

## Dbank's "tsDatabase" COM Object

Dbank's setup program installs a dynamic link library ("readdb.dll") that defines a time-series *database* object model (or a COM library) for time-series storage. This object allows you to build Dbank time-series databases using Microsoft SQL Server. Dbank's database object can be accessed from Microsoft Visual Basic, Visual Basic for Applications, Active Server Pages, Power Builder, and Visual C++. Together with Dbank's time-series object, *dbTimeSeries*, Dbank's database object allows third party programmers to build powerful, customized time-series solutions.

The purpose of this manual is to provide a brief, but comprehensive, description of Dbank's time-series *database* object model. The database object has methods for creating and managing Dbank time-series databases, manipulating Dbank groups, and managing persistent time-series variables (including their footnotes).

A separate license is needed to use Dbank's SQL time-series storage engine. In particular, purchasing a copy of Dbank itself does not grant the purchaser the right to use the SQL backend.

This manual assumes that your base language is either Visual Basic (VB) or Visual Basic for Applications (VBA).

### 1. Accessing Dbank's Time-Series Database Object From VB/VBA/ASP

Before you can access Dbank's time-series database object model, you need to (a) install Dbank; and (b) activate a reference to the dynamic link library that actually defines the object ("readdb.dll", typically saved in the Windows system area).

To access the database object from VB, click on "Project", then "References" and select "Read and Write Dbank Time Series Files" by checking the appropriate check box.

From VBA (e.g., Microsoft Excel), click on "Tools", followed by "References" and select "Read and Write Dbank Time Series Files".

From active server pages, you need to create a *tsDatabase* object using the CreateObject() method available in Visual Basic script.

### Defining a Time-Series Database Object From Visual Basic/Visual Basic for Applications

As mentioned above, Dbank's time-series database object is called "tsDataBase". Like any other object in Visual Basic, VB/VBA's Dim statement is used to declare a new time-series database object.

Thus:

```
Dim X as New tsDatabase
    'Defines X to be a Dbank time-series database object
```

The following syntax is also valid VB/VBA:

```
Dim Y as tsDatabase ` "new" keyword is absent
                    `Defines Y to be an object variable `that
                    can reference a Dbank time-`series database
                    object
```

However, this statement does not actually create a time-series database object; here "Y" can refer only to an existing database object. Normally, statements of this type are followed by:

```
Set Y = X,
```

revealing that Y provides another name for the time-series database object X.

## Defining a Time-Series Object from Visual Script/Active Server Pages

Dbank's tsDatabase object can be used within active server pages. The following VB script creates a tsDatabase object on the server itself:

```
dim objTsd
set objTsd = Server.CreateObject("DbankEngine.tsDatabase")
```

## 2. Reading/Writing Dbank Time-Series Variables

A fully qualified time-series name in Dbank has the following form (optional components are indicated using {}):

```
{<ServerName>}Database{:}{[Group]}SeriesName
```

where:

```
<ServerName> = Name of an accessible Microsoft SQL Server, e.g., <ORION>
Database      = Name of the SQL or Access database to store the time-series
Group         = time-series group or container
SeriesName    = time-series name
```

Time-series names and group names can be (individually) up to 64 characters long. Database names must observe the rules and requirements of the native filing system (e.g., SQL server rules for naming databases, and Windows 95/NT/2000 rules for naming files).

If <ServerName> is omitted, the time-series database object uses the DAO 3.6 engine to save time series (using Access 2000 databases). For example,

```
"c:\Program Files\Dbank32\example[ace]a"
```

implies that "a", a time-series, can be found in the Microsoft Access database

```
"c:\Program Files\Dbank32\example.mdb"
```

in the group called “[ace]”. As another example,

```
"<ORION>example[ace]a"
```

implies that the time-series “a” can be located in the database “example” that exists in the Microsoft SQL Server called “ORION”.

The following VB/VBA statements show how to (a) read a Dbank time-series (using Dbank’s dbTimeSeries object) from an Access MDB database; and (b) write it to a time-series database on a Microsoft SQL Server called “tsSQL”:

```
Dim X as New dbTimeSeries
If X.Read("c:\Program Files\Dbank32\example[ace]a") Then
    X.Save("<tsSQL>newdb[ace]a")
End If
```

Dbank’s “Read” method retrieves a persistent time-series into the object “X”, returning “True” if the read is successful. In the above example, the “Save” method writes the same series to another database called “newdb” that resides in the Microsoft SQL server called “tsSQL”. Note that the “Save” method will create new databases as and when required.

Accessing the actual data values of the time-series is straightforward. You use the “DataValue” property of a series. This property allows you to retrieve and set the value of a series at a particular date (or offset). For example, the following code creates a time-trend starting at 1 with 100 observations, saving it to a database called “example” that can be found on the Microsoft SQL server called “tsSQL”.

```
Dim X as New dbTimeSeries
X.Frequency = "Q"
X.Nobs = 100
X.StartDate = "1961:1"
For j = 1 to X.Nobs
    X.DataValue(j) = j
Next j
X.Save("<tsSQL>example[test]trend")

`Now display 1968:4
Msgbox X.DataValue("1968:4")
```

This completes the basics of defining time-series database objects using Visual Basic/Visual Basic for Applications, and reading and writing time-series variables.

The remaining sections of this guide provide a reference to all the properties and methods supported by Dbank’s time-series database object.

## **dbkActiveDatabase() Method [String]**

---

You need to set the active database before attempting any database operations. This command sets the active time-series database; that is, it associates a physical database with the time series database object.

### **Syntax**

**X.dbkActiveDatabase** = "Dbank database name"

### **Example**

Set the active database to "SQLExample" on the SQL server <tsSQL>.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    X.dbkClose()
End If
```

## **dbkActiveUser() Method [String]**

---

Returns the current SQL user.

### **Syntax**

**X.dbkActiveUser()**

### **Example**

Set the active database to "SQLExample" on the SQL server <tsSQL>, and return the SQL userid of the user.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    MsgBox X.dbkActiveUser
    X.dbkClose()
End If
```

## **dbkChangeLabel() Method [Boolean]**

---

Changes the label (i.e., short descriptor) of the active database. Returns “True” if successful.

### **Syntax**

**X.dbkChangeLabel**( NewLabel as String )

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>. Change the database label to “Hello World”, and then close it.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If Not X.dbkChangeLabel("Hello World") Then
        MsgBox "Failed to change database label to "Hello World""
    End If
    X.dbkClose()
End If
```

## **dbkChangeMemo() Method [Boolean]**

---

Changes the memo (i.e., long descriptor) of the active database. Returns “True” if successful.

### **Syntax**

**X.dbkChangeMemo**( NewMemo as String )

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>. Change the database label to “Memos can be very long indeed”.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If Not X.dbkChangeMemo("Memo entries can be very long.") Then
        MsgBox "Failed to change database memo."
    End If
    X.dbkClose()
End If
```

## **dbkClose() Method [Boolean]**

---

Closes the active database (see also **dbkActiveDatabase** method). Returns “True” if successful.

### **Syntax**

**X.dbkClose()**

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>. Then close it using **dbkClose()**.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    X.dbkClose()
End If
```

## **dbkCopySQLToMDB() Method [Boolean]**

---

Copies all the time-series in a SQL database to a Microsoft Access database. Returns “True” if successful. Set the “IgnoreSaveErrors” parameters to “True” if you want the procedure to ignore any errors that occur when saving a time-series to SQL.

### **Syntax**

**X.dbkCopyToSQL**(SQLName as String, MDBFile as String, [*KillExistingDbankFile As Boolean = False*], [*QueryDelete As Boolean = True*], [*IgnoreSaveErrors As Boolean = False*])

### **Example**

Migrate the SQL “SQLExample” database to an Access MDB file.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    If X.dbkCopySQLToMDB("SQLExample", "c:\program _
        files\dbank32\example.mdb") Then
        MsgBox "Migration to Access (MDB) file successful."
    End If
End If
```

## dbkCopyToSQL() Method [Boolean]

---

This procedure migrates all the time-series objects in a Microsoft Access file to a SQL Server database. You can optionally create a report that logs all the time-series that the procedure copied to SQL by supplying a fully qualified filename in the “LogFileName” parameter. This method can optionally compare the source database against the SQL version to verify the SQL copy.

### Syntax

**X.dbkCopyToSQL**(AccessDbank as String, SQLDataBase as String,*[LogFileName As Variant]*, *[TruncateExistingSQLDatabase As Boolean = False]*, *[CompareSourceAgainstTarget As Boolean = False]*, *[QuietMode As Boolean = False]*)

### Example

Migrate the Access “example” database to SQL server <tsSQL>.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    If X.dbkCopyToSQL("c:\program files\dbank32\example.mdb", "SQLExample") Then
        MsgBox "Migration of Access MDB file to SQL successful."
    End If
End If
```

## dbkDelete() Method [Boolean]

---

Deletes a SQL database.

### Syntax

**X.dbkDelete**( *[dbName as Variant]*, *[Query as Boolean=False]* )

### Example

Deletes “SQLExample” on SQL server <tsSQL>.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    If X.dbkDelete("SQLExample") Then
        MsgBox "SQL example deleted from server " & X.Server
    End If
End If
```

## **dbkEmpty() Method [Boolean]**

---

Truncates a database. This method deletes all groups and series in a database without destroying the database itself. The physical tables of the database are retained. Thus, user rights and privileges are not affected by **dbkEmpty()**

### **Syntax**

**X.dbkEmpty**( *[dbkName]*, *[Query as Boolean=False]* )

### **Example**

Truncate "SQLExample" on SQL server <tsSQL>.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    If X.dbkEmpty("SQLExample") Then
        MsgBox "SQL example emptied."
    End If
End If
```

## **dbkExists() Method [Boolean]**

---

Indicates whether the Dbank database exists.

### **Syntax**

**X.dbkExists**( *[dbkName as String]* )

### **Example**

Test whether "SQLExample" on the SQL server <tsSQL> exists.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    If X.dbkExists("SQLExample") Then
        MsgBox "SQL example exists in server " & X.Server
    End If
End If
```

## **dbkFindAllDataBases() Method [Boolean]**

---

Returns all the Dbank time-series databases in a given SQL server.

