Afaria® Reference Manual
APIs

Version 6.5

SYBASE®

Afaria Reference Manual | APIs 6.5

Document version 6.50.00

Contents

Preface

This guide is intended for the person responsible for installing and maintaining the Afaria Server. We recommend that you have a working knowledge of the Windows operating system and its conventions, SQL databases, Microsoft Internet Information Server (IIS), and a directory manager such as LDAP or NT. You will also need a working knowledge of the client types you plan to support.

Afaria Support Services

Product technical support: www.sybase.com/support or frontline.sybase.com/support

Americas and Asia-Pacific call support:

(678) 585-7320 Atlanta, Georgia

(800) 669-1211 Toll-free

European call support:

+44 (0) 1628 50 5321 United Kingdom

0825 800372 Toll-free

1

Server Automation

Using Server and Client Automation, you can programmatically control and monitor Server and Client applications and events. You can automate your system using compiled languages such as C/C++, interpretive languages such as Visual Basic and Visual Basic for Applications, and scripting languages such as VBScript and JavaScript.

This guide is the essential reference for developers and covers important technical programming information, including sample code, documentation, technical articles, and anything else you might need to develop solutions using Server and Client Automation.

How Automation Works

The information and examples in this guide use OLE Automation as it was implemented before changes were made in Visual Studio .NET. For examples on using OLE Automation in Visual Studio .NET, refer to the Visual Studio .NET documentation.

Objects are the fundamental building blocks of Server Automation; nearly everything you do in the product involves manipulating objects. Every unit of content and functionality—each worklist, channel, log, and so on—is an object that one can control programmatically. You must have a basic understanding of the following concepts:

- Automation – Automation is a service for integrating development tools and applications. It enables an application to expose its functionality, or to control the functionality of other applications on the same computer, or across networks. As a result of using Automation, applications can be automated and integrated with programming code.
- Automation Servers and Object Models – Applications or software components, called Automation servers, can be controlled because their functionality has been exposed and made accessible to other applications. Examples of Automation servers are all Microsoft Office applications, Microsoft Schedule+, and Microsoft Project. These Automation servers expose their functionality through object models.
- Automation controllers – Other applications or development tools, called Automation controllers, can control Automation servers through programming code by accessing the functionality exposed by the Automation servers. Examples of Automation controllers are Microsoft Visual Basic, Microsoft Visual C++, Microsoft Visual FoxPro, and Microsoft Visual Basic for Applications, which is built into Microsoft Access, Microsoft Excel, and Microsoft Project.

  Automation is the umbrella term for the process by which an Automation controller sends instructions to an Automation server (using the functionality exposed by the Automation server), where the instructions are executed.

  Some applications are both Automation controllers and Automation servers.

Automation Controllers

The term automation controller refers to anything that controls an Automation server. Other common names are OLE Automation controller, or OLE controller. The Automation controller can connect to a server in one of two ways: late binding or early binding.

Late Binding

Late binding declares a variable as an object or a VARIANT. You initialize the variable by calling GetObject or CreateObject and naming the Automation programmatic identifier (ProgID).

For example, if the ProgID is "Mom.ApplePie," the Visual Basic code could appear like this:

```
Dim objPie As Object

Dim objSlice As Variant

Set objPie = CreateObject("Mom.ApplePie")

Set objSlice = CreateObject("Mom.PieSlice")
```

Late binding was the first binding method implemented in controller products. Late binding is the friendly name for what C programmers call IDispatch-based binding. It uses a lot of overhead, which means it is faster than DDE, but slower than early binding. Late binding is used exclusively by Scripting languages.

Early Binding

Early binding declares a variable as an application-defined object type. Early binding is the friendly name for what C programmers call virtual function table bindings or vtable binding. You should initialize the variable with the CreateObject or GetObject commands. A type library, object library, or dynamic-link library is required to declare a variable as an application-defined object type.

```
Dim objPie As New Mom.ApplePie

Or

Dim objPie As Mom.ApplePie

Set objPie = CreateObject("Mom.ApplePie")
```

Object Models

In order to programmatically gain access to an application's content and functionality, you must understand how the content and functionality of an application is partitioned into discrete objects and how these objects are arranged in a hierarchical model.

Think of applications as consisting of content and functionality:

- *Content* refers to the data the application contains; it also refers to information about attributes of individual elements in the application, such as the size of a window or the color of an image.
- *Functionality* refers to all the ways one can work with the content in the application, such as opening, closing, adding, deleting, copying, and so on.

The content and functionality in an application are broken down into discrete units of related content and functionality called *objects*. The top-level object in an application is usually the Application object, which is the application itself. For instance, Microsoft Excel itself is the Application object in the Microsoft Excel object model. The Application object contains other objects that you have access to only when the Application object exists (that is, when the application is running). For example, the Microsoft Excel Application object contains Workbook objects, and the Word Application object contains Document objects. Because the Document object depends on the existence of the Word Application object for its own existence, the Document object is said to be the child of the Application object; conversely, the Application object is said to be the parent of the Document object.

Many objects that are children have children of their own. For example, the Microsoft Excel Workbook object contains, or is parent to, the collection of Worksheet objects that represent all the worksheets in the workbook. A parent object can have multiple children; for instance, the Word Window object has as children the Panes, Selection, and View objects. Likewise, a child object can have multiple parents; for instance, the Word Windows collection object is the child of both the Application object and the Document object.

The way the objects that make up an application are arranged relative to each other, together with the way the content and functionality are divided among the objects, is called the object hierarchy or the object model.

In addition to containing lower level objects, each object in the hierarchy contains content and functionality that apply both to the object itself and to all objects below it in the hierarchy. The higher an object is in the hierarchy, the wider the scope of its content and functionality. For example, in Microsoft Excel, the Application object contains the size of the application window and the ability to quit the application; the Workbook object contains the file name and format of the workbook and the ability to save the workbook; and the Worksheet object contains the worksheet name and the ability to delete the worksheet.

You often don't get to what you think of as the contents of a file (such as the values on a Microsoft Excel worksheet or the text in a Word document) until you've navigated through quite a few levels in the object hierarchy, because this specific information belongs to a very specific part of the application. In other words, the value in a cell on a worksheet applies only to that cell, not to all cells on the worksheet, so you cannot store it directly in the Worksheet object. The content and functionality stored in an object are thus intrinsically appropriate to the scope of the object.

In summary, the content and functionality in an application are divided among the objects in the application's object model. Together, the objects in the hierarchy contain all the content and functionality in the application. Separately, the objects provide access to very specific areas of content and functionality.

Properties and Methods

To get to the content and functionality contained in an object, you use the properties and methods of that object. The following Microsoft Excel example uses the Value property of the Range object to set the contents of cell B3 on the worksheet named "Sales" in the workbook named "Current.xls."

```
Workbooks("Current.xls").Worksheets("Sales").Range("B3").Value = 3
```

The following Word example uses the Close method of the Document object to close the file named "Draft 3.doc."

```
Documents("Draft 3.doc").Close
```

In general, you use properties to get to content, which can include the data contained in an object or the attribute settings for the object; and you use methods to get to functionality, which entails everything you can do to the content.

Beware, however, that this distinction doesn't always hold true; a number of properties and methods in every object model constitute exceptions to this rule.

Collection Objects

When using Visual Basic Help graphics to explore the object model for the application in which you want to program, you may notice that there are many boxes in the graphics that contain two words, usually the singular and plural forms of the same object name, such as "Documents (Document)" or "Workbooks (Workbook)." In these cases, the first name (usually the plural form) is the name of a collection object.

A collection object is an object that contains a set of related objects. You can work with the objects in a collection as a single group rather than as separate entities. The second name (usually the singular form), enclosed in parentheses, is the name of an individual object in the collection. For example, in Word, you can use the Documents collection to work with all the *Document* objects as a group.

Although the *Documents* collection object and the *Document* object are both objects in their own right, each with its own properties and methods, they're grouped as one unit in most object model graphics to reduce complexity. You can use a collection object to get to an individual object in that collection, usually with the *Item* method or property.

The following PowerPoint example uses the *Item* property of the *Presentations* collection object to activate the presentation named "Trade Show" and then close it. All other open presentations are left open.

```
Presentations.Item("Trade Show").Close
```

The Item property or method is the default method for most collections. Therefore, Presentations("Trade Show").Close is equivalent to the preceding example.

You can also create new objects and add them to a collection using the Add method of that collection. The following Word example creates a new document based on the Normal template.

```
Documents.Add
```

You can find out how many objects there are in the collection by using the *Count* property. The following Microsoft Excel example displays the number of open workbooks in a message box if more than three workbooks are open.

```
If Workbooks.Count > 3 Then MsgBox "More than 3 workbooks are open"
```

Collections are useful in other ways as well. For instance, you can perform an operation on all the objects in a given collection, or you can set or test a value for all the objects in the collection. To do this, you use a *For Each...Next* or *For...Next* structure to loop through all the objects in the collection.

Collection Types

There are two basic types of collections: snapshot and live.

- Snapshot – When you reference a snapshot collection, the reference reflects the contents of the collection at the time it was created. For example, suppose you have two macros (A and B). In macro A, you set a global variable to the collection, and in macro B you use the variable. When you use it, the variable will reflect the contents of the collection when it was created in macro A.
- Live – When you reference a live collection, the reference reflects the current contents of the collection. For example, suppose you have two macros (A and B). In macro A, you set a global variable to the collection, and in macro B you use the variable. When you use it, the variable will reflect the current contents of the collection.

Object Model Hierarchy

To manipulate objects in the object model hierarchy, you must know the relationships between them. You have to navigate through the object model to get to its members. This usually means that you have to step down through all the objects above it to access an item.

For example, in Microsoft Excel, you cannot get to a particular cell on a worksheet without first going through the application, which contains the workbook that contains the worksheet that contains the cell.

The following example inserts the value 3 in cell B3 on the worksheet named "Second Quarter" in the workbook named "Annual Sales.xls."

```
Application.Workbooks("Annual Sales.xls").WorkSheets("Second
Quarter").Range("B3").Value = 3
```

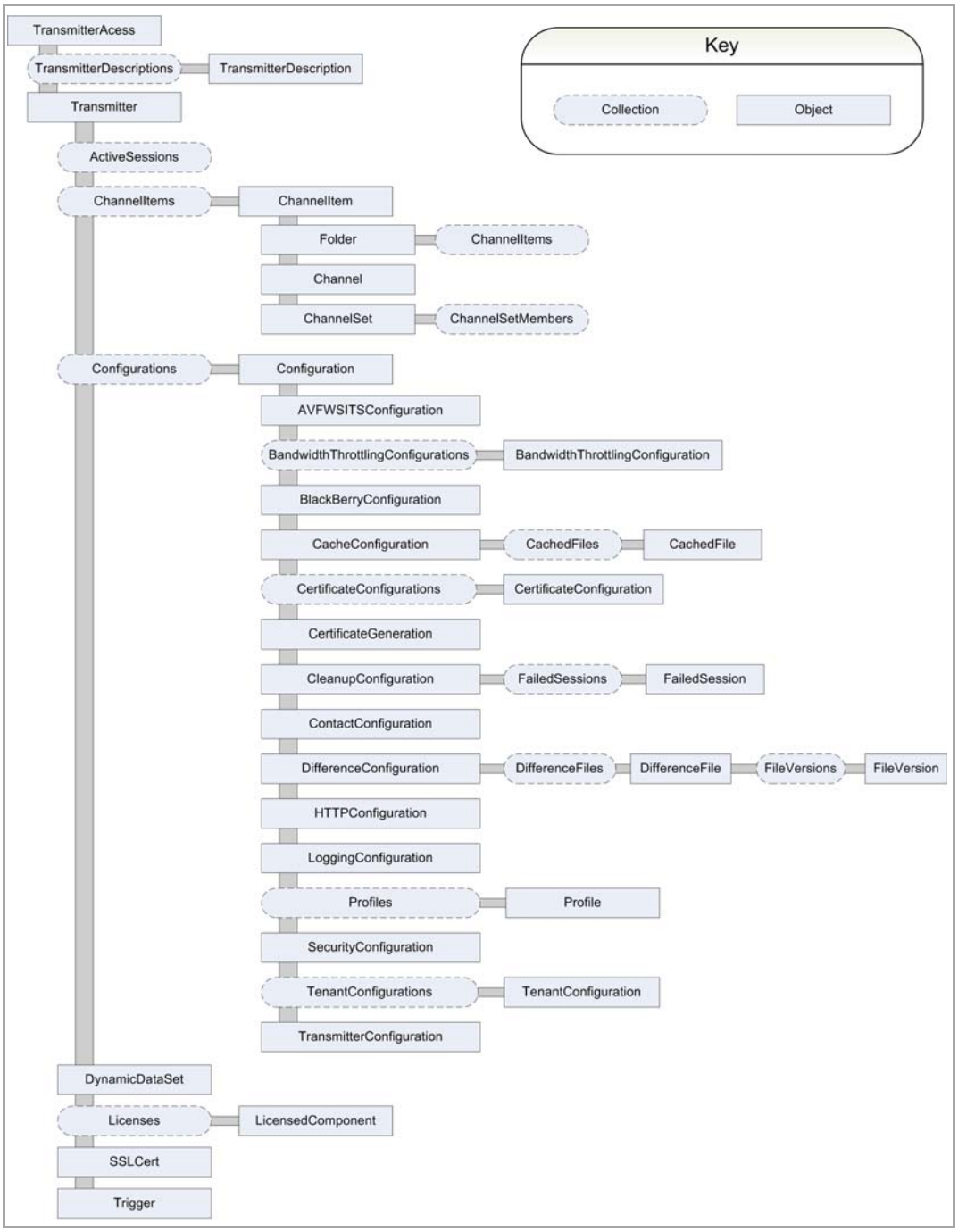
Similarly, the following Word example applies bold formatting to the second word in the third paragraph in the first open document.

```
Application.Documents(1).Paragraphs(3).Range.Words(2).Bold = True
```

In the Afaria object model hierarchy, the TransmitterAccess object is at the top, and the other objects and collections are subordinate to it. This relationship allows you to directly access objects subordinate to the TransmitterAccess object by using the properties and methods of the TransmitterAccess object.

