has been created for an application. This method can be used in custom methods or visibility expressions to retrieve a particular parameter's value. In this particular case, the method is passed a key for an existing short key/value pair and the associated short value is returned.

Considerations:

The key and value must have been added using ***addParameter*** first. If the key is not found a null is returned and a NullPointerException is thrown.

Method Examples:

```
short myShort = ctx.TheParameter.getParameterShortValue("MyShortValue");
```

getParameterStringValue

Signature:

public java.lang.String getParameterStringValue(java.lang.Object KeyObj)

Use/Purpose:

This method can be used to return the value associated for a particular form parameter that has been created for an application. This method can be used in custom methods or visibility expressions to retrieve a particular parameter's value. In this particular case, the method is passed a key for an existing string key/value pair and the associated string is returned.

Considerations:

The key and value must have been added using ***addParameter*** first. If the key is not found a null is returned and a NullPointerException is thrown.

Method Examples:

```
String myString = ctx.TheParameter.getParameterStringValue("MyStringValue");
```

getParameterValue

Signature:

public java.lang.Object getParameterValue(java.lang.Object KeyObj)

Use/Purpose:

This method can be used to return an object reference associated for a particular form parameter that has been created for an application. This method can be used in custom methods or visibility expressions to retrieve a particular parameter's value. In this particular case, the method is passed a key for an existing object key/value pair and the associated object reference is returned.

Considerations:

The key and value must have been added using ***addParameter*** first. If the key is not found a null is returned and a NullPointerException is thrown. The object can be any valid Java object.

Method Examples:

```
Java.sql.Date myDate = ctx.TheParameter.getParameterValue("TheDate");
```

initParameters

Signature:

public void initParameters()

Use/Purpose:

Removes all the parameters and resets the parameter list.

Considerations:

This will remove all keys and associated values from the list. Use with care.

Method Examples:

```
ctx.TheParameter.initParameters();
```

setHashtable

Signature:

public void setHashtable(java.util.Hashtable i_Parameter)

Use/Purpose:

Creates a new hashtable for holding the parameter list.

Considerations:

Will overwrite the current parameter list. Use with care.

12 Interface Request

Request is an interface that defines a platform neutral HTTP request that a Web server receives from a client that uses HTTP.

cleanup

Signature:

public void cleanup()

Use/Purpose:

Releases all resources held by the Request instance

Considerations:

None

getAuthType

Signature:

public java.lang.String getAuthType()

Use/Purpose:

Returns the name of the authentication scheme the server uses, for example, "BASIC" or "SSL," or null if the server does not have an authentication scheme.

Considerations:

The authentication scheme provides a challenge-response model in which the server challenges the client; the client provides authentication information. Same as value of CGI variable AUTH_TYPE.

Returns:

a String specifying the name of the authentication scheme, or null if the server does not have an authentication scheme

getContentLength

Signature:

public int getContentLength()

Use/Purpose:

Returns the length, in bytes, of the content contained in the request and sent by way of the input stream or -1 if the length is not known. Same as the value of the CGI variable CONTENT_LENGTH.

Considerations:

Returns: an integer containing the length of the content in the request or -1 if the length is not known

getCookies

Signature:

public com.neuvis.neuarchitect.runtime.base.Cookie[] getCookies()

Use/Purpose:

Returns an array containing all of the Cookie objects the browser sent with this request. This method returns null if the browser did not send any cookies.

Considerations:

Returns: an array of all the Cookies included with this request, or null if the request has no cookies

getCookieValue

Signature:

public java.lang.String getCookieValue(java.lang.String as_name)

Use/Purpose:

Returns the string value of the named Cookie String This method returns null if the no such Cookie found.

Considerations:

Returns: String value of the named Cookie String included with this request, or null if the request has no matched cookie

getDateHeader

Signature:

public long getDateHeader(java.lang.String name)

Use/Purpose:

Returns the value of the specified request header as a long value that represents a Date object. Use this method with headers that contain dates, such as If-Modified-Since.