### **Syntax**

**X.dbkFindAllDataBases**(ActiveServer As Variant, dbkNames() As Variant, Count As Long)

### **Example**

Find all the databases in the SQL Server called <tsSQL>

```
Dim X as New tsDatabase
Dim dbkNames() as Variant
Dim Count as Long
If X.Connect() Then
    Call X.dbkFindAllDataBases("<tsSQL>", dbkNames(), Count)
    If Count > 0 Then
        MsgBox "Total number of databases found = " & Count
        For j = 0 to Count-1
            MsgBox "Database name " & j & " : " & dbkNames(j)
        Next j
    End If
End If
```

## **dbkGetLabel() Method [String]**

---

Returns the label (i.e., short description) of the active time-series database.

### **Syntax**

**X.dbkGetLabel()**

### **Example**

Retrieve the label of the database "SQLExample" on the SQL server <tsSQL>.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    If X.dbkExists("SQLExample") Then
        X.dbkActiveDatabase = "SQLExample"
        MsgBox "Label of SQLExample: " & X.dbkGetLabel()
    End If
    X.dbkClose
End If
```

## **dbkGetMemo() Method [String]**

---

Returns the memo (i.e., long description) of the active time-series database.

### **Syntax**

**X.dbkGetMemo()**

### **Example**

Retrieve the memo of the database “SQLExample” on the SQL server <tsSQL>.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    If X.dbkExists("SQLExample") Then
        X.dbkActiveDatabase = "SQLExample"
        MsgBox "Memo of SQLExample: " & X.dbkGetMemo()
    End If
    X.dbkClose()
End If
```

## **dbkGetVersion() Method [Integer]**

---

Returns the version number of the time-series database.

### **Syntax**

**X.dbkGetVersion()**

### **Example**

Retrieve the version number of the database “SQLExample” on the SQL server <tsSQL>.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    If X.dbkExists("SQLExample") Then
        X.dbkActiveDatabase = "SQLExample"
        MsgBox "Version number of SQLExample: " & X.dbkGetVersion()
    End If
    X.dbkClose()
End If
```

## **dbkGroupCount() Method [Integer]**

---

Returns the number of time-series groups in the database.

### **Syntax**

**X.dbkGroupCount()**

### **Example**

Determine the number groups in the database “SQLExample” on the SQL server <tsSQL>

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    If X.dbkExists("SQLExample") Then
        X.dbkActiveDatabase = "SQLExample"
        MsgBox "Number of groups in SQLExample: " & X.dbkGroupCount()
    End If
    X.dbkClose()
End If
```

## **dbkMake() Method [Boolean]**

---

Creates a new time-series database. Note that the procedure returns “True” if the Dbank database already exists.

### **Syntax**

**X.dbkMake([dbkName As String], [Label As String])**

### **Example**

Create “SQLExample” on the SQL server <tsSQL>.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    If X.dbkMake("SQLExample") Then
        If X.dbkExists("SQLExample") Then
            MsgBox "SQLExample created successfully on server " & X.Server
        End If
    End If
End If
```

## dbkRename() Method [Boolean]

---

Renames a time-series database. Returns “True” if successful.

### Syntax

**X.dbkRename**(dbkName As String, NewName As String)

### Example

Rename “SQLExample” on the SQL server <tsSQL> to “NewSQLExample”.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    If X.dbkRename("SQLExample", "NewSQLExample") Then
        MsgBox "SQLExample was renamed successfully on server " & X.Server
    End If
    X.dbkClose()
End If
```

## dbkSeriesCount() Method [Integer]

---

Returns the number of series in the time-series database. You can include all deleted<sup>1</sup> series in this count by setting the optional “IncludeDeletedSeries” parameter to “False”.

### Syntax

**X.dbkSeriesCount**(*[IncludeDeletedSeries as Boolean=False]*)

### Example

Determine the number series in the database “SQLExample” on the SQL server <tsSQL>.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    If X.dbkExists("SQLExample") Then
        X.dbkActiveDatabase = "SQLExample"
        MsgBox "Number of series in SQLExample: " & X.dbkSeriesCount()
    End If
    X.dbkClose()
End If
```

---

<sup>1</sup> The database engine supports logically deleted time-series. Technically, such time-series still exist in the database; however, they are no longer visible to the user.

## **dbkTruncate() Method [Integer]**

---

Empties a Dbank database by initializing its internal tables. The internal tables of the database are retained. Thus, user rights and privileges are not affected by **dbkTruncate()**.

### **Syntax**

**X.dbkTruncate**(*[dbkName]* , *[Query as Boolean=False]*)

### **Example**

Initialize the database "SQLExample" on the SQL server <tsSQL>.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    If X.dbkExists("SQLExampe") Then
        X.ActiveDatabase = "SQLExample"
        If X.dbkTruncate() Then MsgBox "SQLExample initialized!"
    End If
    X.dbkClose()
End If
```

## **dbkUseServerSideCursors() Method [Boolean]**

---

The time-series database engine does not use cursors to manipulate time-series. This is because most of its commands operate on a single time-series record. However, if cursors are required, Dbank's engine will use client-side cursors for all database operations by default. This method allows you to change the default cursor to "server-side". Note that using server-side cursors will result in a performance penalty unless the server is very powerful. Since the default cursors are client-side cursors, you can easily return to client-side cursors creating a new instance of *tsDatabase*.

### **Syntax**

Call **X.dbkUserServerSideCursors**

### **Example**

Use server-side cursors for all database operations (at the cost of slower performance)

```
Dim X as New tsDatabase
Call X.UseServerSideCursors
X.Server = "tsSQL"
If X.Connect() Then
    X.ActiveDatabase = "SQLExample"
End If
```

## **grpCanCreate() Method [Boolean]**

---

This method returns “True” if the user can create groups in the active database.

### **Syntax**

Call **X.grpCanCreate()**

### **Example**

Test whether the user can create groups in the active database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.ActiveDatabase = "SQLExample"
    If X.grpCanCreate() Then
        MsgBox X.ActiveUser() & " can create groups."
    End If
End If
```

## **grpCanDelete() Method [Boolean]**

---

This method returns “True” if the user can delete groups in the active database.

### **Syntax**

Call **X.grpCanDelete()**

### **Example**

Test whether the user can delete groups in the active database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.ActiveDatabase = "SQLExample"
    If X.grpCanDelete() Then
        MsgBox X.ActiveUser() & " can delete groups."
    End If
End If
```

## **grpCanUpdate() Method [Boolean]**

---

This method returns “True” if the user can update group attributes in the active database.

### **Syntax**

Call **X.grpCanUpdate()**

### **Example**

Test whether the user can update groups in the active database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.ActiveDatabase = "SQLExample"
    If X.grpCanUpdate() Then
        MsgBox X.ActiveUser() & " can update groups."
    End If
End If
```

## **grpChangeLabel() Method [Boolean]**

---

Changes the label (i.e., short descriptor) of a time-series group or container. Returns “True” if successful.

### **Syntax**

**X.grpChangeLabel**( FullyQualifiedGroupName as String, NewLabel as String )

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>. Change the label of [ace] to “Hello World”, and then close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If Not X.grpChangeLabel("[ace]", "Hello World") Then
        MsgBox "Failed to change label of [ace] to ""Hello World"" "
    End If
    X.dbkClose()
End If
```

## **grpChangeMemo() Method [Boolean]**

---

Changes the memo (i.e., long descriptor) of a time-series group or container. Returns “True” if successful.

### **Syntax**

**X.grpChangeMemo**( FullyQualifiedGroupName as String, NewMemo as String )

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>. Change the memo of the [ace] group to “This can be very long”, and then close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.grpChangeMemo("[ace]", "This can be very long") Then
        MsgBox "Changed label of [ace] successfully"
    End If
    X.dbkClose()
End If
```

## **grpDelete() Method [Boolean]**

---

Deletes an empty group or container (i.e., a container that does not contain any series). Returns “True” if successful.

### **Syntax**

**X.grpDelete**( FullyQualifiedGroupName as String )

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>. Delete the group [make.information], and then close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.grpDelete("[make.information]") Then
        MsgBox "Deleted [make.information] successfully"
    End If
    X.dbkClose()
End If
```

## **grpExists() Method [Boolean]**

---

Tests whether a group or container exists in a time-series database. Returns “True” if successful.