Object Model Hierarchy Diagram

The following diagram identifies parent-child relationships for object and collections.



Automating a Task Using Objects

To automate a task using an Object Model, you first return a reference to the object that contains the content and functionality you want to get to, and then you apply properties and methods to that object.

Returning a Reference to an Object

Before you can do anything with an object, you must return a reference to the object. To do this, you must build an expression that gains access to one object in the object model and then uses properties or methods to move up or down through the object hierarchy until you get to the object with which you want to work.

Object members are the properties and methods you use to return the object from which you started and to move from one object to another. As you build an expression with members to return a reference to an object, keep the following guidelines in mind:

- To gain access to the object model, use the top-level object, which is usually the Application object. Use the Application property to return a reference to the Application object. The following expression returns a reference to the Application object (for any object library that contains an Application object).

  ```
  Application
  ```

- To drill down to an object from the top-level object in a hierarchy, you must step down through all the objects above it in the hierarchy, using members to return one object from another. For example, the Documents property of the Word Application object returns the Documents collection object, which represents all open documents. The following expression returns a reference to the Word Documents collection object.

  ```
  Application.Documents
  ```

- To return a single member of a collection, use the Item property or method with the name or index number of the member. For example, in Microsoft Excel, the following expression returns a reference to an open workbook named "Sales."

  ```
  Workbooks.Item("Sales")
  ```

- The Item property or method is the default method for most collections. Therefore, the following two expressions are equivalent.

  ```
  Workbooks.Item("Sales")
  ```

  ```
  Workbooks("Sales")
  ```

- To navigate from a higher object in the object hierarchy, use the Parent property of the object. Note that the Parent property doesn't always return the immediate parent of an object; it may return the object's "grandparent," especially if the object is a member of a collection. That is, the Parent property of an object in a collection may return the collection's parent instead of the collection itself. For example, the Parent property of a Word Document

object returns the Application object, not the Documents collection. Use the TypeName function in Visual Basic to find out to what kind of object the Parent property of an object returns a reference. For example, in Microsoft Excel, the following statement displays the type of object to which the Parent property of the Worksheet object refers.

```
MsgBox TypeName(Workbooks(1).Worksheets(1).Parent)
```

Applying Properties and Methods to an Object

After you've returned a reference to the object you want to work with it, you can apply properties and methods to the object to set an attribute for it or perform an action on it. You use the "dot" operator (.) to separate the expression that returns a reference to an object from the property or method you apply to the object.

The following example, which you can run from Microsoft Excel, Word, or PowerPoint, sets the left position of the active window by using the Left property of the Window object that the ActiveWindow property returns a reference to.

```
ActiveWindow.Left = 200
```

The following Word example closes the active document by using the Close method of the Document object to which the ActiveDocument property returns a reference.

```
ActiveDocument.Close
```

Properties and methods can take arguments that qualify how they perform. In the following Word example, the PrintOut method of the Document object that the ActiveDocument property returns a reference to takes arguments that specify the range of pages it should print.

```
ActiveDocument.PrintOut From:="3", To:="7"
```

You may have to navigate through several layers in an object model to get to what you consider the real data in the application, such as the values in cells on a Microsoft Excel worksheet or the text in a Word document. The following Word example uses the following properties and methods to navigate from the top of the object model to the text of a document:

- *Application property* returns a reference to the Application object.
- *Documents property of the Application object* returns a reference to the Documents collection.
- *Item method of the Documents collection* returns a reference to a single Document object.
- *Words property* of the Document object returns a reference to the Words collection.
- *Item method of the Words collection* returns a reference to a single Range object.
- *Text property of the Range object* sets the text for the first word of the document.

```
Application.Documents.Item(1).Words.Item(1).Text = "This is a test!"
```

Sample Programming – Visual Basic and Visual C++

Automation makes it easy for interpretive and scripting languages to access COM components. While Automation is easier for the interpretive and macro programmer, it requires more work for the C++ developer.

The examples below show how to perform similar tasks using both Visual Basic and Visual C++.

Example 1

Visual Basic

```
Dim Foo as Object
Set Foo = CreateObject( "Test.foo" )
Foo.Test1
```

Visual C++

```
HRESULT hr = OleInitialize(NULL);
wchar_t progid[] = L"Test.foo" ;
CLSID clsid ;
CLSIDFromProgID( progid, &clsid ) ;
IFoo * pFoo = NULL ;
CoCreateInstance( clsid, NULL, CLSCTX_INPROC_SERVER, IID_IFoo,
(void**)&pIFoo ) ;
pIFoo->Test1() ;
pIFoo->Release() ;
```

Example 2

Visual Basic

```
Dim x as Object
Set x = GetObject(, "Excel.Application" )
Set x = Nothing
```

Visual C++

```
// initialize, etc.
CLSIDFromProgID( "Excel.Application", &clsid ) ;
GetActiveObject( clsid ) ;
```

Example 3

Visual Basic

```
If Window.Visible = False Then
Window.Visible = True
End If
```

Visual C++

```
if ( !(pIWindow->get_Visible()) )
pIWindow->put_Visible( VARIANT_TRUE ) ;
```

Example 4

Visual Basic

```
Dim x as New Foo
x.Fee.Faa.TestNum = 1234
```

Visual C++

```
// initialize, etc.
IFoo * pIFoo = CoCreateInstance( … ) ;
IFee * pIFee = NULL ;
HRESULT hr = pIFoo->get_Fee( &pIFee ) ;
IFaa * pIFaa = NULL ;
hr = pIFee->get_Faa( &pIFaa ) ;
hr = pIFaa->put_TestNum( 1234 ) ;
pIFaa->Release() ;
pIFee->Release() ;
pIFoo->Release() ;
```

Example 5

Visual Basic

```
Sub UseFoo( foo as IFoo )
     foo.x = foo.x - (foo.y * 1000)
     foo.Name = "sample"
End Sub
```

Visual C++

```
void UseFoo( IFoo * pIFoo )
{
     long x, y ;
     HRESULT hr = pIFoo->get_x( &x ) ;
     hr = pIFoo->get_y( &y ) ;
     hr = pIFoo->put_x( x - y * 1000 ) ;
     BSTR bstr = SysAllocString( L("sample") ) ;
     hr = pIFoo->put_Name( bstr ) ;
     SysFreeString( bstr ) ;
}
```

Using Arrays as Method Parameters

At the time of this writing (1999), the VBScript active scripting engine supplied by Microsoft only supported the indexing of SAFEARRAYs of VARIANTs.

While VBScript is capable of accepting arrays of non-variant type for the purposes of boundary checking and passing it to other automation objects, the engine does not allow manipulation of the array contents at this time. To function correctly with applications and components that host VBScript, Server objects package arrays as SAFEARRAYs of VARIANTs. Non-VARIANT data is packaged in the VARIANT elements of the SAFEARRAY. The SAFEARRAY itself is packaged as a VARIANT.

- Scripts written in VBScript can use the TypeName function to check the data type of a variable. The TypeName function returns the string "Variant()," excluding the quotes, when passed an array of VARIANTs.
- Scripts written in JScript (Java Script) should use the typeof operator to test the data type of a variable. The typeof operator returns the string "unknown," excluding the quotes for data types unsupported by JScript.

For example, assume the following IDL definition specifies a method that returns one or more Strings (BSTR in C++). For compatibility with scripting hosts, the Strings are packaged into a SAFEARRAY of VARIANTs of subtype String. The entire SAFEARRAY is itself packaged as a VARIANT.

```
[ propget ]
HRESULT Names( [out,retval] * VARIANT pNames ) ;
Here is an example of using this method from VBScript:
Dim names
names = object.Names()  'Get the list of names.
for each name in names
     Msgbox name
next
```

Here is an implementation of this method in C++:

```
STDMETHODIMP Foo::get_Names(VARIANT * pvaVariant)
{
     HRESULT hr = NOERROR;
     LPSAFEARRAY psa;
     SAFEARRAYBOUND rgsabound[] = { 3, 0 }; // 3 elements, 0-based
     int i;
     if (!pvaVariant) return E_INVALIDARG;
     VariantInit(pvaVariant);
```

```
    psa = SafeArrayCreate(VT_VARIANT, 1, rgsabound);
    if (!psa) return E_OUTOFMEMORY;
    VARIANT vNames[3];
    for (i = 0; i < 3; i++)
    {
      VariantInit(&vNames[i]);
      V_VT(&vNames[i]) = VT_BSTR;
    }
    V_BSTR(&vNames[0]) = SysAllocString(OLESTR("Vanilla"));
    V_BSTR(&vNames[1]) = SysAllocString(OLESTR("Chocolate"));
    V_BSTR(&vNames[2]) = SysAllocString(OLESTR("Espresso Chip"));
    if (!V_BSTR(&vNames[0])
    ||  !V_BSTR(&vNames[1])
    ||  !V_BSTR(&vNames[2]))
    {
      hr = E_OUTOFMEMORY;
      goto Error;
    }
    //Plug references to the data into the SAFEARRAY
    LPVARIANT rgElems;
    if (FAILED(hr = SafeArrayAccessData(psa,(LPVOID*)&rgElems)))
      goto Error;
    for (i = 0; i < 3; i++)
      rgElems[i] = vNames[i];
    SafeArrayUnaccessData(psa);
    V_VT(pvaVariant) = VT_ARRAY | VT_VARIANT;
    V_ARRAY(pvaVariant) = psa;
    return NOERROR;
Error:
    for (i = 0; i < 3; i++)
      if (V_BSTR(&vNames[i])
        VariantClear(&vNames[i]);
```

```
    return hr;
}
```

Additional Technical Notes

Consider the following items about the Object Model:

- Client applications written in Visual Basic must use Visual Basic 5.0 or higher.
- The object model is not thread safe.
- The operations allowed from within a callback notification (event) are restricted. See Events for more information.
- The object model is implemented as an in-proc server (a DLL). The objects are not subject to aggregation.
- Once you have an object in memory, the object stays in memory until it is released (using standard COM reference count rules). The object may therefore become out of date. For example, if you enumerate the files in the Server's compressed file cache, and you later try to delete a specific file from the cache, the cached file may no longer exist. The recommended usage pattern is to release objects as soon as possible.
- The Server object uses a significant amount of computer resources. In addition, it may take several seconds to create this object. Automation clients are advised to use a single Transmitter object that is referenced for the life of the application. The only time you should create multiple Server objects is if you need to simultaneously control multiple Servers.

2

Using Automation with Afaria

Automation (formerly referred to as OLE Automation) provides an infrastructure for calling applications to access and manipulate objects that are shared by other applications.

Using Automation with Afaria does not offer a complete replacement of the Afaria Administrator application. However you can use Automation controls from applications such as Microsoft Word, Excel, Visual Basic, or VBScript embedded in a web page running on Internet Explorer to execute some Afaria functionality.

Afaria has a variety of objects, each with shared properties and methods. Some properties are read-only, whereas others are read/write. This means you can only get the values of some properties, whereas you can get or set the values of others. Methods offer control over performing some Afaria actions.

Working with the TransmitterAccess Object

The TransmitterAccess is the top-level object in the Object Model. Use members—properties or methods—of the TransmitterAccess object to get information about Servers. You can select a specific Server by address, or use the default Server for your workstation. The Object Model is accessible only from the local server. You cannot use a Web browser or other remote means to access functionality.

After successfully connecting to the requested Server, the TransmitterAccess object returns a Server object. The Server object represents a specific Afaria Server. The Server object lets you control or return Server-wide attributes, to control the Server service, and to get to the rest of the object model.

Example – Creating and Starting the Server

The following Visual Basic example shows how to start the service by creating an instance of the Server object and applying the Start method:

```
Dim ta as Afaria.TransmitterAccess

Set ta = CreateObject("TransmitterAccess")

Dim theTransmitter

Set theTransmitter = ta.GetTransmitterFromAddress 'get default
transmitter.

theTransmitter.Start
```

The Server object uses a significant amount of computer resources. In addition, it may take several seconds to create this object. For connecting clients, you are advised to use a single Transmitter object that is referenced for the life of the application. The only time you should create multiple Transmitter objects is if you need to simultaneously control multiple Servers.

Example – Navigating the Object Model

Properties of the Server object also provide access to objects lower in the object hierarchy, such as the Configurations collection (representing all Server options and settings). You use properties, which are sometimes called accessors, to move down through the object hierarchy from the top-level Server object to the lower levels (CachedFiles, FailedSessions, and so forth). You can use the following example to clear the Server's compressed file cache:

```
Dim CacheCfg as CacheConfiguration

Set CacheCfg = Transmitter.Configurations.Item("Cache")
```

```
CacheCfg.Empty
```

Immediate Editing Versus Batch Editing

Most objects perform their operations immediately. For example, the name of a channel is changed as soon as the Channel object's Name method returns. This behavior is known as immediate editing.

For some objects, immediate editing is resource intensive, especially if many properties are changed at once. To improve performance in these situations, you should use batch editing.

To use batch editing, call the BatchEdit method on an object (not all objects support this method; use the AllowBatchEdit property to determine the availability of this feature). The BatchEdit method returns a new reference to the object. The object caches changes to this new reference. To save these changes, call the Commit method on the new reference.