Considerations:

The date is returned as the number of milliseconds since January 1, 1970 GMT. The header name is case insensitive.

If the request did not have a header of the specified name, this method returns -1. If the header can't be converted to a date, the method returns an IllegalArgumentException.

Parameters: **name** - a String specifying the name of the header

Returns: a long value representing the date specified in the header expressed as the number of milliseconds since January 1, 1970 GMT, or -1 if the named header was not included with the reqest

getFileNames

Signature:

public java.util.Enumeration getFileNames()

Use/Purpose:

Returns the names of all the uploaded files as an Enumeration of Strings

Considerations:

None

getFilesystemName

Signature:

public java.lang.String getFilesystemName(java.lang.String name)

Use/Purpose:

Returns the filesystem name of the specified file, or null if the file was not included in the upload. A filesystem name is the name specified by the user. It is also the name under which the file is actually saved.

Considerations:

Parameters: **name** - the file name

Returns: the filesystem name of the file

getFilesystemName

Signature:

public java.lang.String getFilesystemName(java.lang.String name)

Use/Purpose:

Returns the filesystem name of the specified file, or null if the file was not included in the upload. A filesystem name is the name specified by the user. It is also the name under which the file is actually saved.

Considerations:

Parameters: **name** - the file name

Returns: the filesystem name of the file

getFullFileName

Signature:

public java.lang.String getFullFileName(java.lang.String name)

Use/Purpose:

Returns the full filename for the file to upload.

Considerations:

If the file does not exist, a null is returned.

getHeader

Signature:

public java.lang.String getHeader(java.lang.String name).

Use/Purpose:

Returns the value of the specified request header as a String. If the named header wasn't sent with the request, this method returns null. The header name is case insensitive. You can use this method with any request header.

Considerations:

Parameters: **name** - a String specifying the header name

Returns: a String containing the value of the requested header, or null if the request does not have a header of that name

getHeaderNames

Signature:

public java.util.Enumeration getHeaderNames()

Use/Purpose:

Returns an enumeration of all the header names this request contains. If the request has no headers, this method returns an empty enumeration.

Considerations:

Some servlet engines do not allow do not allow servlets to access headers using this method, in which case this method returns null

Returns: an enumeration of all the header names sent with this request; if the request has no headers, an empty enumeration; if the servlet engine does not allow servlets to use this method, null

getIntHeader

Signature:

public int getIntHeader(java.lang.String name)

Use/Purpose:

Returns the value of the specified request header as an integer. If the request does not have a header of the specified name, this method returns -1. If the header cannot be converted to an integer, this method throws a NumberFormatException

Considerations:

The header name is case insensitive.

Parameters: **name** - a String specifying the name of a request header

Returns: an integer expressing the value of the request header or -1 if the request doesn't have a header of this name

getMethod

Signature:

public java.lang.String getMethod()

Use/Purpose:

Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT. The returned String is the same as the value of the CGI variable REQUEST_METHOD.

Considerations:

Returns: a String specifying the name of the method with which this request was made

getParameter

Signature:

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

Use/Purpose:

Returns the value of a request parameter as a String, or null if the parameter does not exist. Request parameters are extra information sent with the request.

Considerations:

You should only use this method when you are sure the parameter has only one value. If the parameter might have more than one value, use getParameterValues(java.lang.String).

If you use this method with a multivalued parameter, the servlet engine determines the return value.

Parameters: **name** - a String specifying the name of the parameter

Returns: a String representing the single value of the parameter

See Also: getParameterValues(java.lang.String)

getParameterNames

Signature:

public java.util.Enumeration getParameterNames()

Use/Purpose:

Returns an Enumeration of String objects containing the names of the parameters contained in this request. If the request has no parameters or if the input stream is empty, returns an empty Enumeration.

Considerations:

The input stream is empty when all the data returned by #getInputStream has been read.

Returns: an Enumeration of String objects, each String containing the name of a request parameter; or an empty Enumeration if the request has no parameters.

getParameterValues

Signature:

public java.lang.String[] getParameterValues(java.lang.String name)

Use/Purpose:

Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist. For example, in an HTTP servlet, this method returns an array of String objects containing the values of a query string or posted form.