### **Syntax**

**X.grpExists**( FullyQualifiedGroupName as String )

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>. Test whether the group [make.math] exist, and then close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.grpExists("[make.math]") Then
        MsgBox "[make.math] exists in " & X.dbkActiveDatabase
    End If
    X.dbkClose()
End If
```

## grpFindAllSeries() Method [Boolean]

---

This method builds an array containing the names of all the series that belong to a group or container. Optionally, the array can include the names of the deleted series. Note that you can also retrieve the attributes of the time-series by providing a container (or array) for these attributes. Method returns “True” if successful (i.e., no error occurs).

### Syntax

**X.grpFindAllSeries**(GroupName, SeriesNames(), SeriesCount, [IncludeDeletedSeries],[ArrayOfSeriesAttributes])

The **grpFindAllSeries** method has these arguments:

<u>Part</u>	<u>Type</u>	<u>Description</u>
<i>GroupName</i>	String	Fully qualified group name
<i>SeriesNames()</i>	Variant	Zero-based Variant Array of Series Names
<i>SeriesCount</i>	Long	Number of sub-groups in the group
<i>[IncludeDeletedSeries]</i>	Boolean	Determines whether deleted series are included in the SeriesNames() array.
<i>[Attributes]</i>	Variant	Zero-based variant array for time-series attributes

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>. Determine all the sub-groups in the time-series container [make.math], and then close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    Dim SeriesNames() as Variant
    Dim SeriesCount as Long
    Dim j as Long
    If X.grpFindAllSeries("[make.math]", SeriesNames(), SeriesCount) Then
        For j = 0 to SeriesCount-1
            MsgBox SeriesNames(j)
        Next j
    End If
    X.dbkClose()
End If
```

## grpFindAllSubGroups() Method [Boolean]

---

Dbank supports a tree directory structure much like Microsoft Windows. This method builds an array containing the names of all the sub groups that belong to a group or container. Method returns “True” if successful (i.e., no error occurs).

### Syntax

**X.grpFindAllSubGroups**(GroupName, SubGroupNames(),Count, ReturnFullNames)

The **grpFindAllSubGroups** method has these parts:

<u>Part</u>	<u>Type</u>	<u>Description</u>
<i>Name</i>	String	Fully qualified group name (which must exist)
<i>SubGroupNames()</i>	Variant	Zero-based Variant Array of Group Names
<i>Count</i>	Long	Number of sub-groups in the group
<i>[ReturnFullNames]</i>	Boolean	Optional parameter; determines whether the array of group names contains the fullyqualified group name.

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>. Determine all the sub-groups in the [make], and then close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    Dim SubGroupNames() as Variant
    Dim Count as Long
    If X.grpFindAllSubGroups("[make]", SubGroupNames(), Count, False) Then
        For j = 0 to Count-1
            MsgBox SubGroupNames(j)
        Next j
    End If
    X.dbkClose()
End If
```

## grpFindTree() Method [Boolean]

---

Dbank supports a tree directory structure much like Microsoft Windows. This method builds an array containing the names of all the sub groups that belong to a group or container. In contrast to **grpFindAllSubGroups()**, this method is recursive. It therefore allows you to enumerate the directory tree of the time-series database, starting from the named group (which, of course, can be the root). Method returns “True” if successful (i.e., no error occurs).

### Syntax

X.**grpFindTree**(GroupName, GroupNames(), Count)

The **grpFindTree** method has these parts:

<u>Part</u>	<u>Type</u>	<u>Description</u>
<i>Name</i>	String	Fully qualified group name (which must exist)
<i>GroupNames()</i>	Variant	Zero-based Variant Array of Group Names
<i>Count</i>	Long	Number of sub-groups in the group

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>. Enumerate the database tree, and then close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    Dim GroupNames() as Variant
    Dim Count as Long
    If X.grpFindTree("",GroupNames(),Count) Then
        For j = 0 to Count-1
            MsgBox GroupNames(j)
        Next j
    End If
    X.dbkClose()
End If
```

## **grpGetLabel() Method [Boolean]**

---

Returns the label (i.e., short description) of a time-series group or container.

### **Syntax**

**X.grpGetLabel**( FullyQualifiedGroupName as String )

### **Example**

Retrieve the label of the group [ace] in “SQLExample” on the SQL server <tsSQL>.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    If X.dbkExists("SQLExampe") Then
        X.dbkActiveDatabase = "SQLExample"
        MsgBox "Label of SQLExample: " & X.dbkGetLabel("[ace]")
    End If
    X.dbkClose()
End If
```

## **grpGetMemo() Method [String]**

---

Returns the memo (i.e., long description) of a time-series group or container.

### **Syntax**

**X.grpGetMemo**( FullyQualifiedGroupName as String )

### **Example**

Retrieve the memo of the group [ace] in “SQLExample” on the SQL server <tsSQL>

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    If X.dbkExists("SQLExampe") Then
        X.dbkActiveDatabase = "SQLExample"
        MsgBox "Memo of SQLExample[ace] " & X.grpGetMemo("[ace]")
    End If
    X.dbkClose()
End If
```

## grpIsEmpty() Method [Boolean]

---

Tests whether a group or container is empty (i.e., does not contain any series, including deleted series). Returns “True” if the group is empty.

### Syntax

**X.grpIsEmpty**( FullyQualifiedGroupName as String )

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>. Test whether the group [make.math] is empty, and then close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.grpIsEmpty("[make.math]") Then
        MsgBox "[make.math] is not empty."
    End If
    X.dbkClose()
End If
```

## grpMake() Method [Boolean]

---

Creates a time-series group or container. Returns “True” if successful or the group already exists.

### Syntax

**X.grpMake**(FullyQualifiedGroupName as String, *[GroupLabels as Variant]*)  
*[GroupLabels can be either a string or an array of strings.]*

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>. Create a group [ace.a.b.c], and then close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.grpMake("[ace.a.b.c]", "My parent is [ace.a.b]") Then
        If X.grpExists("[ace.a.b.c]") Then
            MsgBox "Created [ace.a.b.c] successfully!"
        End If
    End If
    X.dbkClose()
End If
```

## grpMove() Method [Boolean]

---

Moves a group to another group. The target group cannot exist. Returns “True” if successful.

### Syntax

**X.grpMove**(GroupName as String, NewLocation as String, [*NewLabel*])

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>. Create a group [ace.a.b.c], and then move the last group to “d”, creating a group called [d.c]. Close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.grpMake("[ace.a.b.c]", "My parent is [ace.a.b]") Then
        If X.grpExists("[ace.a.b.c]") Then
            If X.grpMove("[ace.a.b.c]", "d") Then
                If X.grpExists("[d.c]") Then
                    MsgBox "Moved [ace.a.b.c] successfully!"
                End If
            End If
        End If
    End If
    X.dbkClose()
End If
```

## **grpRename() Method [Boolean]**

---

Renames a group. The target group cannot exist (i.e., you cannot specify a group name that already exists). Returns “True” if successful.

### **Syntax**

**X.grpRename**(GroupName as String, NewName as String)

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>. Create a group [ace.a.b.c], and then rename the last group to “d”. Close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.grpMake("[ace.a.b.c]", "My parent is [ace.a.b]") Then
        If X.grpExists("[ace.a.b.c]") Then
            If X.grpRename("[ace.a.b.c]", "d") Then
                If X.grpExists("[ace.a.b.d]") Then
                    MsgBox "Renamed [ace.a.b.c] successfully!"
                End If
            End If
        End If
    End If
    X.dbkClose()
End If
```

## grpSeriesCount() Method [Long]

---

Returns the number of series in the group. The count does not include deleted series (if any). Setting the optional “IncludeDeletedSeries” parameter to “True” will change this behavior.

### Syntax

**X.grpSeriesCount**(GroupName as String, [*IncludeDeletedSeries=False*])

### Example

Determine the number of series in the group [ace] in “SQLExample” on the SQL server <tsSQL>

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    If X.dbkExists("SQLExampe") Then
        X.dbkActiveDatabase = "SQLExample"
        MsgBox X.grpSeriesCount("[ace]")
    End If
    X.dbkClose()
End If
```

## grpSubGroupCount() Method [Long]

---

Returns the number of sub-groups in the specified group.

### Syntax

**X.grpSubGroupCount**(GroupName as String)

### Example

Determine the number of series in the group [ace] in “SQLExample” on the SQL server <tsSQL>

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    If X.dbkExists("SQLExampe") Then
        X.dbkActiveDatabase = "SQLExample"
        MsgBox X.grpSubGroupCount("[ace]")
    End If
    X.dbkClose()
End If
```

## grpXDelete() Method [Boolean]

---

Deletes all the series in a group and its sub-groups. Optionally removes all the empty groups. Returns “True” if successful. Please note that this method destroys all the deleted series in the group.

### Syntax

**X.grpXDelete**(GroupName As String, [*RemoveEmptyGroups=False*])

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>. Delete all the series in the database, remove all the empty groups, and close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.grpXDelete("",True) Then
        MsgBox "Emptied" & X.dbkActiveDataBase
    End If
    X.dbkClose()
End If
```

## tsCanCreate() Method [Boolean]

---

This method returns “True” if the user can create time series in the active database.

### Syntax

Call **X.tsCanCreate**()

### Example

Test whether the user can create time-series in the active database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.ActiveDatabase = "SQLExample"
    If X.tsCanCreate() Then
        MsgBox X.ActiveUser() & " can create time-series."
    End If
End If
```

## **tsCanDelete() Method [Boolean]**

---

This method returns “True” if the user can delete time series in the active database.