- The object is locked for editing until the new reference is released (in C++, this occurs when the new object's reference count goes to zero; in Visual Basic, this occurs when the new object goes out of scope or is explicitly set to Nothing).
- Changes to data are lost if the new reference is released before calling Commit.
- For security reasons, the Server may unlock an object that has been locked for a long period of time (the timeout value is configurable via the user interface on the Server). Therefore, you should release locked objects as soon as possible; otherwise, you may get unexpected results.

For following Visual Basic example shows how to batch edit the contents of a Document Manager channel:

```
Dim channel

'Get reference to the Document Manager channel named "Employee Handbook"
located in the root folder:

set channel = transmitter.ChannelItems("\Employee Handbook")

'Get a batch edit lock on the channel content:

Dim edit

set edit = channel.Content.BatchEdit

'Start changing the channel's content properties:

edit.AllowSubscribeByFile = False

edit.HideName = True

edit.MediaLabel = "Employee Handbook"

'Save changes now:

edit.Commit

'Make some more changes:

edit.UseFileDifference = True

edit.MediaSource = "CD"
```

```
'Save new changes and release edit lock:
edit.Commit
Set edit = Nothing
```

Working with Events

An event is an action or occurrence to which a VBScript event handler or other controller can respond.

Clients receive events by implementing one or more Server outbound interfaces (the outbound interfaces are tagged with the [source] attribute in the IDL file). Although outbound interfaces are not required to be dispatch interfaces, Server outbound interfaces are always dispatch interfaces for compatibility with certain scripting environments.

Visual Basic programmers can declare variables that understand the default callback interface type as follows:

```
Dim WithEvents Trans As Afaria.Transmitter
```

In the example above, the presence of the Trans variable definition allows Visual Basic programmers to write event handlers. Visual Basic event handlers are simply functions or subroutines that use the VariableName_EventName convention. For example, to handle the OnTransmitterStateChanged callback on the preceding Trans variable, the Visual Basic programmer would write the following code:

```
Private Sub Trans_OnTransmitterStateChanged( ByVal oldState As
cmTransmitterStateEnum, ByVal newState As cmTransmitterStateEnum, ByVal
pTransmitter As CMS.Transmitter)

     ' Add code here to handle state change

End Sub
```

The Visual Basic virtual machine automatically creates an implementation of TransmitterEvents at run time, mapping the incoming method invocations onto the appropriate user-defined subroutines.

Visual C++ programmers must manually implement the TransmitterEvents interface and use standard COM connection point semantics to setup an advisory connection with the Server.

Each programming language creates instances of Server events differently. The following examples show how to create an OnTransmitterStateChanged event handler in Visual Basic and VBScript.

Example – Visual Basic

```
Option Explicit

Dim WithEvents Trans As Afaria.Transmitter

Private Sub Form_Load()

     Dim ta as Afaria.TransmitterAccess

     Set ta = CreateObject("CMS.TransmitterAccess")

     Set Trans = ta.GetTransmitterFromAddress 'get default transmitter.

     Set ta = Nothing

End Sub

Private Sub Form_Unload(Cancel As Integer)

     Set Trans = Nothing

End Sub

Private Sub Trans_OnTransmitterStateChanged(ByVal oldState As
cmTransmitterStateEnum, ByVal newState As cmTransmitterStateEnum, ByVal
pTransmitter As Transmitter)

     'Add code to handle event here.

End Sub
```

Example – VBScript

Although VBScript does not currently support events, you can use events by embedding your VBScript code on an HTML page running in Microsoft Internet Explorer.

This example creates an instance of a Transmitter object and names it "Trans." The object is identified by its class id. Next, two push buttons labeled "Start" and "Stop" are created. Pressing the Start button starts the Server service on the local machine, while pressing the Stop button stops the service.

The OnTransmitterStateChanged event handler is called whenever the state of the Server services changes. When the event fires, the event handler displays a message box showing the current and previous state of the Server service. A local helper function, getState, is used to convert the raw state values into meaningful strings for display.

Scripting environments are typeless and cannot directly access the constants and enumerated values defined in the Server type library. To work around this limitation, the Server provides the Constants object. This object contains read-only properties whose return values correspond to Afaria constants. The example code creates several global variables that hold the constant values corresponding to the state of the Server service. By using variables at run-time instead of hard-coded values, we help decouple the code from future changes in the object model.

```
<!- Create an instance of a Transmitter object -->
<object classid="clsid:9A64C803-22A2-11D3-8686-080009DC5357" id=Trans>
</object>
<!- Create two buttons to Start and Stop the Transmitter -->
<input type=button name=cmdStart value="Start">
<input type=button name=cmdStop  value="Stop">
<!- VBScript that handles the button click events and the Transmitter
events -->
<script language=vbscript>
  Option Explicit
  'Declare variables to hold values for the different Transmitter states:
  Dim tc
  Set tc = Trans.Constants
  Dim cmStateContinuePending
  Dim cmStatePaused
  Dim cmStatePausePending
  Dim cmStateRunning
  Dim cmStateStartPending
  Dim cmStateStopped
  Dim cmStateStopPending
  'Get the values for the different Transmitter states:
  cmStateContinuePending = tc.cmStateContinuePending
  cmStatePaused          = tc.cmStatePaused
  cmStatePausePending    = tc.cmStatePausePending
  cmStateRunning         = tc.cmStateRunning
  cmStateStartPending    = tc.cmStateStartPending
  cmStateStopped         = tc.cmStateStopped
  cmStateStopPending     = tc.cmStateStopPending
  sub cmdStart_onClick
    'Start the Transmitter
    Trans.Start
  end sub
```

```
  sub cmdStop_onClick
    'Stop the Transmitter
    Trans.Stop
  end sub
  sub Trans_OnTransmitterStateChanged( oldState, newState, theTrans )
    'Display the change in Transmitter state
    MsgBox "Transmitter state changed from " & _
           getState( oldState ) & " to " & getState( newState )
  end sub
  Function getState( state )
    'Returns the string representation of the Transmitter state.
    Select Case state
      Case cmStateContinuePending
        getState = "ContinuePending"
      Case cmStatePaused
        getState = "Paused"
      Case cmStatePausePending
        getState = "PausePending"
      Case cmStateRunning
        getState = "Running"
      Case cmStateStartPending
        getState = "StartPending"
      Case cmStateStopped
        getState = "Stopped"
      Case cmStateStopPending
        getState = "StopPending"
      Case Else
        getState = "unknown"
    End Select
  End Function
</script>
```

Accessing the Server Object Model

You can access the Server object model in two ways:

- Write VBScript macros that script the Server through its object model.
- In applications such as Microsoft Word, Microsoft Excel, Microsoft Visual Basic, or Microsoft Visual C++, write a controller that accesses the Server by creating an instance of it. For example, in Visual Basic, use the CreateObject function to create an instance of a Server. Or in Visual C++, call the Win32 CoCreateInstance function. For more information, see the application's documentation.

Visual Basic applications access the Server object model by using a type library, whereas C/C++ applications would typically use the object model header files.

Server Object Model Code Samples

The product image includes code samples that provide greater context than many of the samples included in this reference documentation.

On your product image, see \Samples.

Controlling Objects with Dual Interfaces

Each Server object implements a dual interface through which you can control the object. Each object implements an IDispatch interface for indirect controller access and a COM interface for direct access to object members (properties, methods, and events).

Controllers written in Visual C++ or Visual Basic versions that support early binding can bind early by using the COM interface. Early binding makes all calls into interface members faster at run time.

The following table shows the dual interface used by each Server object:

Object	*Dual Interface*
ApplicationContent	IApplicationContent
AVFWSITSConfiguration	IAVFWSITSConfiguration
CacheConfiguration	ICacheConfiguration
CachedFile	ICachedFile
CachedFiles	ICachedFiles
Channel	IChannel
ChannelItem	IChannelItem
ChannelItems	IChannelItems
ChannelSet	IChannelSet
ChannelSetMembers	IChannelSetMembers
CleanupConfiguration	ICleanupConfiguration
Configuration	IConfiguration
Configurations	IConfigurations
ContactConfiguration	IContactConfiguration
Content	IGenericContent
DifferenceConfiguration	IDifferenceConfiguration
DifferenceFile	IDifferenceFile
DifferenceFiles	IDifferenceFiles
Documents	IDocuments
FailedSession	IFailedSession
FailedSessions	IFailedSessions
FileVersion	IFileVersion
FileVersions	IFileVersions
Folder	IFolder
License	ILicense

Object	*Dual Interface*
Licenses	ILicenses
Profile	IProfiles
SendList	ISendList
Tenant	ITenantConfigurations
Transmitter	ITransmitter
TransmitterAccess	ITransmitterAccess
TransmitterConfiguration	ITransmitterConfiguration
TransmitterDescription	ITransmitterDescription
TransmitterDescriptions	ITransmitterDescriptions

Alternating Between Dual Interfaces

In Visual C++, you can switch from one dual interface to another by calling QueryInterface. For example, to switch from the COM Iconfiguration interface on a CacheConfiguration object to the COM IcacheConfiguration interface, use the following code:

```
Iconfiguration* pConfig = … // obtained elsewhere.
BSTR bstrType;
pConfig->get_Type(&bstrType)
if (!_wcscmp(bstrType, L"Cache"))
{
  // It is a cache configuration document,
  // ..so QI for the right interface
  IcacheConfiguration* pCacheConfig = 0;
  pConfig->QueryInterface(IID_IcacheConfiguration, &pCacheConfig);
  // Now, we can use cache-specific members of pCacheConfig
  pCacheConfig->set_PercentDiskSpace( 40 );
  pCacheConfig->Release();
}
SysFreeString(bstrType)
pConfig->Release();
```

The ActiveX Template Library (ATL) provides smart COM pointers that can help you. See the CComPtr and CComQIPtr classes for details.

Using Return Values from Dual Interfaces

If a property or method of a Server object returns a value of type T, then the corresponding dual interface method returns an HRESULT but accepts an additional argument of type “pointer to T” (or T*). This argument is at the end of the argument list. For example, if a property returns a value of type Long, then the corresponding dual interface method accepts an additional argument of type “pointer to Long” (or Long*).

When the method returns, the return value is stored where the last parameter points. For example, consider a method named Start that returns a Boolean value. The dual interface version of this method is declared as HRESULT Start(VARIANT_BOOL* pfStarted). The last parameter, which is declared as VARIANT_BOOL*, is where the return value is stored.

The Visual C++ code to do this looks like the following:

```
VARIANT_BOOL retVal;

if (Start(&retStatus) == S_OK)

{

  //code here could check the value of retStatus returned by Start

}
```

If the function succeeds, the HRESULT returned by the dual interface method is a success code like S_OK. If the function fails, however, the function returns an error code instead. This error code has the same value as one thrown in a dispatch exception when you call the method through the dispatch interface.

The Server error codes are in the XsAuto.h header file.

Manipulating Properties Through Dual Interfaces

To get a property of an object, prefix “get_” to the property name. Alternatively, to set the property of an object, prefix “put_” to the property name.

For example, to get the percent disk space of the CacheConfiguration object, prefix “get_” to the PercentDiskSpace property, as shown in the following code:

```
long nSize = 0;

if (pCacheConfiguration->get_PercentDiskSpace(&nSize) == S_OK)

{

  // code here references the PercentDiskSpace property through nSize.

}
```

Alternatively, to set the value of the PercentDiskSpace property to 40, you would use the following code:

```
pCacheConfiguration->put_PercentDiskSpace(40);
```

Calling Methods of Dual Interfaces

Some dual interface methods use parameters called “Reserved” of type VARIANT, and other methods use parameters that are optional. With VBScript macros or controllers authored in Visual Basic, you can omit such parameters.

However, with controllers authored in Visual C++, you must explicitly use these parameters.

For each optional VARIANT parameter, you must pass an empty VARIANT of type VT_ERROR, and you must specify the scode of the VARIANT as DISP_E_PARAMNOTFOUND. For example:

```
// C++ Syntax
_variant_t  vtEmpty (DISP_E_PARAMNOTFOUND, VT_ERROR);
pObject->Foo( vtEmpty );  // Foo takes an optional VARIANT.
```

For each optional BSTR parameters, you must pass an empty string. For example:

```
// C++ Syntax
_bstr_t     bstrEmpty(L"");
pObject->Fee( bstrEmpty );  // Fee takes an optional BSTR.
```

If you want to use the default value for a parameter, you must explicitly specify the default value. For example:

```
// C++ Syntax
pObject->Fum( -1 );  // Fum takes an optional LONG with default value of
-1.
```

The MIDL interface definition for the examples above would be similar to the following:

```
interface IObject : public IDispatch {
  HRESULT Foo( [in, optional] VARIANT v ) ;
  HRESULT Fee( [in, string, optional] BSTR s ) ;
  HRESULT Fum( [in, optional, defaultvalue(-1)] LONG value ) ;
}
```

Not all parameters use default values.

Reserved Methods

Some interfaces have methods called "Reserved1," "Reserved2," and so on.

Do not use reserved methods—they are reserved for future use.

3

Objects

The Afaria object model allows you to programmatically control and monitor Afaria applications and events. This object model consists of the hierarchy of Afaria objects, and their associated properties, methods, and events.

About objects

You can control several aspects of the Server and server operations programmatically by manipulating the server through Automation. For example, you can control the Server service by manipulating its corresponding object. To manipulate objects, you must know the relationships between them. The TransmitterAccess object is at the top, and the other objects and collections are subordinate to it. See "Object Model Hierarchy" on page 23 for a diagram of the parent-child relationships that exist between objects and collections in the object model.