Considerations:

If the parameter has a single value, the array has a length of 1.

Parameters: **name** - a String containing the name of the parameter whose value is requested

Returns: an array of String objects containing the parameter's values

See Also: getParameter(java.lang.String)

getPathInfo

Signature:

public java.lang.String getPathInfo()

Use/Purpose:

Returns any extra path information associated with the URL the client sent when it made the request.

Considerations:

None

getQueryString

Signature:

public java.lang.String getQueryString()

Use/Purpose:

Returns the query string that is in the request URL after the path information.

Considerations:

None

getRemoteUser

Signature:

public java.lang.String getRemoteUser()

Use/Purpose:

Returns the name of the user making this request, if the user has logged in using HTTP authentication. This method returns null if the user login is not authenticated.

Considerations:

Whether the user name is sent with each subsequent request depends on the browser. Same as the value of the CGI variable REMOTE_USER.

Returns: a String specifying the name of the user making this request, or null

getServerName

Signature:

public java.lang.String getServerName()

Use/Purpose:

Returns the Server Name of the host that received the request

Considerations:

None

getServerVariable

Signature:

public java.lang.String getServerVariable(java.lang.String sItem)

Use/Purpose:

Returns the value of the specified request header in the form of a string.

Considerations:

None

13 Interface Response

Response is an interface that defines a platform neutral HTTP response that a Web server sends to a client using the HTTP protocol.

AddCookie

Signature:

public void AddCookie(com.neuvis.neuarchitect.runtime.base.Cookie cookie)

Use/Purpose:

Adds the specified cookie to the response. It can be called multiple times to set more than one cookie.

Considerations:

Parameters: **cookie** - the Cookie to return to the client

AddHeader

Signature:

public void AddHeader(java.lang.String name, int value)

Use/Purpose:

Adds a field to the response header with the given name and integer value. If the field had already been set, the new value overwrites the previous one.

Considerations:

The ***containsHeader*** method can be used to test for the presence of a header before setting its value.

Parameters:

name - the name of the header field

value - the header field's integer value

AddHeader

Signature:

public void AddHeader(java.lang.String name, long date)

Use/Purpose:

Adds a field to the response header with the given name and date-valued field. The date is specified in terms of milliseconds since the epoch.

Considerations:

If the date field had already been set, the new value overwrites the previous one. The ***containsHeader*** method can be used to test for the presence of a header before setting its value.

Parameters:

name - the name of the header field

value - the header field's date value

AddHeader

Signature:

public void AddHeader(java.lang.String name, java.lang.String value)

Use/Purpose:

Adds a field to the response header with the given name and value.

Considerations:

If the field had already been set, the new value overwrites the previous one. The ***containsHeader*** method can be used to test for the presence of a header before setting its value.

Parameters:

name - the name of the header field

value - the header field's value

cleanup

Signature:

public void cleanup()

Use/Purpose:

Releases all resources held by the Response instance and sets the Response pointer to null.

Considerations:

None

Flush

Signature:

public void Flush()

Use/Purpose:

Sends buffered output immediately and clears the buffer.

Considerations:

None

getNextDoc

Signature:

public java.lang.String getNextDoc()

Use/Purpose:

Return a string of the base filename of the next document/servlet/asp to which control is redirected.

Considerations:

The filename returned will be a servlet, JSP or ASP.

Redirect

Signature:

public void Redirect(java.lang.String newURL)

throws java.io.IOException

Use/Purpose:

Sends a temporary redirect response to the client using the specified redirect location URL.

Considerations:

URL must be absolute (e.g., https://hostname/path/file.html). Relative URLs are not permitted here.

Parameters: **newURL** - the redirect location URL

setContentLength

Signature:

public void setContentLength(int len)

Use/Purpose:

Sets the length of the content the server returns to the client.

Considerations:

In HTTP servlets, this method sets the HTTP Content-Length header.