### **Syntax**

Call **X.tsCanDelete()**

### **Example**

Test whether the user can delete time-series in the active database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.ActiveDatabase = "SQLExample"
    If X.tsCanDelete() Then
        MsgBox X.ActiveUser() & " can delete time-series."
    End If
End If
```

## **tsCanUpDate() Method [Boolean]**

---

This method returns “True” if the user can update time series in the active database.

### **Syntax**

Call **X.tsCanUpDate()**

### **Example**

Test whether the user can update time-series in the active database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.ActiveDatabase = "SQLExample"
    If X.tsCanUpDate() Then
        MsgBox X.ActiveUser() & " can update time-series."
    End If
End If
```

## **tsChangeConversionMethod() Method [Boolean]**

---

Updates the conversion method of a time series. Returns “True” if successful.

### **Syntax**

**X.tsChangeConversionMethod**(FullyQualifiedTimeSeriesName As Variant,  
NewConversionMethod as Variant)

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>, and change the conversion method of “[ace]a” to “Last”. Alternatively, for the second argument of the function you may specify an enum value (see Table 1) that represents a particular conversion method.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsChangeConversionMethod("[ace]a", tsLast) Then
        MsgBox "Changed conversion method of [ace]a to "Last""
    End If
    X.dbkClose()
End If
```

Table 1: Valid Conversion Methods for Time Series

<b>Enum Representation</b>	<b>Value</b>	<b>Code</b>	<b>Description</b>
tsAverage	1	A	Average
tsFirst	2	F	First
tsMidPoint	3	M	Midpoint
tsLast	4	L	Last
tsSum	5	S	Sum
tsNoMethod	6	N	None
tsMinimum	7	I	Minimum
tsMaximum	8	X	Maximum
tsFirstValid	9	B!	First (Ignore Missing)
tsLastValid	10	E!	Last (Ignore Missing)
tsRange	11	R	Range
tsCount	12	C	Count
tsVariance	13	V	Variance
tsStDev	14	S	Standard Deviation
tsAverageIgnoreMissing	21	A!	Average (Ignore Missing)
tsFirstIgnoreMissing	22	F!	First (Ignore Missing)
tsMidPointIgnoreMissing	23	M!	Midpoint (Ignore Missing)
tsLastIgnoreMissing	24	L!	Last (Ignore Missing)
tsSumIgnoreMissing	25	S!	Sum (Ignore Missing)
tsNoMethodIgnoreMissing	26	N!	None (Ignore Missing)
tsMinimumIgnoreMissing	27	I!	Minimum (Ignore Missing)
tsMaximumIgnoreMissing	28	X!	Maximum (Ignore Missing)
tsFirstValidIgnoreMissing	29	B!	First (Ignore Missing)
tsLastValidIgnoreMissing	30	E!	Last (Ignore Missing)
tsRangeIgnoreMissing	31	R!	Range (Ignore Missing)
tsCountIgnoreMissing	32	C!	Count (Ignore Missing)
tsVarianceIgnoreMissing	33	V!	Variance (Ignore Missing)
tsStDevIgnoreMissing	34	D!	Standard Deviation (Ignore Missing)

**Note:** *tsFirstValid = tsFirstIgnoreMissing; tsLastValid = tsLastIgnoreMissing.*

## tsChangeCost() Method [Boolean]

---

This time-series attribute provides a means of recording the monetary cost of downloading the time-series from the server. The tsChangeCost() method updates this value, which must be greater than or equal to zero. It is measured in cents. Returns “True” if successful.

### Syntax

**X.tsChangeCost**(FullyQualifiedTimeSeriesName As Variant, NewCost as Variant)

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>, and change the cost of downloading “[ace]a” to one dollar. Alternatively, for the second argument of the function you may specify an enum value (see Table 1) that represents a particular conversion method.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsChangeCost("[ace]a", 100) Then
        MsgBox "Changed cost of [ace]a to 1 dollar."
    End If
    X.dbkClose()
End If
```

## tsChangeDataSource() Method [Boolean]

---

This method updates the data source attribute of a time series. Returns “True” if successful.

### Syntax

**X.tsChangeDataSource**(FullyQualifiedTimeSeriesName As Variant, NewDataSource as Variant)

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>, and change the data source of [ace]a to “IMF”.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsChangeDataSource("[ace]a", "IMF") Then
        MsgBox "Data source of [ace]a changed to IMF."
    End If
    X.dbkClose()
End If
```

## **tsChangeDecimals() Method [Boolean]**

---

This method updates the decimal setting of a time-series. Returns “True” if successful. The valid range for decimal settings is [0,15].

### **Syntax**

**X.tsChangeDecimals**(FullyQualifiedTimeSeriesName As Variant, NewSetting as Variant)

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>, and change the decimal setting to 12.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsChangeDecimals("[ace]a", 12) Then
        MsgBox "Decimal setting of [ace]a changed to 12."
    End If
    X.dbkClose()
End If
```

## **tsChangeFiscalLead() Method [Boolean]**

---

This method updates the fiscal lead setting of a time-series. Returns “True” if successful. Fiscal lead may be set to any valid integer.

### **Syntax**

**X.tsChangeFiscalLead**(FullyQualifiedTimeSeriesName As Variant, NewFiscalLead as Variant)

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>, and change the fiscal lead to 12.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsChangeFiscalLead("[ace]a", 12) Then
        MsgBox "Fiscal lead setting of [ace]a changed to 12."
    End If
    X.dbkClose()
End If
```

## **tsChangeIgnoreMissingSeries() Method [Boolean]**

---

This method updates the “**IgnoreMissingSeries**” attribute of a time-series. It provides a means of controlling the default operation of Dbank’s Series Make module (or equation parser). Setting “**IgnoreMissingSeries**” to True instructs Series Make to ignore missing series (i.e., do not flag an error if the time-series referred to does not exist in any make operation, effectively replacing it with a vector of zeros or ones (depending on the operator). This attribute is used only if the time-series is derived from a make expression, and returns “True” if successful.

### **Syntax**

**X.tsChangeIgnoreMissingSeries**(FullyQualifiedTimeSeriesName As Variant,  
*[NewSetting as Boolean=False]*)

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>, and change the “IgnoreMissingSeries” attribute of the time-series [ace]a to “True”.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsChangeIgnoreMissingSeries("[ace]a", True) Then
        MsgBox " Missing series will be ignored for [ace]a."
    End If
    X.dbkClose()
End If
```

## **tsChangeIgnoreMissingValues() Method [Boolean]**

---

This method updates the “**IgnoreMissingValues**” attribute of a time-series. It provides a means of controlling the default operation of Dbank’s Series Make module (or equation parser). Setting “**IgnoreMissingValue**” to True instructs Series Make to ignore missing values during any make operation that involves an arithmetic operator (akin to Microsoft Excel’s behavior when adding or subtracting blank spreadsheet cells). This attribute is referenced only if the time-series is derived from a make expression, and returns “True” if successful.

### **Syntax**

**X.tsChangeIgnoreMissingValue**(FullyQualifiedTimeSeriesName As Variant,  
*[NewSetting as Boolean=False]*)

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>, and change the “IgnoreMissingValues” attribute of the time-series [ace]a to “True”.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsChangeIgnoreMissingValues("[ace]a", True) Then
        MsgBox " Missing values will be ignored for [ace]a."
    End If
    X.dbkClose()
End If
```

## tsChangeInterpolationMethod() Method [Boolean]

---

Updates the default interpolation method of a time-series. The default interpolation method is used to expand a time-series to a greater frequency (e.g., from annual to monthly). Returns “True” if successful.

### Syntax

**X.tsChangeInterpolationMethod**(FullyQualifiedTimeSeriesName As Variant, NewInterpolationMethod as Variant)

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>, and change the interpolation method of “[ace]a” to “Repeat (Last)”. Alternatively, for the second argument of the function you may specify an enum value (see Table 2) that represents a particular interpolation method.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsChangeInterpolationMethod("[ace]a", tsRepeat) Then
        MsgBox "Changed conversion method of [ace]a to ""Repeat (Last)""
    End If
    X.dbkClose()
End If
```

**Table 2: Supported Interpolation Methods**

Value	Enum	Code	Position	Description
0	tsExpand	Expand (Last)	Last	Expand the series without interpolation, actual data values positioned at the end of the sample
1	tsSpline	Spline (Last)	Last	Cubic spline interpolation, actual data values positioned at the end of the period
2	tsLinear	Linear (Last)	Last	Linear interpolation, actual data values positioned at the end of the period
3	tsRepeat	Repeat (Last)	Last	Interpolation by repeating available values, actual data values positioned at the end of the period