Each object implements a dual interface through which you can manipulate the object. Each object implements an IDispatch interface for Automation and a Component Object Model (COM) interface for direct access to object members (properties, methods, events). An object inherits members from its parent and may introduce members of its own.

Automation servers can employ early binding by using the COM interface. Early binding makes all calls into interface members faster at run time.

The following objects are available:

Object	*Represents*
AVFWSITSConfiguration object	The Server's Antivirus/Firewall component configuration settings.
BandwidthThrottlingConfiguration object	The Server's bandwidth throttling configuration
BlackBerryConfiguration object	The Server's SMTP server configuration
CacheConfiguration object	The Server's cache configuration
CachedFile object	A single file in the Server's compressed file cache
CertificateConfiguration object	The settings and options for a Server's security certificate
CertificateGeneration object	The settings and options for a Server's Certificate generation
Channel object	A single channel
ChannelItem object	A generic channel-related item
ChannelSet object	A channel set
CleanupConfiguration object	The Server's cleanup configuration
Configuration object	One or more related options and settings for a Server
ContactConfiguration object	The Server's contact configuration
DifferenceConfiguration object	The Server's file difference configuration
DifferenceFile object	A difference file
DynamicDataSet object	A set of one or more named data values

Object	*Represents*
FailedSession object	A failed session
FileVersion object	A version of a difference file
Folder object	A folder
HTTPConfiguration object	A Server's HTTP configuration
LicensedComponent object	A particular license type
LoggingConfiguration object	The Server's logging policy settings
Profile object	A group profile
SecurityConfiguration object	The Server's security configuration
SSLCert object	The properties for a security certificate for a Server
TenantConfiguration object	A single tenant record
Transmitter object	The Server on the local machine
TransmitterAccess object	The Servers available from the local workstation
TransmitterConfiguration object	The Server's address and name configuration
TransmitterDescription object	The description for a Server that is available from the local workstation
Trigger object	The start times, repetition criteria, and other schedule-related information for a monitor

AVFWSITSConfiguration object

Inherits from parent object – Configuration object

This object represents the antivirus and firewall component settings for a Server. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
ClientApprovalDir property	N/A
ClientHoldForApproval property	
DefApprovalDir property	
DefHoldForApproval property	
SITSServerAddress property	

The object has all the members of a Configuration object plus additional members that relate to the object's unique functions.

You must stop and restart the Server to use any new settings for this configuration.

BandwidthThrottlingConfiguration object

Inherits from parent object – BandwidthThrottlingConfigurations collection

This object represents the options and settings for a Server's bandwidth throttling. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
ConfigurationSet property	**InitInstance method**
Description property	
MaximumClientThroughput property	
MinimumClientThroughput property	
ReadOnly property	
ThrottleDownPercentage property	
ThrottleDownThreshold property	
ThrottleDownWaitTime property	

The object has all the members of a Configuration object plus additional members that relate to the object's unique functions.

You must stop and restart the Server to use any new settings for this configuration.

BlackBerryConfiguration object

Inherits from parent object – Configuration object

The BlackBerryConfiguration object[1] represents the options and settings for a Server's SMTP server configuration. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
ReplyAddress property	N/A
SMTPServer property	
SMTPUserID property	
Type property	

The object has all the members of a Configuration object plus additional members that relate to the object's unique functions.

You must stop and restart the Server to use any new settings for this configuration.

1. In earlier Afaria releases, the BlackBerry Configuration object represented options and settings for configuration items relating to BlackBerry pagers, which included SMTP server configuration. The current release's BlackBerry support no longer requires the BlackBerry-specific items, but the product still supports using an SMTP server.

CacheConfiguration object

Inherits from parent object – Configuration object

This object represents the options and settings for a Server's file compression cache. The file compression cache is used to store compressed files that are frequently sent to Clients. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
CachedFiles property	**EmptyCache method**
LastRefreshTime property	**RefreshCache method**
PercentDiskSpace property	
Type property	

The object has all the members of a Configuration object plus additional members that relate to the object's unique functions.

You must stop and restart the Server to use any new settings for this configuration.

CachedFile object

Inherits from parent object – CachedFiles collection

This object represents a single file in the Server's compressed file cache. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
Compressible property	**Delete method**
FileSize property	
HitRate property	
LastAccessed property	
LastChecked property	
LastUpdated property	
SourceFileName property	
SourceFileSize property	

A file is no longer subject to compression if the attempt to compress the file fails, or the size of the compressed file is greater than or equal to the size of the original file. If a file is no longer subject to compression, the following properties of the CachedFile object are invalid: FileSize, FullName, LastAccessed, Name, Parent, Path.

The Server automatically caches compressed files sent to Clients. Files less than 16384 bytes in size are not automatically cached.

CertificateConfiguration object

Inherits from parent object – CertificateConfigurations collection

This object represents the settings and options for a Server's security certificate.The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Method*
Alg property	**InitInstance method**
CertificateFileName property	
CurveType property	
ExpDate property	
IssuerAddress property	
IssuerCommonName property	
IssuerCountry property	
IssuerLocality property	
IssuerOrgName property	
IssuerState property	
IssuerUnit property	
KeyType property	
PubKey property	
SerialNumber property	
UserAddress property	
UserCommonName property	
UserCountry property	
UserLocality property	
UserOrgName property	
UserState property	
UserUnit property	
ValidDate property	

This object has all the members of a Configuration object plus members that relate to cleanup.

You must stop and restart the Server to use any new settings for this configuration.

CertificateGeneration object

Inherits from parent object – Configuration object

This object represents the settings and options for Server certificate generation. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
CertificateDirectory property	**AssociateCertificate method**
	GenerateCertificateEx method

The object has all the members of a Configuration object plus additional members that relate to the object's unique functions.

You must stop and restart the Server to use any new settings for this configuration.

Channel object

Inherits from parent object – ChannelItem object

This object represents a channel, a conceptual unit of information or content that is centrally managed on a Server and delivered to Clients. A system channel is owned and created by the Server and is not editable. Non-system channels are created by the user and are editable. The Channel object has all the members of a ChannelItem object plus additional members that relate to channels. You can deploy channels to Clients across a local or wide-area network, the Internet or an intranet, from a Web browser, or the Channel Viewer program.

Use the Content property to get a Content object that represents a channel's content. For more information, see **ChannelItem object on page 67**. The following tables list the object's properties and methods. Click the name to view details.

Properties

Authenticate property	**HTMLControlType property**
AutoDelete property	**HTMLDisableMessages property**
AutoDeleteTime property	**HTMLHideMessages property**
AutoPublish property	**HTMLHideStatus property**
AutoPublishTime property	**InventoryOptions property**
AutoRefreshContent property	**MasterCopyID property**
AutoSubscribe property	**PasswordEncoded property**
AutoUnpublish property	**PasswordPlain property**
AutoUnpublishTime property	**PasswordProtected property**
ConfigXML property	**Published property**
ContentHomeDirectory property	**RunOnlyIfNewer property**
ContentID property	**SendEncrypted property**
ContentSize property	**VisibilityWindowBegin property**
Description property	**VisibilityWindowBeginEnabled property**
Hidden property	**VisibilityWindowEnd property**
HTMLButtonImage property	**VisibilityWindowEndEnabled property**
HTMLButtonText property	**WorkingCopyID property**
HTMLCloseImmediately property	**WorkObjectName property**

Properties

HTMLCode property

Methods

Add method (ChannelItems)
RefreshContent method
SetPublish method
SetWorklistFile method
SetWorklist method

ChannelItem object

Inherits from parent object – ChannelItems collection

This object represents a channel-related item. A channel is a conceptual unit of information or content that is centrally managed on a Server and delivered to Clients. A system channel is owned and created by the Server and is not subject to edit. Non-system channels are created and edited by the user. You can deploy channels to Clients across the Internet or an intranet, from a web browser, or directly to the Channel Viewer program at the Client. You can use folders to organize channels into groups that have related tasks or content. The root folder represents the top-level folder (its name is the same as the Server). With channel sets, you can easily and efficiently deliver multiple channels to Clients from a Web browser. Each channel contains a content item which uniquely defines the channel's behavior. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
ClassID property	**CopyToFolder method**
ID property	**CopyToFolderEx method**
LastUpdated property	**Delete method**
ParentFolder property	**MoveToFolder method**
System property	
Type property	

A ChannelItem object represents either a Channel, Folder, or ChannelSet. Use a channel item's Type property to determine which members you can access. Channel item types have all the members of a ChannelItem object and also have members that allow you to access their specific attributes. For example, if the type is "Channel Set," the channel item is a Channel Set, and you can access ChannelSet members as well as ChannelItem members.

The following table lists all channel item types and their object representations.

ChannelItem type	*Object representation*
Channel	Channel
Channel Set	ChannelSet
Folder	Folder

You can deploy channels and channel sets to Clients across the Internet or an intranet from a Web browser. To do this, you select a channel or channel set, then set one or more of its HTMLXXX properties (where XXX represents a specific HTML property, such as HTMLButtonImage).

The Server automatically generates the HTML code to run the channel or channel set on the Client from a Web page. Simply copy the generated HTML code into your Web page. The HTMLControlType property determines how the channel or channel set runs on the Client:

- **Text**. The channel appears as a hyperlink. Use the HTMLButtonText property to set the text for the hyperlink (default is the channel name).
- **Button**. The channel appears as a standard button. Use the HTMLButtonText property to set the button text (default is the channel name).
- **Image**. The channel appears as a bitmap. Use the HTMLButtonImage property to set the image path.
- **Connect On Load**. The Client immediately begins to run the channel when the Web page appears. This item does not appear on the web page.

You can also control how messages appear on the Client machine when the channel executes.

- **Hide Status**. Shows or hides the Client status window (as defined by the HTMLHideStatus property).
- **Hide Messages**. Shows or hides the Client messages window (as defined by the HTMLHideMessages property). The user can view messages by clicking the Messages button.
- **Disable Messages**. Enable or disables the Client messages window (as defined by the HTMLDisableMessages property). The user cannot view messages.

If you change any HTML property, you must change the corresponding HTML code in your Web pages.

ChannelSet object

Inherits from parent object – ChannelItem object

This object represents a set of one or more channels that are delivered to Clients from a web browser. Channel sets are a convenient way of organizing groups of related channels into a single logical channel. When Clients request a channel set (perhaps by clicking on a button on a Web page), all the channels in the channel set are delivered to the Client. A channel set can not contain references to folders or other channel sets. For more information, see ChannelItem object. The ChannelSet object has all the members of a ChannelItem object plus additional members that relate to channel sets. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
ChannelSetMembers property	N/A
HTMLButtonImage property	
HTMLButtonText property	
HTMLCloseImmediately property	
HTMLCode property	
HTMLControlType property	
HTMLDisableMessages property	
HTMLHideMessages property	
HTMLHideStatus property	

CleanupConfiguration object

Inherits from parent object – Configuration object

This object represents the settings and options for a Server's cleanup. Cleanup includes how often channel content is refreshed and how long the system should keep track of failed sessions. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
ChannelUpdateSchedule property	**ResetChannelUpdateSchedule method**
DefaultFailedSessionCleanupSchedule property	**ResetDeletedChannelCleanupSchedule method**
DeletedChannelCleanupSchedule property	**ResetFailedSessionCleanupSchedule method**
FailedSessionCleanupSchedule property	
FailedSessions property	

The object has all the members of a Configuration object plus additional members that relate to the object's unique functions.

You must stop and restart the Server to use any new settings for this configuration.

Configuration object

Inherits from parent object – Configurations collection

This object represents one or more related options and settings for a Server. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
Type property	N/A

The overall behavior of a Server is determined by a set of user-configurable options and settings. Each group of related options and settings is represented by a Configuration object.

Use a configuration's Type property to determine which members you can access. Configuration types have all the members of a Configuration object and also have members that allow you to access their specific attributes. For example, if the type is "Contact," the configuration is a contact configuration, and you can access ContactConfiguration members as well as Configuration members.

The following table lists all configuration types and their object representations.

Configuration type	*Object representation*
AVFWSITS	**AVFWSITSConfiguration object**
BandwidthThrottling	**BandwidthThrottlingConfiguration object**
BlackBerry	**BlackBerryConfiguration object**
Cache	**CacheConfiguration object**
Certificate	**CertificateConfiguration object**
CertificateGeneration	**CertificateGeneration object**
Cleanup	**CleanupConfiguration object**
Contact	**ContactConfiguration object**
Difference	**DifferenceConfiguration object**
HTTP	**HTTPConfiguration object**
Logging	**LoggingConfiguration object**
Security	**SecurityConfiguration object**
Tenant	**TenantConfiguration object**
Transmitter	**TransmitterConfiguration object**

ContactConfiguration object

Inherits from parent object – Configuration object

This object represents the settings and options for a Server's contact information. The contact information contains details on who to contact for additional information and assistance. This information is displayed in the Channel Viewer application in the Server Properties dialog under the Contact tab. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
Address property	N/A
Description property	
PhoneNumber property	

The object has all the members of a Configuration object plus additional members that relate to the object's unique functions.

You must stop and restart the Server to use any new settings for this configuration.

DifferenceConfiguration object

Inherits from parent object – Configuration object

This object represents the settings and options for a Server's file difference cache. The file difference cache is used to store different versions of files that are frequently sent to Clients. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
DifferenceFiles property	**EmptyCache method**
LastRefreshTime property	**RefreshCache method**
PercentDiskSpace property	

The object has all the members of a Configuration object plus additional members that relate to the object's unique functions.

You must stop and restart the Server to use any new settings for this configuration.