Parameters: **len** - an integer specifying the length of the content being returned to the client; sets the Content-Length header

setContentType

Signature:

public void setContentType(java.lang.String type)

Use/Purpose:

Sets the content type of the response the server sends to the client.

Considerations:

The content type may include the type of character encoding used, for example, text/html; charset=ISO-8859-4.

You can only use this method once, and you should call it before you obtain a PrintWriter or ServletOutputStream object to return a response.

Parameters: **type** - a String specifying the MIME type of the content

setExpires

Signature:

public void setExpires(int minutes)

Use/Purpose:

Sets the length of time before a page cached on a browser expires. If the user returns to the same page before it expires, the cached version is displayed.

Considerations:

Creates an HTTP header indicating the time on the server. If the system time on the client is earlier than the system time on the server (due to either the client or server having an inaccurate time setting, or time-zone differences) setting the parameter to 0 will not have the effect of expiring the page immediately. You can use Response.setExpiresAbsolute to achieve immediate expiration of a page. In addition, you can use a negative number for the Expires property. For example will expire the response immediately.

setExpiresAbsolute

Signature:

public void setExpiresAbsolute(long date)

Use/Purpose:

Sets the date and time at which a page cached on a browser expires.

Considerations:

If the user returns to the same page before that date and time, the cached version is displayed.

setNextDoc

Signature:

public void setNextDoc(java.lang.String as)

Use/Purpose:

NextDoc indicating which servlet/asp will be redirected to.

Considerations:

Parameter must represent a valid application document.

setStatus

Signature:

public void setStatus(int sc)

Use/Purpose:

Sets the status code for this response.

Considerations:

This method is used to set the return status code when there is no error (for example, for the status codes SC_OK or SC_MOVED_TEMPORARILY).

Parameters: **sc** - the status code

setStatus

Signature:

public void setStatus(int sc, java.lang.String sm)

throws java.io.IOException

Use/Purpose:

Sets the status code and message for this response.

Considerations:

If the field had already been set, the new value overwrites the previous one. The message is sent as the body of an HTML page, which is returned to the user to describe the problem. The page is sent with a default HTML header; the message is enclosed in simple body tags (<body></body>).

Parameters:

sc - the status code

sm - the status message

Write

Signature:

public void Write(java.sql.Time in)

throws java.io.IOException

Use/Purpose:

Writes the value of the java.sql.Time parameter passed into the function to the HTTP output.

Considerations:

None

Write

Signature:

public void Write(double in)

throws java.io.IOException

Use/Purpose:

Writes the value of the parameter passed into the function to the HTTP output.

Considerations:

None

Write

Signature:

public void Write(int in)

throws java.io.IOException

Use/Purpose:

Writes the value of the parameter passed into the function to the HTTP output.

Considerations:

None

Write

Signature:

public void Write(java.lang.String s)

throws java.io.IOException

Use/Purpose:

Writes a specified string to the current HTTP output.

Considerations:

This value cannot contain the character combination %>; instead you should use the escape sequence %\>. The Web server will translate the escape sequence when it processes the script.

Write

Signature:

public void Write(java.sql.Time in)

throws java.io.IOException

Use/Purpose:

Writes the value of the parameter passed into the function to the HTTP output.

Considerations:

Must be a valid java.sql.Time object.

Write

Signature:

public void Write(java.sql.Timestamp in)

throws java.io.IOException

Use/Purpose:

Writes the value of the parameter passed into the function to the HTTP output.

Considerations: Must be a valid java.sql.Timestamp object.

14 Interface Server

Server is an interface that defines a platform neutral object representing a Web server.

AppendToLog

Signature:

public void AppendToLog(java.lang.String msg)

Use/Purpose:

Writes the specified message to the Web Server log file, which is usually an event log.

Considerations:

The message provides explanatory information about an exception or error or an action the servlet engine takes. The name and type of the message is specific to the Web Server.

Parameters: **msg** - a String specifying the explanatory message to be written to the log file

cleanup

Signature:

public void cleanup()

Use/Purpose:

Releases all resources held by the Server instance

Considerations:

None

EncodeURL

Signature:

public java.lang.String EncodeURL(java.lang.String url)

Use/Purpose:

Encodes the specified URL.