4	tsProRate	ProRate (Last)	Last	Interpolation by prorating known values, actual data values positioned at the end of the period
5	tsGeometric	Geometric (Last)	Last	Geometric interpolation, actual values positioned at the end of the period
6	tsPolyNomial	Polynomial (Last)	Last	Polynomial interpolation, actual values positioned at the end of the period
20	tsExpandLocBeg	Expand (Beginning)	Beginning	Expand the series without interpolation, actual data values positioned at the start of the sample
21	tsSplineLocBeg	Spline (Beginning)	Beginning	Cubic spline interpolation, actual data values positioned at the start of the period
22	tsLinearLocBeg	Linear (Beginning)	Beginning	Linear interpolation, actual data values positioned at the start of the period
23	tsRepeatLocBeg	Repeat (Beginning)	Beginning	Interpolation by repeating available values, actual data values positioned at the start of the period
24	tsProrateLocBeg	ProRate (Beginning)	Beginning	Interpolation by prorating known values, actual data values positioned at the start of the period
25	tsGeometricLocBeg	Geometric (Beginning)	Beginning	Geometric interpolation, actual values positioned at the start of the period
26	tsPolyNomialLocBeg	Polynomial (Beginning)	Beginning	Polynomial interpolation, actual values positioned at the start of the period
40	tsExpandLocMidPoint	Expand (Centered)	Mid Point	Expand the series without interpolation, actual data values

				positioned in the middle of the period
41	tsSplineLocMidPoint	Spline (Centered)	Mid Point	Cubic spline interpolation, actual data values positioned at the middle of the period
42	tsLinearLocMidPoint	Linear (Centered)	Mid Point	Linear interpolation, actual data values positioned at the middle of the period
43	tsRepeatLocMidPoint	Repeat (Centered)	Mid Point	Interpolation by repeating available values, actual data values positioned at the middle of the period
44	tsProrateLocMidPoint	ProRate (Centered)	Mid Point	Interpolation by prorating known values, actual data values positioned at the middle of the period
45	tsGeometricLocMidPoint	Geometric (Centered)	Mid Point	Geometric interpolation, actual values positioned at the middle of the period
46	tsPolynomialLocMidPoint	Polynomial (Centered)	Mid Point	Polynomial interpolation, actual values positioned at the middle of the period
99	tsNoInterpolationMethod	None	None	Do not interpolate
100	tsSplineForcedAverage	Spline(Last, ForcedAverage)	Last	Cubic spline interpolation, actual data values positioned at the end of the period, and force the average of interpolated values to equal available value for the period
101	tsSplineLocBegForcedAverage	Spline(Beginning, ForcedAverage)	Beginning	Cubic spline interpolation, actual data values positioned at the beginning of the period, and force the average of interpolated values to equal available value for the period

102	tsSplineLocMidPointForcedAverage	Spline(Centered, ForcedAverage)	Mid Point	Cubic spline interpolation, actual data values positioned in the middle of the period, and force the average of interpolated values to equal available value for the period
110	tsPolyNomialForcedAverage	Polynomial(Last, ForcedAverage)	Last	Polynomial interpolation, actual values positioned at the end of the period, and force the average of interpolated values to equal available value for the period
111	tsPolyNomialLocBegForcedAverage	Polynomial(Beginning, ForcedAverage)	Beginning	Polynomial interpolation, actual values positioned at the start of the period, and force the average of interpolated values to equal available value for the period
112	tsPolynomialLocMidPointForcedAverage	Polynomial(Centered, ForcedAverage)	Mid Point	Polynomial interpolation, actual values positioned in the middle of the period, and force the average of interpolated values to equal available value for the period
120	tsGeometricForcedAverage	Geometric(Last, ForcedAverage)	Last	Geometric interpolation, actual values positioned at the end of the period, and force the average of interpolated values to equal available value for the period
121	tsGeometricLocBegForcedAverage	Geometric(Beginning, ForcedAverage)	Beginning	Geometric interpolation, actual values positioned in the beginning of the period, and force the average of interpolated values to equal available value for the period

122	tsGeometricLocMidPointForcedAverage	Geometric(Centered, ForcedAverage)	Mid Point	Geometric interpolation, actual values positioned in the middle of the period, and force the average of interpolated values to equal available value for the period
200	tsLinearForcedAverage	Linear>Last, ForcedAverage)	Last	Linear interpolation, actual values positioned at the end of the period, and force the average of interpolated values to equal available value for the period
201	tsLinearLocBegForcedAverage	Linear(Beginning, ForcedAverage)	Beginning	Linear interpolation, actual values positioned in the beginning of the period, and force the average of interpolated values to equal available value for the period
202	tsLinearLocMidPointForcedAverage	Linear(Centered, ForcedAverage)	Mid Point	Linear interpolation, actual values positioned in the middle of the period, and force the average of interpolated values to equal available value for the period

## tsChangeLongName() Method [Boolean]

---

This method updates the “long name” of a time-series. It provides a means of attaching a more descriptive name to the time series. Returns “True” if successful.

### Syntax

**X.tsChangeLongName**(FullyQualifiedTimeSeriesName As Variant, NewLongName as Variant)

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>, and change the long name of the time-series [ace]a to “aggregate\_value”.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsChangeLongName("[ace]a", "aggregate_value") Then
        MsgBox "Long name of [ace]a changed to aggregate_value."
    End If
    X.dbkClose()
End If
```

## tsChangeMemo() Method [Boolean]

---

This method updates the “memo” of a time-series. It provides a means of attaching lengthy notes/descriptions to time-series. Returns “True” if successful.

### Syntax

**X.tsChangeMemo**(FullyQualifiedTimeSeriesName As Variant, NewMemo as Variant)

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>, and change the memo entry of the time-series [ace]a to “This item can be very long.”

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsChangeMemo("[ace]a", "This item can be very long.") Then
        MsgBox "Memo of [ace]a changed to aggregate_value."
    End If
    X.dbkClose()
End If
```

## tsChangeRefreshOnRead() Method [Boolean]

---

This method updates the “**RefreshOnRead**” attribute of a time-series. It provides a means of controlling the default read operation of Dbank’s Series Make module (or equation parser). Setting “**RefreshOnRead**” to True instructs Dbank to re-compute the time-series before it is returned to the user. It can also be described as dynamic remake. Setting this attribute to “True” reduces Dbank’s speed, but is also obviates the need to periodically recomputed the entire database to reflect the impact of data edits. This attribute is referenced only if the time-series is derived from a make expression, and returns “True” if successful.

### Syntax

**X.tsChangeRefreshOnRead**(FullyQualifiedTimeSeriesName As Variant, [*NewSetting as Boolean=False*])

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>, and change the “RefreshOnRead” attribute of the time-series [ace]a to “True”.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsChangeRefreshOnRead("[ace]a", True) Then
        MsgBox "[ace]a will be recomputed every time it is read."
    End If
    X.dbkClose()
End If
```

## **tsChangeSigDigits() Method [Boolean]**

---

This method updates the significant digits setting of a time-series. Returns “True” if successful. The valid range for significant digits is [1,16].

### **Syntax**

**X.tsChangeSigDigits**(FullyQualifiedTimeSeriesName As Variant, NewSetting as Variant)

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>, and change the significant digits setting to 5.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsChangeSigDigits("[ace]a", 5) Then
        MsgBox "Significant digit setting of [ace]a changed to 5."
    End If
    X.dbkClose()
End If
```

## **tsChangeTitle() Method [Boolean]**

---

This method updates the “title” (or short description) of a time-series. It is a descriptive attribute, meaning that it does not influence the manner in which the time-series is manipulated. Returns “True” if successful.

### **Syntax**

**X.tsChangeTitle**(FullyQualifiedTimeSeriesName As Variant, NewTitle as Variant)

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>, and change the title of the time-series [ace]a to “Random, Normal Variates”.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsChangeTitle("[ace]a", "Random, Normal Variates") Then
        MsgBox "Title of [ace]a changed to Random, Normal Variates."
    End If
    X.dbkClose()
End If
```

## tsChangeTrimBeforeSave() Method [Boolean]

---

This method updates the “**TrimBeforeSave**” attribute of a time-series. It provides a means of controlling the default operation of Dbank’s Series Make module (or equation parser). Setting “**TrimBeforeSave**” to True instructs Series Make to remove leading and trailing missing values before saving the result of a Series Make. This attribute is referenced only if the time-series is derived from a make expression, and returns “True” if successful.

### Syntax

**X.tsChangeTrimBeforeSave**(FullyQualifiedTimeSeriesName As Variant, [*NewSetting as Boolean=False*])

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>, and change the “TrimBeforeSave” attribute of the time-series [ace]a to “True”.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsChangeTrimBeforeSave("[ace]a", True) Then
        MsgBox "Leading/trailing missing values will be ignored:[ace]a."
    End If
    X.dbkClose()
End If
```

## tsChangeUnits() Method [Boolean]

---

This method updates the “units” of a time-series. It provides a means of describing the time-series units of measurement. It is a descriptive attribute, meaning that it does not influence the manner in which the time-series is manipulated. Returns “True” if successful.

### Syntax

**X.tsChangeUnits**(FullyQualifiedTimeSeriesName As Variant, NewUnits as Variant)

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>, and change the units of the time-series [ace]a to “Billions”

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsChangeUnits("[ace]a", "Billions.") Then
        MsgBox "Units of [ace]a changed to Billions."
    End If
    X.dbkClose()
End If
```

## tsCopy() Method [Boolean]

---

Copies a single time-series to another time-series or database. Returns “True” if successful.

### Syntax

**X.tsCopy**(FullyQualifiedTimeSeriesName As String, FullyQualifiedTarget as String, [*OverWriteTargetIfNecessary=True*])

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>. Copy the time-series [ace]a to [copy]ts, and close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsCopy("[ace]a", "[copy]ts") Then
        MsgBox "Successfully copied [ace]a to [copy]ts"
    End If
    X.dbkClose()
End If
```

## tsDelete() Method [Boolean]

---

Deletes a time-series. This method performs a “soft” (or logical) delete of the time-series; tsDelete() returns “True” if successful.