DifferenceFile object

Inherits from parent object – DifferenceFiles collection

This object represents a single file in the Server's difference file cache. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
FileSize property	**Delete method**
FileVersionCount property	
FileVersionInfo property	
FileVersions property	
FullName property	
HitRate property	
LastAccessed property	
LastChecked property	
LastUpdated property	
SourceFileName property	
SourceFileSize property	

DynamicDataSet object

Inherits from parent object – Transmitter object

This object represents a set of one or more named data values.The DynamicDataSet object is used to efficiently query for a set of one or more data values. For a list of valid data names, see the DynamicData property. Use the GetData property to get the latest values associated with the data set.

The DynamicDataSet object has the following property: **“GetData property”**

FailedSession object

Inherits from parent object – FailedSessions collection

This object represents a single failed session. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
ChannelName property	**Delete method**
ClientName property	
Date property	
UserName property	

The Server usually keeps track of failed sessions in order to perform automatic session recovery. If a communication session between a Client and Server fails, the Server stores information about the failed session for later use. If a Client connects to the Server before the session recovery timeout (specified by the FailedSessionCleanupSchedule property of the CleanupConfiguration object), the Server restarts the failed session from the point of failure. The Server deletes the automatic session recovery information once a session's recovery timeout expires.

Use the Delete method on a FailedSession object to force the Server to restart the session from the beginning for all future connections.

FileVersion object

Inherits from parent object – FileVersions collection

This object represents a particular version of a file in the Server's difference file cache. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
FileSize property	N/A
FileVersionInfo property	
SourceFileName property	
SourceFileSize property	

Folder object

Inherits from parent object – ChannelItem object

This object represents a folder, a conceptual way of organizing channel information into groups that have related tasks or content. The root folder represents the top-level folder (its name is the same as the Server). A channel or channel set is always contained within a folder (a.k.a. the parent folder). For more information, see ChannelItem object. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
ChannelItems property	N/A
Description property	
FullName property	
Hidden property	
PasswordEncoded property	
PasswordPlain property	
PasswordProtected property	
SortMode property	
VisibilityWindowBegin property	
VisibilityWindowBeginEnabled property	
VisibilityWindowEnd property	
VisibilityWindowEndEnabled property	

HTTPConfiguration object

Inherits from parent object – Configuration object

This object represents HTTP (Hypertext Transfer Protocol) settings and options for a Server. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
DefaultHTTPPort property	**ResetPort method**
EnableHTTP property	
Port property	

The object has all the members of a Configuration object plus additional members that relate to the object's unique functions.

You must stop and restart the Server to use any new settings for this configuration.

LicensedComponent object

Inherits from parent object – Licenses collection

This object represents a single licensed component.

The following table lists related properties. Click the name to view details.

Properties	*Methods*
Description property	N/A
Value property	

LoggingConfiguration object

Inherits from parent object – Configuration object

This object represents the logging policy settings for a Server. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
AlertLogSettings property	N/A
AllLogSettings property	
MsgLogSettings property	
RepLogSettings property	
SessLogSettings property	

The object has all the members of a Configuration object plus additional members that relate to the object's unique functions.

You must stop and restart the Server to use any new settings for this configuration.

Profile object

Inherits from parent object – Profiles collection

This object represents a group profile. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
AllowedChannels property	**AddAssignment method**
Assignments property	**AddChannel method**
Description property	**AddMonitorAction method**
	RemoveAllAssignments method
	RemoveAllChannels method
	RemoveAssignment method
	RemoveChannel method
	RemoveChannelByID method
	RemoveMonitorAction method

Example – Profile Object Tasks

The following example demonstrates tasks for the profile object.

```
Option explicit
Dim ta, t, prfs, prf, chanAllowed, chanAssigned, chanID, chanItems, flags,
chanItem
set ta = CreateObject("Afaria.TransmitterAccess")
set t = ta.GetTransmitterFromAddress
t.TenantName = "CJ"
serverID = t.ServerID

prfs = t.Profiles
prf = prfs.Item("CJ")
chanAllowed = prf.AllowedChannels
chanAssigned = prf.Assignments
flags = t.Constant("cmFilterByAll")
chanItems = t.ChannelItems("", flags)
chanItem = chanItems.Item("MyExistingChannel")
chanID = chanItem.ID


' ===== ADD MONITORS TO PROFILE =====
prf.AddMonitorAction "Connection Monitor", "log", "<Log additionalText=""Log
```

```
Event Only"" />", True
prf.AddMonitorAction "Directory Monitor", "ExecuteProgram", "<Program
waitForCompletion=""False"" name=""Notepad"" parameters="""" /
><Criteria><Connection type=""None"" /></Criteria><ErrorRetries count=""0""
intervalMinutes=""0"" />", False
prf.AddMonitorAction "Memory Monitor", "ExecuteScript", "<Script
waitForCompletion=""False"" engine=""JScript"" name=""Script""
function=""ScriptFunction"" parameters="""" /><Criteria><Connection
type=""None"" /></Criteria><ErrorRetries count=""0"" intervalMinutes=""0"" />",
True
prf.AddMonitorAction "Schedule Monitor", "RunChannel", "<Channel
waitForCompletion=""False"" origChannelId=""105"" origTransmitterId=""ko$o"" /
><Criteria><Connection type=""None"" /></Criteria><ErrorRetries count=""0""
intervalMinutes=""0"" />", False

' ===== REMOVE MONITORS FROM PROFILE =====
prf.RemoveMonitorAction "Connection Monitor", "log", "<Log additionalText=""Log
Event Only"" />"
prf.RemoveMonitorAction "Directory Monitor", "ExecuteProgram", "<Program
waitForCompletion=""False"" name=""Notepad"" parameters="""" /
><Criteria><Connection type=""None"" /></Criteria><ErrorRetries count=""0""
intervalMinutes=""0"" />"
prf.RemoveMonitorAction "Memory Monitor", "ExecuteScript", "<Script
waitForCompletion=""False"" engine=""JScript"" name=""Script""
function=""ScriptFunction"" parameters="""" /><Criteria><Connection
type=""None"" /></Criteria><ErrorRetries count=""0"" intervalMinutes=""0"" />"
prf.RemoveMonitorAction "Schedule Monitor", "RunChannel", "<Channel
waitForCompletion=""False"" origChannelId=""105"" origTransmitterId=""ko$o"" /
><Criteria><Connection type=""None"" /></Criteria><ErrorRetries count=""0""
intervalMinutes=""0"" />"


' ===== ASSIGN GROUP TO PROFILE =====
prf.AddAssignment "", "", "System"
prf.AddAssignment "", "ClientGroup", "Client"
prf.AddAssignment "", "Administrators" , "Local"
prf.AddAssignment "ntdomain.com", "Users", "Domain"
prf.AddAssignment "testdomain.com","OU=OrgUnit-CJ,DC=testdomain,DC=com",
"LDAPOU"
prf.AddAssignment "testdomain.com", "CN=Domain
Admins,CN=Users,DC=testdomain,DC=com", "LDAPOBJ"

' ===== REMOVE GROUP FROM PROFILE =====
prf.RemoveAllAssignments()
prf.RemoveAssignment "", "", "System"
prf.RemoveAssignment "", "ClientGroup", "Client"
prf.RemoveAssignment "", "Administrators" , "Local"
prf.RemoveAssignment "ntdomain.com", "Users", "Domain"
prf.RemoveAssignment "testdomain.com","OU=OrgUnit-CJ,DC=testdomain,DC=com",
"LDAPOU"
prf.RemoveAssignment "testdomain.com", "CN=Domain
Admins,CN=Users,DC=testdomain,DC=com", "LDAPOBJ"
```

```
' ===== ADD CHANNELS TO PROFILE =====
prf.AddChannel("\InvMgr\BB Scan")
prf.AddChannel("\InvMgr\BB HW Scan")
prf.AddChannel("\InvMgr\Palm Scan")
prf.AddChannel("\InvMgr\Palm HW Scan")
prf.AddChannel("\InvMgr\Symbian Scan")
prf.AddChannel("\InvMgr\Symbian HW Scan")
prf.AddChannel("\InvMgr\Windows Scan")
prf.AddChannel("\InvMgr\Windows HW Scan")
prf.AddChannel("\InvMgr\WMPro Scan")
prf.AddChannel("\InvMgr\WMPro HW Scan")
prf.AddChannel("\InvMgr\WMStd Scan")
prf.AddChannel("\InvMgr\WMStd HW Scan")

prf.AddChannel("\SesMgr\BB Session")
prf.AddChannel("\SesMgr\Palm Session")
prf.AddChannel("\SesMgr\Symbian Session")
prf.AddChannel("\SesMgr\Windows Session")
prf.AddChannel("\SesMgr\WMPro Session")
prf.AddChannel("\SesMgr\WMStd Session")

prf.AddChannel("\Config\BB Config")
prf.AddChannel("\Config\Palm Config")
prf.AddChannel("\Config\WMPro Config")
prf.AddChannel("\Config\WMStd Config")

' ===== REMOVE CHANNELS FROM PROFILE =====
prf.RemoveAllChannels
prf.RemoveChannel("\CJAFARIA\Config\BB Config")
prf.RemoveChannelByID (chanID)
prf.RemoveChannelByID (serverID, chanID)


MsgBox "Finished updating Profile"
```

SecurityConfiguration object

Inherits from parent object – Configuration object

This object represents the Server's security settings for user authentication.

The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
AuthenticationServer property	N/A
CertificationDatabase property	
DomainListNames property	
EnableSSL property	
EnableUserAuthentication property	
LDAPAssignmentNodeTypes property	
LDAPSearchRoot property	
SSLPort property	
UseLDAP property	
UserAssignmentTimeout property	
UserAuthenticationRenew property	
UserAuthenticationTimeout property	

The object has all the members of a Configuration object plus additional members that relate to the object's unique functions.

You must stop and restart the Server to use any new settings for this configuration.

SSLCert object

Inherits from parent object – Transmitter object

This object represents configuration settings for session protocol and ports, server authentication, and client authentication. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
CompanyName property	**GetCertificates method**
DefaultHTTPSPort property	**SetDefaultHTTPSPort method**
DefaultSSLPort property	**SetDefaultSSLPort method**
DisableMD5 property	
EnableClientCert property	
EnableFIPS property	
EnableHTTPS property	
EnableSSL property	
HTTPSPort property	
Parent property	
SSLPort property	
Transmitter property	
Unit property	

TenantConfiguration object

Inherits from parent object – TenantConfigurations collection

This object represents a single tenant record. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
Description property	N/A
Enabled property	
ID property	

Performing tasks for a particular tenant requires that you first establish a tenant context before performing the tasks. Establish the tenant context by setting either the ID or name property to the tenant of interest. While setting either property is sufficient for establishing a tenant context, it is acceptable to set both properties.

See “TenantName property” on page 214 for an example of changing tenant context before executing a task.

See “TenantConfigurations collection” on page 109 for an example of manipulating the TenantConfigurations collection to access an individual TenantConfiguration object.

Transmitter object

Inherits from parent object – TransmitterAccess object

This object represents your Server. From the Transmitter object, you can directly access other objects by using the Transmitter object's properties and methods, or you can indirectly access objects through other objects obtained by these properties and methods.

The Transmitter object uses a significant amount of computer resources. In addition, it may take several seconds to create this object. Automation clients are advised to use a single Transmitter object that is referenced for the life of the application. The only time you should create multiple Transmitter objects is if you need to simultaneously control multiple Servers.

The following table lists related properties, methods, and events. Click the name to view details.

Properties	*Methods*	*Events*
ChannelItems property	**CheckVersion method**	**OnTransmitterStateChanged event**
Configurations property	**Start method**	
Constant property	**Stop method**	
DynamicData property		
DynamicDataSet property		
FarmID property		
FullName property		
GetTrigger property		
Licenses property		
Name property		
Parent property		
Path property		
Profiles property		
ServerID property		
SSLCert property		
TenantID property		
TenantName property		
Transmitter property		
TransmitterState property		

Properties | Methods | Events

Version property

TransmitterAccess object

Inherits from parent object – N/A

This object represents the servers that are accessible to automation clients from the local workstation. To control a particular Server, use the GetTransmitterFromAddress method to obtain a Transmitter object that represents the requested Server. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
TransmitterDescriptions property	**GetTransmitterFromAddress method**
	GetTransmitterFromAddress2 method

TransmitterConfiguration object

Inherits from parent object – Configuration object

This object represents the settings and options for a Server's name, address and port. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
Address property	**ResetAddress method**
Port property	**ResetName method**
	ResetPort method

The object has all the members of a Configuration object plus additional members that relate to the object's unique functions.

You must stop and restart the Server to use any new settings for this configuration.

TransmitterDescription object

Inherits from parent object – TransmitterDescriptions collection

This object represents information about a Server that can be accessed from the local workstation. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
Address property	N/A
Description property	
Name property	

Trigger object

Inherits from parent object – Transmitter object

This object represents start times, repetition criteria, and other server schedule information. The Trigger object is created by automation clients. Use the GetTrigger property on a Transmitter object to get an uninitialized Trigger.

You must use the TriggerType property to set the trigger type before getting or setting any other trigger properties.

The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
BeginDate property	**SetDailyTrigger method**
DailyInterval property	**SetHourlyTrigger method**
DayOfTheMonth property	
EndDate property	
MinutesDuration property	
MinutesInterval property	
MonthlyInterval property	
StartHour property	
StartMinute property	
TriggerFlags property	
TriggerString property	
TriggerType property	
ValidDaysOfMonth property	
ValidDaysOfWeek property	
ValidMonths property	
WeeklyInterval property	
WeekOfTheMonth property	

4

Collections

A collection object is an object that contains a set of related objects. You can work with the objects in a collection as a single group rather than as separate entities.

There are two basic types of collections: *snapshot* and *live*. When you reference a snapshot collection, the reference reflects the contents of the collection at the time it was created. When you reference a live collection, the reference reflects the current contents of the collection.