Considerations:

All URLs emitted should be run through this method.

Parameters: url - the url to be encoded.

Returns: The encoded URL if encoding is needed; the unchanged URL otherwise.

MapPath

Signature:

public java.lang.String MapPath(java.lang.String path)

Use/Purpose:

Maps the specified relative or virtual path to the corresponding physical directory on the server.

Considerations:

The real path returned is in a form appropriate to the computer and operating system on which the Web Server is running, including the proper path separators. This method returns null if the Web Server cannot translate the virtual path to a real path for any reason.

Parameters: **path -** a String specifying a virtual path, in the form /dir/dir/file.ext

Returns: a String specifying the real path, with path separators appropriate for the system on which the Web Server is running

15 Interface Session

Session is an interface that defines a platform neutral object representing a session. The programmer can store and retrieve data for a particular session with this object.

cleanup

Signature:

public void cleanup()

Use/Purpose:

Releases all resources held by the Request instance

Considerations:

None

getBooleanValue

Signature:

public boolean getBooleanValue(java.lang.String name)

Use/Purpose:

Retrieve a named Boolean value from the Session object. The values are stored in Session in the form of name/value pairs. The values can be retrieved using the name and the appropriate function.

Considerations:

The value needs to have been place into the Session object using the **putValue (String name, boolean value)** function first. The ***name*** parameter used during the put is passed to getBooleanValue in order to retrieve the associated value.

getCharValue

Signature:

public char getCharValue(java.lang.String name)

Use/Purpose:

Retrieve a char value from the Session object. The values are stored in Session in the form of name/value pairs. The values can be retrieved using the name and the appropriate function.

Considerations:

The value needs to have been place into the Session object using the **putValue (String name, char value)** function first. The ***name*** parameter used during the put is passed to getCharValue in order to retrieve the associated value.

getDoubleValue

Signature:

public double getDoubleValue(java.lang.String name)

Use/Purpose:

Retrieve a double value from the Session object. The values are stored in Session in the form of name/value pairs. The values can be retrieved using the name and the appropriate function.

Considerations:

The value needs to have been place into the Session object using the **putValue (String name, double value)** function first. The ***name*** parameter used during the put is passed to getDoubleValue in order to retrieve the associated value.

getFloatValue

Signature:

public float getFloatValue(java.lang.String name)

Use/Purpose:

Retrieve a float value from the Session object. The values are stored in Session in the form of name/value pairs. The values can be retrieved using the name and the appropriate function.

Considerations:

The value needs to have been place into the Session object using the **putValue (String name, float value)** function first. The ***name*** parameter used during the put is passed to getFloatValue in order to retrieve the associated value.

getId

Signature:

public java.lang.String getId()

Use/Purpose:

Returns a string containing the unique identifier assigned to this session.

Considerations:

The identifier is assigned by the servlet engine and is implementation dependent.

Returns: a string specifying the identifier assigned to this session

getIntValue

Signature:

public int getIntValue(java.lang.String name)

Use/Purpose:

Retrieve an integer value from the Session object. The values are stored in Session in the form of name/value pairs. The values can be retrieved using the name and the appropriate function.

Considerations:

The value needs to have been place into the Session object using the **putValue (String name, int value)** function first. The ***name*** parameter used during the put is passed to getIntValue in order to retrieve the associated value.

getLongValue

Signature:

public long getLongValue(java.lang.String name)

Use/Purpose:

Retrieve a long value from the Session object. The values are stored in Session in the form of name/value pairs. The values can be retrieved using the name and the appropriate function.

Considerations:

The value needs to have been place into the Session object using the **putValue (String name, long value)** function first. The ***name*** parameter used during the put is passed to getLongValue in order to retrieve the associated value.

getShortValue

Signature:

public short getShortValue(java.lang.String name)

Use/Purpose:

Retrieve a short value from the Session object. The values are stored in Session in the form of name/value pairs. The values can be retrieved using the name and the appropriate function.