### Syntax

**X.tsDelete**(TimeSeries As String, [*SoftDelete=True*])

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>. Delete the time-series [ace]c, and then close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsDelete("[ace]c") Then
        MsgBox "Successfully deleted [ace]c {soft-delete}"
        If X.tsDelete("[ace]c", False) Then
            MsgBox "Successfully removed [ace]c from the database."
        End If
    End If
    X.dbkClose()
End If
```

## tsDeleted() Method [Boolean]

---

Tests whether a time-series has been deleted. Returns “True” if this is the case. Please note that this procedure will return “False” if a time-series does not physically exist in a database.

### Syntax

**X.tsDeleted**(TimeSeries As String)

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>. Test whether the time-series [ace]c has been deleted, and then close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsDeleted("[ace]c") Then
        MsgBox "[ace]c has been deleted, but still resides in SQLExample"
    End If
    X.dbkClose()
End If
```

## tsExists() Method [Boolean]

---

Tests whether a time-series exists in the active database, returning “True” if this is the case. Optionally, you can ignore the deleted flag to test for the existence of a deleted series<sup>2</sup>.

### Syntax

**X.tsExists**(TimeSeries As String, [*IgnoreDeletedFlag=False*])

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>. Test whether the time-series [ace]d exists, and then close the active database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsExist("[ace]d") Then
        MsgBox "Time-series [ace]d exists in " & X.dbkActiveDataBase
    End If
    X.dbkClose()
End If
```

---

<sup>2</sup> Time-series are not physically removed from the database. They can therefore “exist” and yet be deleted.

## **tsFindAllLinks() Method [Boolean]**

---

Determines all the link-variables that point to the named time-series.

**X.tsFindAllLinks**(FullyQualifiedTimeSeries, LinkNames(), Count)

The **tsFindAllLinks** method has these parts:

<u>Part</u>	<u>Type</u>	<u>Description</u>
<i>TimeSeries</i>	String	Fully qualified time-series name
<i>LinkNames()</i>	Variant	Zero-based Variant Array of Link Names
<i>Count</i>	Long	Number of links

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>. Determine all the link variables that point to [ace]a.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    Dim LinkNames() as Variant
    Dim Count as Long
    If X.tsFindAllLinks("[ace]a",LinkNames(),Count) Then
        For j = 0 to Count-1
            MsgBox LinkNames(j)
        Next j
    End If
    X.dbkClose()
End If
```

## **tsGetConversionMethod() Method [Boolean]**

---

Returns the default conversion method of a time-series. By default, this method returns the string representation of the conversion method. Set the “ReturnInteger” parameter to True to return an integer/enum value.

### **Syntax**

**X.tsGetConversionMethod**(TimeSeries As String, [*ReturnInteger=False*])

### **Example**

Determine the default interpolation method of [ace]a in SQLExample.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsGetConversionMethod("[ace]a") Then
    X.dbkClose()
End If
```

## **tsGetCost() Method [Variant]**

---

Returns the cost attribute of a time-series, measured in cents per time series.

### **Syntax**

**X.tsGetCost**(TimeSeries As String)

### **Example**

Retrieve the cost of [ace]a in SQLExample.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsCost("[ace]a")
    X.dbkClose()
End If
```

## **tsGetDataSource() Method [Variant]**

---

Returns the data source setting of a time-series.

### **Syntax**

**X.tsGetDataSource**(TimeSeries As String)

### **Example**

Retrieve the data source of [ace]a in SQLExample.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsDataSource("[ace]a")
    X.dbkClose()
End If
```

## **tsGetDecimals() Method [Variant]**

---

Returns the decimals setting of a time-series.

### **Syntax**

**X.tsGetDecimals**(TimeSeries As String)

### **Example**

Retrieve the decimals setting of [ace]a in SQLExample.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsGetDecimals("[ace]a")
    X.dbkClose()
End If
```

## tsGetFiscalLead() Method [Variant]

---

Returns the fiscal lead setting of a time-series.

### Syntax

**X.tsGetFiscalLead**(TimeSeries As String)

### Example

Retrieve the fiscal lead setting of [ace]a in SQLExample.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsGetFiscalLead("[ace]a")
    X.dbkClose()
End If
```

## tsGetFrequency() Method [Variant]

---

Returns the frequency of a time-series. By default, this method returns the string representation of the frequency. Set the "ReturnInteger" parameter to "True" to return an integer/enum value.

### Syntax

**X.tsGetFrequency**(TimeSeries As String, [*ReturnInteger=False*])

### Example

Retrieve the frequency of [ace]a in SQLExample.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsGetFrequency("[ace]a")
    X.dbkClose()
End If
```

## **tsGetInterpolationMethod() Method [Variant]**

---

Returns the default conversion method of a time-series. By default, this method returns the string representation of the conversion method. Set the “ReturnInteger” parameter to “True” to return an integer/enum value.

### **Syntax**

**X.tsGetInterpolationMethod**(TimeSeries As String, [*ReturnInteger=False*])

### **Example**

Retrieve the default interpolation method of [ace]a in SQLExample.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsGetInterpolationMethod("[ace]a")
    X.dbkClose()
End If
```

## **tsGetLongName() Method [Variant]**

---

Returns the long name attribute of a time-series.

### **Syntax**

**X.tsGetLongName**(TimeSeries As String)

### **Example**

Retrieve the long name of [ace]a in SQLExample.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsGetLongName("[ace]a")
    X.dbkClose()
End If
```

## **tsGetMakeIgnoreMissingSeries () Method [Boolean]**

---

Returns the “**IgnoreMissingSeries**” setting of a time series, which influences the behavior of Dbank’s Series Make.

### **Syntax**

**X.tsGetMakeIgnoreMissingSeries**(TimeSeries As String)

### **Example**

Retrieve the “ignore missing series” setting of [ace]a in SQLExample.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsGetMakeIgnoreMissingSeries("[ace]a")
    X.dbkClose()
End If
```

## **tsGetMakeIgnoreMissingValues () Method [Boolean]**

---

Returns the “**IgnoreMissingValues**” setting of a time series, which influences the behavior of Dbank’s Series Make.

### **Syntax**

**X.tsGetMakeIgnoreMissingValues**(TimeSeries As String)

### **Example**

Retrieve the missing value setting of [ace]a in SQLExample.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsGetMakeIgnoreMissingSeries("[ace]a")
    X.dbkClose()
End If
```

## **tsGetMakeRefreshOnRead () Method [Boolean]**

---

Returns the “**RefreshOnRead**” setting of a time series, which influences the behavior of Dbank when reading a time series.

### **Syntax**

**X.tsGetMakeIgnoreMissingValues**(TimeSeries As String)

### **Example**

Retrieve the missing value settings of [ace]a in SQLExample.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsGetMakeIgnoreMissingSeries("[ace]a")
    X.dbkClose()
End If
```

## **tsGetMakeTrimBeforeSave () Method [Boolean]**

---

Returns the “**MakeTrimBeforeSave**” attribute of a time series. This attribute influences the behavior of Series Make when saving a time series.

### **Syntax**

**X.tsGetMakeIgnoreMissingValues**(TimeSeries As String)

### **Example**

Retrieve the missing value settings of [ace]a in SQLExample.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsGetMakeIgnoreMissingSeries("[ace]a")
    X.dbkClose()
End If
```

## **tsGetMakeString() Method [Variant]**

---

Returns the make string attribute of a time-series.

### **Syntax**

**X.tsGetMakeString**(TimeSeries As String)

### **Example**

Retrieve the make string attached to [ace]a in SQLExample.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsGetMakeString("[ace]a")
    X.dbkClose()
End If
```

## **tsGetMemo() Method [Variant]**

---

Returns the memo of a time-series.

### **Syntax**

**X.tsGetMemo**(TimeSeries As String)

### **Example**

Retrieve the memo entry attached to [ace]a in SQLExample.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsGetMemo("[ace]a")
    X.dbkClose()
End If
```

## **tsGetSigDigits() Method [Variant]**

---

Returns the significant digits setting of a time-series.

### **Syntax**

**X.tsGetSigDigits**(TimeSeries As String)

### **Example**

Retrieve the significant digits attribute of [ace]a in SQLExample.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsGetSigDigits("[ace]a")
    X.dbkClose()
End If
```

## **tsGetTitle() Method [Variant]**

---

Returns the title of a time-series.

### **Syntax**

**X.tsGetTitle**(TimeSeries As String)

### **Example**

Retrieve the title of [ace]a in SQLExample.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsGetTitle("[ace]a")
    X.dbkClose()
End If
```

## tsGetUnits() Method [Variant]

---

Returns the units of a time-series.