About collections

Collections are objects that contain other objects. Collection objects provide a standard set of methods that let you enumerate the items in the collection.

You usually get a collection by calling a property with a similar name. For example, to get the ChannelItems collection representing all the channel items on the Server, you call the ChannelItems property on the Transmitter object.

The following table lists the available collection objects:

Collections	*Represent*
ActiveSessions collection	All active Client sessions.
BandwidthThrottlingConfigurations collection	All the configurations that represent a Server's bandwidth throttling.
CachedFiles collection	All files in the Server's compressed file cache.
CertificateConfigurations collection	All the configurations that represent a Server's certificates.
ChannelItems collection	All the Server's channel-related items (includes Channels, Folders, Channel Sets, and so on).
ChannelSetMembers collection	All the channels in a channel set.
Configurations collection	All configurations for a Server.
DifferenceFiles collection	All the files in the Server's difference file cache.
FailedSessions collection	All the failed sessions for a Server.
FileVersions collection	All the versions of a file difference.
Licenses collection	All licenses for a Server.
Profiles collection	All profiles for a Server.
TenantConfigurations collection	All tenants for a Server.
TransmitterDescriptions collection	Information about Servers that can be accessed from the local workstation.

ActiveSessions collection

Inherits from Parent object – Transmitter object

The ActiveSessions collection represents all the Server's active Client sessions.

The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
ActiveSessionCounts property	N/A
ActiveSessionsDetails property	
ActiveSessionsDetailsForConnection property	

BandwidthThrottlingConfigurations collection

Inherits from Parent object – Configuration object

The BandwidthThrottlingConfigurations collection represents all the Server's bandwidth throttling configurations. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
_NewEnum property	**CreateFrom method**
Count property	**Remove method**
DefaultConfiguration property	
EnableBandwidthThrottling property	
EnableCalibration property	
EnableEventLogging property	
Type property	

Remarks

The BandwidthThrottlingConfigurations object is a collection object that only contains BandwidthThrottlingConfiguration objects. To find a particular object, iterate through the collection by using For Each ... Next in a VBScript macro.

```
Dim bandwidthThrottlingConfigs
Dim bandwidthThrottlingConfig
Set bandwidthThrottlingConfigs = Transmitter.Configurations("BandwidthThrottling")
For Each bandwidthThrottlingConfig in bandwidthThrottlingConfigs
    Access bandwidthThrottlingConfig here.
    For example:
    MsgBox bandwidthThrottlingConfig.Description
Next
```

CachedFiles collection

Inherits from Parent object – CacheConfiguration object

Collection type: snapshot

The CachedFiles collection represents all the files in the Server's compressed file cache. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
N/A	**Add method (CachedFiles)** **Remove method**

Remarks

The CachedFiles object is a collection object that only contains CachedFile objects. To find a particular object, iterate through the collection by using For Each ... Next in a VBScript macro.

```
Dim compressedFile
For Each compressedFile in Transmitter.Configurations("Cache").CachedFiles
    ' Access compressedFile here.
    ' For example:
     MsgBox compressedFile.Name
Next
```

CertificateConfigurations collection

Inherits from Parent object – Configuration object

The CertificateConfigurations collection represents all the certificate configurations for a Server. The following table displays the properties and methods. Click the name to view details.

Properties	*Methods*
_NewEnum property	**AddCertificate method**
Count property	**ChangePassword method**
Parent property	**DeleteCertificate method**
Transmitter property	**Item method**
Type property	**ResetAll method**
	SetPassword method

Remarks

The CertificateConfigurations object is a collection object that only contains CertificateConfiguration objects. To find a particular object, iterate through the collection by using For Each ... Next in a VBScript macro.

```
Dim certificateConfigs
Dim certificateConfig
Set certificateConfigs = Transmitter.Configurations("Certificate")
For Each certificateConfig in certificateConfigs
    Access certificateConfig here.
    For example:
    MsgBox certificateConfig.CertificateFileName
Next
```

ChannelItems collection

Inherits from Parent object – Transmitter object, Folder object

Collection type: snapshot

The ChannelItems collection represents all channel-related items for a Server. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
_NewEnum property	**Add method (ChannelItems)**
Count property	**Folder method**
	GetItemByID method
	Item method
	Remove method
	ResetAll method

Remarks

The ChannelItems object is a collection object that only contains ChannelItem objects. To find a particular object, iterate through the collection by using For Each ... Next in a VBScript macro.

```
Dim chanItem
For Each chanItem in Transmitter.ChannelItems
  ' Access chanItem here.
  ' For example:
     MsgBox chanItem.Name
 Next
```

ChannelSetMembers collection

Inherits from Parent object – ChannelSet object

Collection Type: snapshot

The ChannelSetMembers collection represents all the channels in a channel set. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
N/A	**Add method (ChannelSetMembers)**

Remarks

The ChannelSetMembers object is a collection object that only contains Channel objects. To find a particular object, iterate through the collection by using For Each ... Next in a VBScript macro.

```
Dim channel, channelSet
For Each channel In channelSet
  ' Access channel here.
  ' For example:
     MsgBox channel.Name
 Next
```

Configurations collection

Inherits from Parent object – Transmitter object

Collection Type: snapshot

The Configurations collection represents all configurations for a Server. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
_NewEnum property	**Item method**
Count property	**ResetAll method**

Interating Through a Configurations Collection

Iterate through the collection by using For Each ... Next in a VBScript macro, as demonstrated in the following example.

```
Dim myTenant
For Each myTenant in Transmitter.Configurations("Tenant").TenantConfiguration
    ' Access myTenant here.
    ' For example:
     MsgBox myTenant.Enabled
Next
```

Manipulating a Single Item in a Configurations Collection

Manipulate a single item in a collection by using the Item method.

```
Public Class Form1
    Dim ta As New Afaria.TransmitterAccess
    Dim t
    Dim tenantConfig
    Dim tenantConfigs

   Private Sub Form1_Load(ByVal eventSender As System.Object, ByVal eventArgs As
System.EventArgs) Handles MyBase.Load

        ta = CreateObject("Afaria.TransmitterAccess")
        t = ta.GetTransmitterFromAddress
        tenantConfigs = t.Configurations("TenantConfigurations")
        tenantConfig = tenantConfigs.Item(1)
        tenantConfig.Name = "My new tenant name"
        tenantConfig.Description = "My new description"
```

```
        tenantConfig.Enabled = false

        End

    End Sub

End Class
```

DifferenceFiles collection

Inherits from Parent object – DifferenceConfiguration object

Collection Type: snapshot

The DifferenceFiles collection represents all the files in the Server's difference file cache. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
N/A	**Add method (DifferenceFiles)** **Remove method**

Remarks

The DifferenceFiles object is a collection object that only contains DifferenceFile objects. To find a particular object, iterate through the collection by using For Each ... Next in a VBScript macro.

```
Dim diffFile
For Each diffFile in Transmitter.Configurations("Difference").DifferenceFiles
    ' Access diffFile here.
    ' For example:
     MsgBox diffFile.Name
Next
```

FailedSessions collection

Inherits from Parent object – CleanupConfiguration object

Collection Type: snapshot

The FailedSessions collection represents all the failed sessions for a Server. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
N/A	**Remove method**

Remarks

The FailedSessions object is a collection object that only contains FailedSession objects. To find a particular object, iterate through the collection by using For Each ... Next in a VBScript macro.

```
Dim failedSession
For Each failedSession in Transmitter.Configurations("Cleanup").FailedSessions
    ' Access failedSession here.
    ' For example:
     MsgBox failedSession.ChannelName
Next
```

FileVersions collection

Inherits from Parent object – DifferenceFile object

Collection Type: snapshot

The FileVersions collection represents all the versions of a file in the Server's difference file cache. The object does not introduce new members to the object model; it relies solely on inherited members.

Remarks

The FileVersions object is a collection object that only contains FileVersion objects. To find a particular object, iterate through the collection by using For Each ... Next in a VBScript macro.

```
Dim diffFile, fileVersion
For Each diffFile in Transmitter.Configurations("Difference").DifferenceFiles
  For Each fileVersion in diffFile.FileVersions
      ' Access fileVersion here.
      ' For example:
       MsgBox fileVersion.Name
  Next
Next
```

Licenses collection

Inherits from Parent object – Transmitter object

Collection Type: snapshot

The Licenses collection represents all licenses for a Server. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
_NewEnum property	**IsClientTypeLicensed method**
Count property	**IsProductLicensedForAnyClientType method**
ExpirationDate property	**Item method**
LicenseKey property	
Parent property	
SessionLimit property	
Transmitter property	

Remarks

The Licenses object is a collection object that only contains LicensedComponent objects. To find a particular object, iterate through the collection by using For Each ... Next in a VBScript macro.

```
Dim myLicense
For Each myLicense in Transmitter.Licenses
    ' Access myLicense here.
    ' For example:
     MsgBox myLicense.Type
Next
```

Profiles collection

Inherits from Parent object – Configuration object

Collection Type: snapshot

The Profiles collection represents all group profiles for a Server. The object does not introduce new members to the object model; it relies solely on inherited members.

Remarks

The Profiles object is a collection object that only contains Profile objects. To find a particular object, iterate through the collection by using For Each ... Next in a VBScript macro.

```
Dim myProfile
For Each myProfile in Transmitter.Profiles
    ' Access myProfile here.
    ' For example:
     MsgBox myProfile.Assignments
Next
```

TenantConfigurations collection

Inherits from Parent object – Configuration object

Collection Type: snapshot

The TenantConfigurations object is a collection object that only contains TenantConfiguration objects. It represents all tenants for a Server. The object does not introduce new members to the object model; it relies solely on inherited members.

.Item 1 in the collection is reserved for the system-defined tenant.

TransmitterDescriptions collection

Inherits from Parent object – TransmitterAccess object

Collection Type: snapshot

The TransmitterDescriptions collection represents information about Servers that can be accessed from the local workstation. The following table lists the members that the object introduces to the object model. Click the name to view details.

Properties	*Methods*
_NewEnum property	**Item method**
Count property	

Remarks

The TransmitterDescriptions object is a collection object that only contains TransmitterDescription objects. To find a particular object, iterate through the collection by using For Each ... Next in a VBScript macro.

```
Dim myConfiguration
For Each myConfiguration in Transmitter.Configurations
    ' Access myConfiguration here.
    ' For example:
     MsgBox myConfiguration.Name
Next
```

5

Properties

In general, you use properties to get to content, which can include the data contained in an object or the attribute settings for the object.

About properties

Properties are characteristics of objects. For example, Transmitter objects have Name and Version properties. These properties correspond to the name and version of a Server.

For more background about properties, **see "Properties and Methods" on page 20.**

_NewEnum property

References objects in a collection.

Table

Syntax	*object.*_**NewEnum**
Parameters	N/A
Remarks	With Visual C++, you can browse a collection to find a particular item by using the _NewEnum property or the Item method. In Visual Basic, you do not need to use the _NewEnum property, because it is automatically used in the implementation of For Each ... Next.
Example	This Visual C++ example shows a command handler using the ActiveX Template Library. This example uses the _NewEnum property to iterate through all channel items and display their names in the output window. ***Tip*** You can iterate through these channel items more efficiently by using the Item method of the ChannelItems object.

```
CComPtr<IDispatch> pDisp;

CComQIPtr<IChannelItems, &IID_IChannelItems> pChannels;
m_pTransmitter->get_Channels(&pDisp);
pChannels = pDisp;
pDisp = NULL;

CComPtr<IUnknown> pUnk;
CComQIPtr<IEnumVARIANT, &IID_IEnumVARIANT> pNewEnum;
if (SUCCEEDED(pChannels->get__NewEnum(&pUnk)) && pUnk != NULL)
{
  pNewEnum = pUnk;
  VARIANT varChannel;
  CComQIPtr<IChannel, &IID_IChannelItem> pChannel;
  while (pNewEnum->Next(1, &varChannel, NULL) == S_OK)
  {
    ASSERT (varChannel.vt == VT_DISPATCH);

    pChannel = varChannel.pdispVal;
    VariantClear(&varChannel);
    CComBSTR bstrName;
    pName->get_Name(&bstrName);
    OutputDebugStringW( bstrName);
  }
}
```

ActiveSessionCounts property

Returns the count of active Client sessions.

Table

Syntax	*object.***ActiveSessionCounts**
Parameters	Count Returned integer of active sessions.
Remarks	The ActiveSessionCounts property has the **Long** type. It returns the number of active sessions.
Example	This example gets the count of active Client sessions for a Server: `Msgbox.Transmitter.Connection.ActiveSessions.ActiveSessionCounts`

ActiveSessionsDetails property

Gets the details of all current active sessions.

Table

Syntax	*object.***ActiveSessionsDetails** *Details*
Parameters	N/A

Table

Remarks

The ActiveSessionsDetails property has the **Recordset** type. The recordset gets the details of all the current active sessions.The recordset contains the following fields:

- **User**. User running the session.
- **Machine**. Machine running the session.
- **InternetAddress**. Address of the machine running the session.
- **ChannelName**. Channel running at the Client.
- **Status**. Status of the channel running at the Client.
- **Progress**. Progress of the channel running at the Client.
- **TotalEvents**. Total events in the channel.
- **CurrentEvent**. Current channel event being processed.
- **SessStartTime**. Time session started.
- **EventStartTime**. Time current event started.
- **ServerPath**. Path of the file on Server involved in the event (if applicable).
- **ClientPath**. Path of the file on Client involved in the event (if applicable).
- **BWTThroughput**. Bandwidth throttling throughput set for connection.
- **BWTState**. Bandwidth throttling state set for connection.
- **ConnectionID**. Unique ID used to identify the connection.
- **LineStatus**. Status of the connection.
- **Command**. Event ID of the current command being executed.
- **FileName**. Name of the file involved in the event (if applicable).
- **FileDate**. Date of the file involved in the event (if applicable).
- **FileSize**. Size of the file involved in the event (if applicable).
- **FileBytesTransferred**. Number of bytes transferred during a session (if applicable).