Considerations:

The value needs to have been place into the Session object using the **putValue (String name, short value)** function first. The ***name*** parameter used during the put is passed to getShortValue in order to retrieve the associated value.

getStringValue

Signature:

public java.lang.String getStringValue(java.lang.String name)

Use/Purpose:

Retrieve a string value from the Session object. The values are stored in Session in the form of name/value pairs. The values can be retrieved using the name and the appropriate function.

Considerations:

The value needs to have been place into the Session object using the **putValue (String name, String value)** function first. The ***name*** parameter used during the put is passed to getStringValue in order to retrieve the associated value.

getValue

Signature:

public java.lang.Object getValue(java.lang.String name)

Use/Purpose:

Returns the object bound with the specified name in this session or null if no object of that name exists.

Considerations:

Parameters: **name** - a string specifying the name of the object

Returns: the object with the specified name

Throws: IllegalStateException - if the session is invalid

getValueNames

Signature:

Use/Purpose: getValueNames

Returns an array containing the names of all the objects bound to this session.

Considerations:

This method is useful, for example, when you want to delete all the objects bound to this session.

Returns: an array of strings specifying the names of all the objects bound to this session

Throws: IllegalStateException - if the session is invalid

invalidate

Signature:

public void invalidate()

Use/Purpose:

Invalidates this session and unbinds any objects bound to it.

Considerations:

Throws: IllegalStateException - if the session is already invalid

putValue

Signature:

public void putValue(java.lang.String name, boolean value)

Use/Purpose:

Places a name/value pair in the session object. The value in this case is of type boolean.

Considerations:

Throws: IllegalStateException - if the session is invalid

putValue

Signature:

public void putValue(java.lang.String name, char value)

Use/Purpose:

Places a name/value pair in the session object. The value in this case is of type char.

Considerations:

Throws: IllegalStateException - if the session is invalid

putValue

Signature:

public void putValue(java.lang.String name, double value)

Use/Purpose:

Places a name/value pair in the session object. The value in this case is of type double.

Considerations:

Throws: IllegalStateException - if the session is invalid

putValue

Signature:

public void putValue(java.lang.String name, float value)

Use/Purpose:

Places a name/value pair in the session object. The value in this case is of type float.

Considerations:

Throws: IllegalStateException - if the session is invalid

putValue

Signature:

public void putValue(java.lang.String name, int value)

Use/Purpose:

Places a name/value pair in the session object. The value in this case is of type int.

Considerations:

Throws: IllegalStateException - if the session is invalid

putValue

Signature:

public void putValue(java.lang.String name, long value)

Use/Purpose:

Places a name/value pair in the session object. The value in this case is of type long.

Considerations:

Throws: IllegalStateException - if the session is invalid

putValue

Signature:

public void putValue(java.lang.String name, java.lang.Object value)

Use/Purpose:

Binds an object to this session, using the name specified.

Considerations:

If an object of the same name is already bound to the session, the object is replaced.

Parameters:

name - the name to which the object is bound; cannot be null

value - the object to be bound; cannot be null

Throws: IllegalStateException - if the session is invalid

putValue

Signature:

public void putValue(java.lang.String name, short value)

Use/Purpose:

Places a name/value pair in the session object. The value in this case is of type short.

Considerations:

Throws: IllegalStateException - if the session is invalid

putValue

Signature:

public void putValue(java.lang.String name, java.lang.String value)

Use/Purpose:

Places a name/value pair in the session object. The value in this case is of type String.

Considerations:

Throws: IllegalStateException - if the session is invalid

removeValue

Signature:

public void removeValue(java.lang.String name)

Use/Purpose:

Removes the object bound with the specified name from this session.

Considerations:

If the session does not have an object bound with the specified name, this method does nothing.

After this method executes, and if the object implements HttpSessionBindingListener, the object calls HttpSessionBindingListener.valueUnbound.

Parameters: **name** - the name of the object to remove from this session

Throws: IllegalStateException - if the session is invalid