### Syntax

**X.tsGetUnits**(TimeSeries As String)

### Example

Retrieve the units of [ace]a in SQLExample.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsGetUnits("[ace]a")
    X.dbkClose()
End If
```

## tsGetConversionMethod() Method [Boolean]

---

Returns the default conversion method of a time-series. By default, this method returns the string representation of the conversion method. Set the "ReturnInteger" parameter to True to return an integer/enum value.

### Syntax

**X.tsGetConversionMethod**(TimeSeries As String, [*ReturnInteger=False*])

### Example

Determine the default interpolation method of [ace]a in SQLExample.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsGetConversionMethod("[ace]a") Then
    X.dbkClose()
End If
```

## **tsIsALink() Method [Boolean]**

---

Indicates whether a time-series is a link variable. Returns “True” if this is the case.

### **Syntax**

**X.tsIsALink**(TimeSeries As String)

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>. Test whether the time-series [ace]d is a link variable, and then close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsIsALink("[ace]d") Then
        MsgBox "Time-series [ace]d is a link-variable."
    End If
    X.dbkClose()
End If
```

## **tsKey() [Long]**

---

Returns the unique integer identity key (always positive) of the time-series. This key uniquely identifies a time-series

### **Syntax**

**X.tsKey**(TimeSeries As String)

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>. Determine the unique key of the time-series, and then close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsKey("[ace]d")
    X.dbkClose()
End If
```

## **tsLastRevised() Method [Date]**

---

Returns the revision date (always positive) of the time-series by re-reading it directly from the time-series record (i.e., not from memory). This method is guaranteed to return the actual revision date of a time-series.

### **Syntax**

**X.tsLastRevised**(TimeSeries As String)

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>. Determine the most current revision date the time-series, and then close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox Format(X.tsLastRevised("[ace]d"),"dddd tttt")
    X.dbkClose()
End If
```

## **tsLastRevisor() Method [Date]**

---

Returns the user-id of the last user who revised the time-series by re-reading it directly from the time-series record.

### **Syntax**

**X.tsLastRevisor**(TimeSeries As String)

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>. Determine the most current revision date the time-series, and then close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    MsgBox X.tsLastRevisor("[ace]d")
    X.dbkClose()
End If
```

## tsLinkName() Method [String]

---

Returns the name of actual time-series variable that a link variable points to. Note that even though a link variable can point to another link variable, this method will always return the fundamental (or non-linked) time-series name. Returns a blank string if the time-series is not a link variable.

### Syntax

**X.tsLinkName**(TimeSeries As String)

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>. Test whether the time-series [ace]d is a link variable, and then display the name of the time-series it points to.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsIsALink("[ace]d") Then
        MsgBox "[ace]d points to " & X.tsLinkName("[ace]d")
    End If
    X.dbkClose()
End If
```

## tsLinkTo() Method [Boolean]

---

Creates a link-variable or pointer to a time-series. A link variable is essentially another name for a time-series. The linked time-series “points” to another time-series, which could be a link itself. By working through the links, the fundamental time-series can be retrieved.

When an application accesses a link-variable, Dbank always retrieves the fundamental time-series rather than the link-variable itself.

### Syntax

**X.tsLinkTo**(TimeSeriesName As String, LinkName As String, [*NewTitle*])

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>. Create a link variable [link]g that points to [ace]a. Test the pointer to ensure that it points to the correct place.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsLinkTo("[ace]a",[link]g") Then
        If X.tsIsALink("[link]g") Then
            MsgBox "[link]g points to " & X.tsLinkName("[link]g")
        End If
    End If
    X.dbkClose()
End If
```

## tsMove() Method [Boolean]

---

Moves a time-series to another group. The destination group need not exist; however, it must reside in the same database.

### Syntax

**X.tsMove**(SeriesName as String, TargetGroup as String)

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>. Move [ace]a to [app], then close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsExists("[ace]a") Then
        If X.tsMove("[ace]a", "[app]") Then
            If X.tsExists("[app]a") Then
                MsgBox "Moved [ace]a to [app]a successfully!"
            End If
        End If
    End If
    X.dbkClose()
End If
```

## **tsRead() Method [Boolean]**

---

Reads a time-series from the named database into a Dbank “dbTimeSeries” time-series object. Returns “True” if successful.

### **Syntax**

**X.tsRead**(SeriesName as String, TimeSeriesObject as dbTimeSeries,  
*[SkipObservationRead=False]*)

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>. Read the time-series [ace]a, display its title, and then close the database.

```
Dim X as New tsDatabase
Dim Z as New dbTimeSeries
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsRead("[ace]a",Z) Then
        MsgBox "Read [ace]a successfully!; its title is " & Z.Title
    End If
End If
X.dbkClose()
```

## tsRename() Method [Boolean]

---

Renames a time-series. The resulting series must reside in the same group.

### Syntax

**X.tsRename**(SeriesName as String, NewName as String)

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>. Rename the series [ace]a to [ace]a1, then close the database.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsExists("[ace]a") Then
        If X.tsRename("[ace]a", "a1") Then
            If X.tsExists("[ace]a1") Then
                MsgBox "Renamed [ace]a to [ace]a1 successfully!"
            End If
        End If
    End If
End If
X.dbkClose()
End If
```

## tsSave() Method [Boolean]

---

Saves a dbTimeSeries object to the named database. Returns "True" if successful.

### Syntax

**X.tsSave**(TimeSeries, SeriesObject, [*AllowedToOverWrite=True*], [*UpdateAttributesOnly=False*])

The **tsSave** method has these parts:

<u>Part</u>	<u>Type</u>	<u>Description</u>
<i>TimeSeriesName</i>	String	Fully qualified time-series name
<i>SeriesObject()</i>	Variant	Zero-based Variant Array of Link Names
<i>AllowedToOverWrite</i>	Boolean	Overwrites Existing Series (True)
<i>UpdateAttributesOnly</i>	Boolean	Only Updates the time-series attributes.

### Example

'Construct a time-series using dbTimeSeries

```
Dim X as New dbTimeSeries
X.Frequency = "Q"
X.Nobs = 100
X.StartDate = "1961:1"
For j = 1 to X.Nobs
    X.DataValue(j) = j
Next j
```

'Now saved it to tsSQL

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsSave("[ace]trend") Then
        MsgBox "Saved [ace]trend successfully!"
    End If
    X.dbkClose()
End If
```

## tsUnDelete() Method [Boolean]

---

Recovers a deleted time-series.

### Syntax

**X.tsUnDelete**(TimeSeries As String)

### Example

Set the active database to "SQLExample" on the SQL server <tsSQL>. Delete the time-series [ace]c, and then restore it.

```
Dim X as New tsDatabase
X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.tsDelete("[ace]c") Then
        MsgBox "Successfully deleted [ace]c {soft-delete}"
        If X.tsUnDelete("[ace]c") Then
            MsgBox "Successfully recovered [ace]c."
        End If
    End If
    X.dbkClose()
End If
```

## **fnChangeDate() Method [Boolean]**

---

Changes the date serial of a footnote entry.

### **Syntax**

**X.fnChangeDate**(TimeSeriesName, TimeSeriesFrequency, FootNoteName, tsDate, NewtsDate)

The **fnChangeDate** method has these parts:

<u>Part</u>	<u>Type</u>	<u>Description</u>
<i>TimeSeriesName</i>	Variant	Fully qualified time-series name
<i>TimeSeriesFrequency</i>	Variant	Time-series frequency
<i>FootNoteName</i>	String	Name/tag of footnote entry
<i>tsDate</i>	Variant	Dbank date string/date serial
<i>NewtsDate</i>	Variant	Revised Dbank date string/date serial

**fnChangeDate()** returns True if successful; False otherwise.

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>. Move the footnote entry called “A” (attached to [first]bpimr) to “1992:4”.

```
Dim X as New tsDatabase

X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.fnChangeDate("[first]bpimr", "Q", "A", "1990:1", "1992:4") Then
        MsgBox "Successfully moved footnote entry ""A"" to 1992:4"
    End If
    X.dbkClose()
End If
```

## fnCount() Method [Long]

---

Returns the number of footnotes attached to a particular series for a given date [*the latter argument being optional*].

### Syntax

X.**fnCount**(TimeSeriesName, TimeSeriesFrequency, [*tsDate*])

The **fnCount** method has these parts:

<u>Part</u>	<u>Type</u>	<u>Description</u>
<i>TimeSeriesName</i>	Variant	Fully qualified time-series name
<i>TimeSeriesFrequency</i>	Variant	Time-series frequency
[ <i>tsDate</i> ]	Variant	Dbank date string/date serial

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>. Return the number of footnote entries attached to “[first]bpimr”, a quarterly series, in 1990:1.

```
Dim X as New tsDatabase
```

```
X.Server = "tsSQL"  
If X.Connect() Then  
    X.dbkActiveDatabase = "SQLExample"  
    MsgBox X.fnCount("[first]bpimr", "Q", "1990:1")  
    X.dbkClose()  
End If
```

*Note: if “1990:1” is omitted from the above function call, the footnote count will include all the footnotes attached to the named time-series.*

## **fnDelete() Method [Boolean]**

---

Deletes a footnote entry from the database.

### **Syntax**

**X.fnDelete**(TimeSeriesName, TimeSeriesFrequency, FootNoteName, tsDate)

The **fnDelete** method has these parts:

<u>Part</u>	<u>Type</u>	<u>Description</u>
<i>TimeSeriesName</i>	Variant	Fully qualified time-series name
<i>TimeSeriesFrequency</i>	Variant	Time-series frequency
<i>FootNoteName</i>	String	Name/tag of footnote entry
<i>tsDate</i>	Variant	Dbank date string/date serial

**fnDelete()** returns True if successful; False otherwise.

### **Example**

Set the active database to "SQLExample" on the SQL server <tsSQL>. Delete the footnote entry called "A" for "[first]bpimr", a quarterly series, in 1990:1.