Example

This example gets the details of all the current active sessions:

```
Dim m_RecSet
Dim m_Field
Set m_RecSet = Createobject(ADODB.Recordset”)
Set m_RecSet =
Transmitter.Connection.ActiveSessions.ActiveSessionsDetails

Do while not m_RecSet.EOF
   For each m_Field in m_RecSet.Fields
       ‘Steps to process each field
   next
   m_RecSet.MoveNext
Loop
```

ActiveSessionsDetailsForConnection property

Gets the details of a single current active session.

Table

Syntax	*object.***ActiveSessionsDetails** *ConnectionID*
Parameters	ConnectionID GUID of conenction.
Remarks	The ActiveSessionsDetailsForConnection property has the Recordset type. The recordset gets the details of a single current active session. The recordset contains the following fields: • **User**. User running the session. • **Machine**. Machine running the session. • **InternetAddress**. Address of the machine running the session. • **ChannelName**. Channel running at the Client. • **Status**. Status of the channel running at the Client. • **Progress**. Progress of the channel running at the Client. • **TotalEvents**. Total events in the channel. • **CurrentEvent**. Current channel event being processed. • **SessStartTime**. Time session started. • **EventStartTime**. Time current event started. • **ServerPath**. Path of the file on Server involved in the event (if applicable). • **ClientPath**. Path of the file on Client involved in the event (if applicable). • **BWTThroughput**. Bandwidth throttling throughput set for connection. • **BWTState**. Bandwidth throttling state set for connection. • **ConnectionID**. Unique ID used to identify the connection. • **LineStatus**. Status of the connection. • **Command**. Event ID of the current command being executed. • **FileName**. Name of the file involved in the event (if applicable). • **FileDate**. Date of the file involved in the event (if applicable). • **FileSize**. Size of the file involved in the event (if applicable). • **FileBytesTransferred**. Number of bytes transferred during a session (if applicable).

Table

Example This example gets the details of a single current active session:

```
Dim m_RecSet
Dim m_Field
Set m_RecSet = CreateObject("ADODB.Recordset")
Set m_RecSet =
Transmitter.Connection.ActiveSessionsDetailsForConnection(ConnectionI
D)

Do while not m_RecSet.EOF
   For each m_Field in m_RecSet.Fields
       'Steps to proceed each field
   next
   m_RecSet.MoveNext
Loop
```

Address property

Gets and sets the address of an object.

Table

Syntax	*object*.**Address** [= *value*]
Parameters	value A String that represents the new address.
Remarks	The Address property has the **String** type. The following table summarizes the results of using the Address property with the objects listed:

Table

Object	Results
ContactConfiguration	Gets and sets the e-mail address that Channel Viewer users can use for assistance.
TransmitterConfiguration	Gets and sets the network IP address used by clients to connect to the Server. The address can be a machine name, such as "companyname.com," or a numeric IP address, such as "128.56.22.8."
TransmitterDescription	Gets the name of the Server.

Table

Example	This example sets the value from the configuration object: `Transmitter.Configurations("Contact").Address = "helpdesk@company.com"`

AlertLogSettings property

Gets and sets the alert logging policies in a logging configuration.

Table

Syntax	*object*.**AlertLogSettings** [= *value*]
Parameters	value An Integer value representing the AlertLogSettings.
Remarks	The AlertLogSettings property has the **Integer** type.

Example	This example gets the value from the configuration object:
	
	`Dim alertLogSettings`
	`alertLogSettings =`
	`  Transmitter.Configurations ("Logging").AlertLogSettings`

Alg property

Gets the certificate algorithm.

Table

Syntax	*object*.**Alg**
Parameters	N/A
Remarks	The Alg property has the **String** data type.

Table

Example	This example gets the value from the configuration object:

```
Dim CertificateConfigs, Alg
Set CertificateConfigs =
Transmitter.Configurations("CertificateConfigurations")
Alg = CertificateConfigs.Item(1).Alg
```

AllLogSettings property

Gets and sets the enable all logging policies in a logging configuration.

Table

Syntax	*object*.**AllLogSettings** [= *value*]
Parameters	value An Integer value representing the AllLogSettings.
Remarks	The AllLogSettings property has the **Integer** type.
Example	This example gets the value from the configuration object:

```
Dim AllLogSettings
allLogSettings =
  Transmitter.Configurations ("Logging").AllLogSettings
```

AllowedChannels property

Gets the explicitly allowed channels associated with the profile. An explicitly allowed channel is one that is manually added to a profile's allowed channels list, which is distinct from a implicitly allowed channel. An implicitly allowed channel is one that is present on the allowed channels list by virtue of its status as the profile's default channel or as a member of a channel set that is on the allowed channels list.

Table

Syntax	*object*.**AllowedChannels**
Parameters	N/A

Table

Remarks	Returns a BSTR containing an XML document describing the channels. <?xml version='1.0' encoding='utf-16'?> <AllowedChannels> <Channel Name="..." OriginalTransmitterID="..." OriginalChannelID="..."> </AllowedChannels> The AllowedChannels node contains zero or more Channel nodes. The Channel node has three attributes: Name, OriginalTransmitterID, and OriginalChannelID. Name: Fully qualified name of channel. OriginalTransmitterID: ID of server on which the channel was originally created. OriginalChannelID: ID of channel as it was originally created.
Example	This example gets the current profile's channel list. `set channels = prf.AllowedChannels` See "Profile object" on page 82 for an extended example.

Authenticate property

Gets and sets whether a channel requires client user authentication.

Table

Syntax	*object*.**Authenticate** [= *value*]
Parameters	value A Boolean that specifies whether an object requires client user authentication.
Remarks	The Authenticate property has the **Boolean** type.
Example	

Assignments property

Gets the user group assignments associated with a profile.

Table

Syntax	*object*.**Assignments**
Parameters	N/A
Remarks	Returns a BSTR containing an XML document describing the user assignments <?xml version='1.0' encoding='utf-16'?> <Assignments> <Assignment Domain="..." Group="..." Type="..."> </Assignments> The Assignments node contains zero or more Assignment nodes. The Assignment node has three attributes: Domain, Group, and Type. Domain: Must be empty ("") for local and client groups and the system "All Clients" group. For LDAP, caller must pass full domain, such as win2003mixed.testdomain.com. For domain groups, it must contain the domain, such as testdomain. Group: For local, domain and client groups, this must be the name of the group, such as MyClientGroup, Administrators, and so on. For the system group, this must be empty (""). For LDAP, it must look something like this: CN=mygroup,DC=testdomain,DC=com. Type: Must be "local", "domain", "client", system (without quotation marks), "ldapou" or "ldapobj".
Example	This example gets the current profile's assignments list. `set channels = prf.Assignments` See "Profile object" on page 82 for an extended example.

Authenticate property

Gets and sets whether a channel requires client user authentication.

Table

Syntax	*object*.**Authenticate** [= *value*]

Table

Parameters	value A Boolean that specifies whether an object requires client user authentication.
Remarks	The Authenticate property has the **Boolean** type.
Example	

AuthenticationServer property

Gets and sets the name of the server used to authenticate users.

Table

Syntax	*object*.**AuthenticationServer** [= *value*]
Parameters	value A String that specifies the name of the server used to authenticate users.
Remarks	The AuthenticationServer property has the **String** type. By default, the local NT domain is used. If LDAP is enabled, this is a read-only property that represents the name of the LDAP server.
Example	This example gets the value from the configuration object: `Transmitter.Configurations("Security").AuthenticationServer`

AutoDelete property

Gets and sets whether an object automatically deletes itself.

The order of operations is important. It is an error to set the auto delete time without first enabling it. Similarly, it is an error to get the auto delete time if it is disabled. See the example below for how to correctly set and get the auto delete time.

Table

Syntax	*object*.**AutoDelete** [= *value*]
Parameters	value A Date that specifies when an object automatically deletes itself.

Table

Remarks	The AutoDeleteTime property has the **Date** type.

Example	Example 1
	This example sets the auto delete time for a channel:
	
	`channel.AutoDelete = True`
	`channel.AutoDeleteTime = CDate("January 1, 2009")`
	
	Example 2
	This example gets the auto delete time for a channel:
	
	`If channel.AutoDelete Then`
	`  Msgbox "Auto delete time is " & channel.AutoDeleteTime`
	`Else`
	`  Msgbox "Auto delete time not enabled"`
	`End If`

AutoDeleteTime property

Gets and sets when an object automatically deletes itself.

The order of operations is important. It is an error to set the auto delete time without first enabling it. Similarly, it is an error to get the auto delete time if it is disabled. See the example below for how to correctly set and get the auto delete time.

Table

Syntax	*object*.**AutoDeleteTime** [= *value*]
Parameters	value A Boolean that specifies whether an object automatically deletes itself.
Remarks	The AutoDelete property has the **Boolean** type.

Table

Example	Example 1
	This example sets the auto delete time for a channel:
	
	`channel.AutoDelete = True`
	`channel.AutoDeleteTime = CDate("January 1, 2009")`
	
	Example 2
	This example gets the auto delete time for a channel:
	
	`If channel.AutoDelete Then`
	`  Msgbox "Auto delete time is " & channel.AutoDeleteTime`
	`Else`
	`  Msgbox "Auto delete time not enabled"`
	`End If`

AutoPublish property

Gets and sets whether a channel automatically publishes itself.

The order of operations is important. It is an error to set the auto publish time without first enabling it. Similarly, it is an error to get the auto publish time if it is disabled. See the example below for how to correctly set and get the auto publish time.

Table

Syntax	*object*.**AutoPublish** [= *value*]
Parameters	value A Boolean that specifies whether a channel automatically publishes itself.
Remarks	The AutoPublish property has the **Boolean** type.

Table

Example	Example 1
	This example sets the auto publish time for a channel:
	
	`channel.AutoPublish = True`
	`channel.AutoOPublishTime = CDate("January 1, 2009")`
	
	Example 2
	This example gets the auto publish time for a channel:
	
	`If channel.AutoPublish Then`
	`  Msgbox "Auto publish time is " & channel.AutoPublishTime`
	`Else`
	`  Msgbox "Auto publish time not enabled"`
	`End If`

AutoPublishTime property

Gets and sets when a channel automatically publishes itself.

The order of operations is important. It is an error to set the auto publish time without first enabling it. Similarly, it is an error to get the auto publish time if it is disabled. See the example below for how to correctly set and get the auto publish time.

Table

Syntax	*object*.**AutoPublishTime** [= *value*]
Parameters	value A Date that specifies when a channel automatically publishes itself.
Remarks	The AutoPublish property has the **Date** type.

Table

Example	Example 1
	This example sets the auto publish time for a channel:
	
	`channel.AutoPublish = True`
	`channel.AutoOPublishTime = CDate("January 1, 2009")`
	
	Example 2
	This example gets the auto publish time for a channel:
	
	`If channel.AutoPublish Then`
	`  Msgbox "Auto publish time is " & channel.AutoPublishTime`
	`Else`
	`  Msgbox "Auto publish time not enabled"`
	`End If`

AutoRefreshContent property

Gets and sets whether a channel automatically updates its content.

Table

Syntax	*object*.**AutoRefreshContent** [= *value*]
Parameters	value A Boolean that specifies whether a channel automatically updates its content.
Remarks	The AutoRefreshContent property has the **Boolean** type.
Example	This example enables automatic content updates for a channel: `channel.AutoRefreshContent = True`

AutoSubscribe property

Gets and sets whether a channel is automatically sent to Clients.

Table

Syntax	*object*.**AutoSubscribe** [= *value*]

Table

Parameters	value A Boolean that specifies whether a channel is automatically sent to Clients.
Remarks	The AutoSubscribe property has the **Boolean** type. This option "forces" or "pushes" channel content to the Client when you want the Client to have the most up-to-date information. This option removes the Client's ability to unsubscribe to the channel. The channel is automatically sent to Clients the next time they connect to the Server.
Example	This example automatically sends the channel to Clients: `channel.AutoSubscribe = True`

AutoUnpublish property

Gets and sets whether a channel automatically unpublishes itself.

The order of operations is important. It is an error to set the auto unpublish time without first enabling it. Similarly, it is an error to get the auto unpublish time if it is disabled. See the example below for how to correctly set and get the auto unpublish time.

Table

Syntax	*object*.**AutoUnpublish** [= *value*]
Parameters	value A Boolean that specifies whether a channel automatically unpublishes itself.
Remarks	The AutoUnpublish property has the **Boolean** type.

Example	Example 1
	This example sets the auto unpublish time for a channel:
	`channel.AutoUnpublish = True`
	`channel.AutoUnpublishTime = CDate("January 1, 2009")`
	
	Example 2
	This example gets the auto unpublish time for a channel:
	`If channel.AutoUnpublish Then`
	`  Msgbox "Auto unpublish time is " & channel.AutoUnpublishTime`
	`Else`
	`  Msgbox "Auto unpublish time not enabled"`
	`End If`

AutoUnpublishTime property

Gets and sets when a channel automatically unpublishes itself.

The order of operations is important. It is an error to set the auto unpublish time without first enabling it. Similarly, it is an error to get the auto unpublish time if it is disabled. See the example below for how to correctly set and get the auto publish time.