```
Dim X as New tsDatabase
```

```
X.Server = "tsSQL"  
If X.Connect() Then  
    X.dbkActiveDatabase = "SQLExample"  
    If X.fnDelete("[first]bpimr", "Q", "A", "1990:1") Then  
        MsgBox "Successfully deleted ""A"""  
    End If  
    X.dbkClose()  
End If
```

## **fnDeleteAll() Method [Boolean]**

---

Permanently removes *all* (irrespective of their frequency) the footnote entries attached to a time-series.

### **Syntax**

X.**fnDeleteAll**(TimeSeriesName)

The **fnDeleteAll** method has these parts:

<u>Part</u>	<u>Type</u>	<u>Description</u>
<i>TimeSeriesName</i>	Variant	Fully qualified time-series name

**fnDeleteAll()** returns True if successful; False otherwise.

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>. Delete all the footnote entries for “[first]bpimr”.

```
Dim X as New tsDatabase

X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.fnDeleteAll("[first]bpimr") Then
        MsgBox "Successfully deleted all the footnotes attached to
[first]bpimr"
    End If
    X.dbkClose()
End If
```

## **fnExists() Method [Boolean]**

---

Tests whether a footnote entry exists.

**fnExists**(TimeSeriesName, TimeSeriesFrequency, FootNoteName, [*tsDate*])

The **fnExists** method has these parts:

<u>Part</u>	<u>Type</u>	<u>Description</u>
<i>TimeSeriesName</i>	Variant	Fully qualified time-series name
<i>TimeSeriesFrequency</i>	Variant	Time-series frequency
<i>FootNoteName</i>	String	Name/tag of footnote entry
[ <i>tsDate</i> ]	Variant	Dbank date string/date serial

**fnExists()** returns True if successful; False otherwise.

### **Example**

Set the active database to “SQLExample” on the SQL server <tsSQL>. Test whether a footnote entry called “A” for “[first]bpimr”, a quarterly series, exists for “1990:1”

```
Dim X as New tsDatabase
```

```
X.Server = "tsSQL"  
If X.Connect() Then  
    X.dbkActiveDatabase = "SQLExample"  
    If X.fnExists("[first]bpimr", "Q", "A", "1990:1") Then  
        MsgBox "Footnote entry ""A"" exists!"  
    End If  
    X.dbkClose()  
End If
```

*Note: if “1990:1” is omitted in the above example, the function returns “True” if there is at least one footnote entry in 1990:1.*

## fnGet() Method [Boolean]

---

Retrieves footnote entries. You can control the number of footnotes returned using the function's optional parameters. Function returns a 2 dimensional variant array containing the footnotes and their associated attributes (see description of *Result()* array immediately below and the example).

**X.fnGet**(TimeSeriesName, Result(), fnCount, [*TimeSeriesFrequency*], [*FootNoteName*], [*tsDate*])

The **fnGet()** method has these parts:

<u>Part</u>	<u>Type</u>	<u>Description</u>
<i>TimeSeriesName</i>	Variant	Fully qualified time-series name
<i>fCount</i>	Long	Number of footnote strings (set by fnGet())
<i>Result()</i>	Variant	(0 to 5, 0 to fCount-1) array of footnote entries
[ <i>TimeSeriesFrequency</i> ]	Variant	Time-series frequency (optional)
[ <i>FootNoteName</i> ]	String	Footnote name/tag (optional).
[ <i>tsDate</i> ]	Variant	Dbank date string/date serial (optional)

### Example

Set the active database to "SQLExample" on the SQL server <tsSQL>. Retrieve all the footnote entries for "[first]bpimr", a quarterly series, in 1990:1.

```
Dim X as New tsDatabase
Dim Result() as Variant
Dim fCount as Long
Dim p as Long

X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.fnGet("[first]bpimr", Result(), "Q",,fCount) And fcount > 0
    Then
        For p = 0 to fCount-1
            MsgBox Result(0,p) 'Footnote Name
            MsgBox Result(1,p) 'Footnote Frequency
            MsgBox Result(2,p) 'Footnote Date
            MsgBox Result(3,p) 'Footnote Description
            MsgBox Result(4,p) 'Date/time footnote was created (VB date serial)
            MsgBox Result(5,p) 'Date/time footnote was last updated (VB date serial)
        Next p
    End If
    X.dbkClose()
End If
```

## fnRename() Method [Boolean]

---

Renames a footnote entry.

### Syntax

X.**fnRename**(TimeSeriesName, TimeSeriesFrequency, FootNoteName, tsDate, NewName)

The **fnRename** method has these parts:

<u>Part</u>	<u>Type</u>	<u>Description</u>
<i>TimeSeriesName</i>	Variant	Fully qualified time-series name
<i>TimeSeriesFrequency</i>	Variant	Time-series frequency
<i>FootNoteName</i>	String	Footnote name/tag.
<i>tsDate</i>	Variant	Dbank date string/date serial
<i>NewName</i>	String	New Footnote name/tag

**fnRename()** returns True if successful; False otherwise.

### Example

Set the active database to "SQLExample" on the SQL server <tsSQL>. Rename the footnote entry called "A" (attached to [first]bpimr) to "B", a quarterly series, in 1990:1.

```
Dim X as New tsDatabase
```

```
X.Server = "tsSQL"  
If X.Connect() Then  
    X.dbkActiveDatabase = "SQLExample"  
    If X.fnRename("[first]bpimr", "Q", "A", "1990:1", "B") Then  
        MsgBox "Successfully renamed footnote entry ""A"" to ""B"""  
    End If  
    X.dbkClose()  
End If
```

## fnSave() Method [Boolean]

---

Saves a footnote entry.

The **fnSave()** method has these parts:

**X.fnSave**(TimeSeriesName, TimeSeriesFrequency, FootNoteName, tsDate, FootNoteItem)

Part	Type	Description
<i>TimeSeriesName</i>	Variant	Fully qualified time-series name
<i>TimeSeriesFrequency</i>	Variant	Time-series frequency
<i>FootNoteName</i>	String	Footnote name/tag.
<i>tsDate</i>	Variant	Dbank date string/date serial
<i>FootNoteItem</i>	String	Footnote description

### Example

Set the active database to “SQLExample” on the SQL server <tsSQL>. Attach a footnote entry called “A” to “[first]bpimr”, a quarterly series, in 1990:1. The description is “Hello World...”

```
Dim X as New tsDatabase
```

```
X.Server = "tsSQL"  
If X.Connect() Then  
    X.dbkActiveDatabase = "SQLExample"  
    If X.fnSave("[first]bpimr", "Q", "A", "1990:1", "Hello World...")  
Then  
    MsgBox "Successfully saved ""A"""  
    End If  
    X.dbkClose()  
End If
```

## **fnXDelete() Method [Boolean]**

---

Permanently removes all footnote entries attached to a particular date.

### **Syntax**

**X.fnXDelete**(TimeSeriesName, TimeSeriesFrequency, tsDate)

The **fnXDelete** method has these parts:

<u>Part</u>	<u>Type</u>	<u>Description</u>
<i>TimeSeriesName</i>	Variant	Fully qualified time-series name
<i>TimeSeriesFrequency</i>	Variant	Time-series frequency
<i>tsDate</i>	Variant	Dbank date string/date serial

**fnXDelete()** returns True if successful; False otherwise.

### **Example**

Set the active database to "SQLExample" on the SQL server <tsSQL>. Delete all the footnote entries attached to "[first]bpimr", a quarterly series, for 1990:1.

```
Dim X as New tsDatabase

X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.fnXDelete("[first]bpimr", "Q", "1990:1") Then
        MsgBox "Successfully deleted all the 1990:1 footnotes for
[first]bpimr."
    End If
    X.dbkClose()
End If
```

## fnXDeleteAll() Method [Boolean]

---

Permanently removes all footnote entries *of a given frequency* attached to a particular series.

### Syntax

X.**fnXDeleteAll**(TimeSeriesName, TimeSeriesFrequency)

The **fnXDeleteAll** method has these parts:

<u>Part</u>	<u>Type</u>	<u>Description</u>
<i>TimeSeriesName</i>	Variant	Fully qualified time-series name
<i>TimeSeriesFrequency</i>	Variant	Time-series frequency

**fnXDeleteAll**() returns True if successful; False otherwise.

### Example

Set the active database to "SQLExample" on the SQL server <tsSQL>. Delete all the footnote entries attached to "[first]bpimr", a quarterly series.

```
Dim X as New tsDatabase

X.Server = "tsSQL"
If X.Connect() Then
    X.dbkActiveDatabase = "SQLExample"
    If X.fnXDeleteAll("[first]bpimr", "Q") Then
        MsgBox "Successfully deleted all the footnotes attached to
[first]bpimr"."
    End If
    X.dbkClose()
End If
```