Table

Syntax	*object*.**AutoUnpublishTime** [= *value*]
Parameters	value A Date that specifies when a channel automatically unpublishes itself.
Remarks	The AutoUnpublishTime property has the **Date** type.

Example	Example 1
	This example sets the auto unpublish time for a channel:
	`channel.AutoUnpublish = True`
	`channel.AutoUnpublishTime = CDate("January 1, 2009")`
	
	Example 2
	This example gets the auto unpublish time for a channel:
	`If channel.AutoUnpublish Then`
	`  Msgbox "Auto unpublish time is " & channel.AutoUnpublishTime`
	`Else`
	`  Msgbox "Auto unpublish time not enabled"`
	`End If`

BeginDate property

Gets and sets the year, month, and day that a trigger activates.

You must use the TriggerType property to set the trigger type before getting or setting any other trigger properties.

Table

Syntax	*object*.**BeginDate** [= *value*]
Parameters	value A Date that specifies the year, month, and day that a trigger activates.

Table

Remarks	The BeginDate property has the **Date** type. Use the StartHour and StartMinute properties to set the starting hour and minute, respectively.

Example	This example creates a trigger that fires once every day, starting on January 1, 2009:
	
	`Dim ta As afaria.TransmitterAccess`
	`Dim trans As afaria.Transmitter`
	`Dim trigger As afaria.trigger`
	`Set ta = CreateObject("Afaria.TransmitterAccess")`
	`Set trans = afaria.GetTransmitterFromAddress`
	`Set trigger = trans.GetTrigger`
	`trigger.TriggerType =`
	`Transmitter.Constant("cmTASK_TIME_TRIGGER_DAILY")`
	`trigger.BeginDate = CDate("January 1, 2009")`

CachedFiles property

Gets the CachedFiles collection.

Table

Syntax	*object*.**CachedFiles**
Parameters	N/A
Remarks	The CachedFiles property has the **CachedFiles** type. The CachedFiles object represents all the files in the Server's compressed file cache. Each item in this collection is represented by a CachedFile object.
Example	This example gets the value from the configuration object: `Transmitter.Configurations("Cache").CachedFiles`

CertificateDirectory property

Gets the certificate directory for the Server.

Table

Syntax	*object*.**CertificateDirectory**
Parameters	N/A
Remarks	The CertificateDirectory property has the **String** type.
Example	This example gets the value from the configuration object:

```
Msgbox
Transmitter.Configurations("CertificateGeneration").CertificateDirec
tory
```

CertificateFileName property

Gets the file name value in a certificate configuration.

Table

Syntax	*object*.**CertificateFileName** [= *value*]
Parameters	N/A
Remarks	The CertificateFileName property has the String data type.
Example	This example gets the value from the configuration object:

```
Dim certificateConfigs, certificateFileName
Set certificateConfigs =
Transmitter.Configurations("Certificate")
certificateFileName =
certificateConfigs.Item(1).CertificateFileName
```

CertificationDatabase property

Gets and sets the SSL (Secure Sockets Layer) certification database.

Table

Syntax	*object*.**CertificationDatabase** [= *value*]
Parameters	value A String that specifies the name of the SSL (Secure Sockets Layer) certification database.
Remarks	The CertificationDatabase property has the **String** type.
Example	This example gets the value from the configuration object: `Transmitter.Configurations("Security").CertificationDatabase`

ChannelItems property

Gets the ChannelItems collection.

Table

Syntax	*object*.**ChannelItems** [*StartingFolder*] [, *Flags = cmFilterByAll*]
Parameters	StartingFolder (Optional) A String that represents the starting folder for the collection. The default for a Transmitter object is the top-most folder (a.k.a. the root folder). The default for a Folder object is the folder. It is an error to request a starting folder that does not exist or does not represent an actual folder item. To specify the root folder, do not specify a value for the parameter, or use the string "\". To specify the name of a folder other than the root folder, specify the fully qualified folder name, as in "\Folder1" or "\Folder1\SubFolder11". Do not use a backslash character at the end of the folder name. Flags (Optional) A single value from the cmChannelFilterEnum enumeration specifying which items to include in the collection. If this parameter is missing or has a value of cmFilterByAll, the collection will contain all channel items, starting at the specified folder. The enumeration does not include items in any subfolders.

Table

Remarks	The ChannelItems property has the **ChannelItems** type. For a Transmitter object, the ChannelItems object represents the collection of channel-related items associated with a Server, starting from the root folder. For a Folder object, the ChannelItems object represents the collection of channel-related items associated with a particular folder, starting from the folder represented by the Folder object. Each item in this collection is represented by a ChannelItem object. By default, the ChannelItems object enumerates all channel item types starting from the specified folder, without recursion. Use the optional parameters to restrict the items in the enumeration.

Table

Example ChannelItems property

Consider a hypothetical Server named MOTHRA with the following channel hierarchy.
This Server contains two folders located in the root folder named Folder1 and Folder2.
Folder1 contains a subfolder named Folder11.
The Server also contains four Session objects located in the root folder named Session1, Session2, Session3, and Session4.

This example shows how to enumerate channel items using different starting folders.

```
Dim ta As Afaria.TransmitterAccess
Dim trans As Afaria.Transmitter
Set ta = CreateObject("Afaria.TransmitterAccess")
Set trans = ta.GetTransmitterFromAddress
Dim chanItem As Afaria.ChannelItem
Dim flags As Long
```

Example 1
Use the default parameters. This will enumerate all channel items starting at the root folder, without recursion.

```
For Each chanItem In Transmitter.ChannelItems
  MsgBox chanItem.Type
Next
```

Example 2
Same as Example 1, but explicitly specifying the parameters using the default values.

```
flags = cmFilterByAll
For Each chanItem In Transmitter.ChannelItems("\", flags)
  MsgBox chanItem.Type
Next
```

Example 3
Enumerate all system channel items starting at the root folder, without recursion.

```
flags = cmFilterBySystem
For Each chanItem In Transmitter.ChannelItems( , flags)
  MsgBox chanItem.Type
Next
```

Example 4
Enumerate all folders, starting from Folder1, without recursion.

```
flags = cmFilterByFolders
For Each chanItem In Transmitter.ChannelItems( "\Folder1",
flags)
  MsgBox chanItem.Type
Next
```

ChannelName property

Gets the name of the Channel associated with the object.

Table

Syntax	*object*.**ChannelName**
Parameter	N/A
Remarks	The ChannelName property has the **String** type. For FailedSession, gets the name of the channel that failed during a communication session between a Client and the Server.

Example	This example gets the value from the configuration object:
	`Dim name`
	`name =`
	`Transmitter.Configurations("Cleanup").FailedSessions(1).ChannelName`

ChannelSetMembers property

Gets the name of the Channel associated with the object.

Table

Syntax	*object*.**ChannelName**
Parameter	N/A
Remarks	The ChannelName property has the **String** type. For FailedSession, gets the name of the channel that failed during a communication session between a Client and the Server.

Example	This example gets the value from the configuration object:
	`Dim name`
	`name =`
	`Transmitter.Configurations("Cleanup").FailedSessions(1).ChannelName`

ChannelUpdateSchedule property

Gets and sets the schedule for automatically updating channel contents.

Table

Syntax	*object*.**ChannelUpdateSchedule** [= *Trigger*]
Parameters	Trigger A Trigger object that represents the new schedule.
Remarks	The ChannelUpdateSchedule property has the **Trigger** type. The channel update schedule specifies how often the Server should refresh channel content. The Server refreshes channel content to ensure that the most current version of data is available to Clients. The default value is once every day.

Example	This example sets the value for the configuration object to once every two days:
	
	`Dim trigger`
	`Set trigger = Transmitter.GetTrigger`
	`trigger.TriggerType =`
	`Transmitter.Constant("cmTASK_TIME_TRIGGER_DAILY")`
	`trigger.DailyInterval = 2`
	`Transmitter.Configurations("Cleanup").ChannelUpdateSchedule = trigger`

ClassID property

Gets the COM (Component Object Model) class identifier associated with an object.

Table

Syntax	*object*.**ClassID**
Parameters	N/A
Remarks	The ClassID property has the **String** type. The ClassID property represents an identifier that uniquely distinguishes an object's class from other classes. The identifier is returned as a string consisting of eight hexadecimal digits followed by a hyphen, then three groups of four hexadecimal digits each followed by a hyphen, then twelve hexadecimal digits (e.g., "{6B29FC40-CA47-1067-B31D-00DD010662DA}"). The string includes enclosing braces, which are an OLE convention.
Example	This example gets the class ID associated with the first channel item: `Transmitter.ChannelItems(1).ClassID`

ClientApprovalDir property

Gets and sets the path for storing the antivirus and firewall component client files that await administor approval.

Table

Syntax	object.**ClientApprovalDir** [=Value]
Parameters	Value A string value that specifies a directory on the Afaria server.
Remarks	If the directory name begins with .\ (example: .\Policy\AVFirewall\ClientHold), then the path is relative to the Afaria server's data directory.

Example	This example sets the client approval directory.
	
	`avfw.ClientApprovalDir = "C:\MyAfariaData\AVFWClientHold"`
	
	This example gets the client approval directory.
	
	`If avfw.ClientApprovalDir.StartsWith(".\") Then`
	`  MsgBox("Client approval dir is Afaria\Data\" &`
	`avfw.ClientApprovalDir.Substring(2))`
	`Else`
	`  MsgBox("Client approval dir is " & avfw.ClientApprovalDir)`
	`End If`

ClientHoldForApproval property

Gets and sets the value for whether to hold new antivirus and firewall component client files to await administrator approval or to immediately release new files to update current antivirus clients.

Table

Syntax	object.**ClientHoldForApproval** [=Value]
Parameters	Value An boolean value that specifies the server action for new client files. True to hold AV/Firewall client binaries for administrator approval. False to send AV/Firewall client binaries to clients on their next connection.

Table

Example	This example enables holding AV/Firewall client binaries for approval.
	
	`avfw.ClientHoldForApproval = True`
	
	This example gets the approval setting.
	
	`If avfw.ClientHoldForApproval Then`
	`  MsgBox("Clients are being held for approval")`
	`Else`
	`  MsgBox("Clients are being delivered without approval")`
	`End If`

ClientName property

Gets the name of the client associated with the object.

Table

Syntax	*object*.**ClientName**
Parameter	N/A
Remarks	The ClientName property has the **String** type. For FailedSession, gets the name of the Client associated with a failed communication session between a Client and the Server.

Example	This example gets the value from the configuration object:
	
	`Dim name`
	`name =`
	`Transmitter.Configurations("Cleanup").FailedSessions(1).ClientName`

CompanyName property

Gets and sets the value of the certificate settings configuration Company Name value for client authentication. The company name value corresponds to a client SSL certificate's Organization value.

Table

Syntax	*object*.**CompanyName**
Parameter	N/A
Remarks	**String** type. The Company Name field for client authentication.
Example	

Compressible property

Gets whether a cached file is compressible.

Table

Syntax	*object*.**Compressible**
Parameters	N/A
Remarks	The Compressible property has the **Boolean** type. If a file is not compressible, the Server sends the original source file to the Client.

Example	This example gets the value from the configuration object for the first item:
	
	`Dim cachedFiles`
	`Set cachedfiles = Transmitter.Configurations("Cache").CachedFiles`
	`If cachedFiles.Item(1).Compressible Then`
	`  Msgbox "Cached file is compressible"`
	`End`

ConfigurationSet property

Gets and sets the Bandwidth Throttling Configuration data structure.

Table

Syntax	*object*.**ConfigurationSet** [= *value*]
Parameters	value A structure that represents the set of values for the configuration.
Remarks	The ConfigurationSet property has the BANDWIDTH_THROTTLING_CONFIGURATION_SET structure data type.

Example	This example gets the value from the configuration object for the first item:
	
	`Dim configs`
	`Dim configurationSet`
	`Set configs = Transmitter.Configurations("BandwidthThrottling")`
	`Set configurationSet = configs.Item(1).ConfigurationSet`

Configurations property

Gets the Configurations collection.

Table

Syntax	*object*.**Configurations**
Parameters	N/A
Remarks	The Configurations property has the **Configurations** type. The Configurations object represents the collection of configurations associated with a Server. Each item in this collection is represented by a Configuration object.

Example	This example gets the value from the configuration object for each item in the collection:
	
	`For i = 1 to Transmitter.Configurations.Count`
	`  MsgBox Transmitter.Configurations.Item(i).Type`
	`Next`

ConfigXML property

Gets and sets the XML contents of a Configuration Manager channel, where "Configuration" refers to the ContentType property type. Get returns an XML string. Set defines a path on the Afaria server to the XML configuration file.

Table

Syntax	*object*.**ConfigXML** [= *XmlFile*]
Parameters	XmlFile A String that represents an XML snippet containing the implementation details for a Configuration Manager channel.
Example	This example sets the server path to the configuration file: `Channel.ConfigXML ="c:\configxm.xml"`

Constant property

Gets the value associated with the named constant.

Table

Syntax	*object*.**Constant** *Name*
Parameters	Name A String that is the name of a constant or enumeration in the Server object model. For a list of all valid constants, see Enumerations.
Remarks	The Constant property has the **Variant** type. This property is typically used by scripting clients since they do not have direct access to enumerated types.

Example	This example gets the Long value corresponding to the cmCacheNormal enumerated value:
	
	`Dim value`
	`value = transmitter.Constant("cmCacheNormal")`

ContentHomeDirectory property

Gets the path to the content information associated with the channel.

Table

Syntax	*object*.**ContentHomeDirectory**
Parameters	N/A
Remarks	The ContentHomeDirectory property has the **String** type.
Example	This example gets the path to the content for all channel items:

```
For Each channel In transmitter.ChannelItems
  If ( "Channel" = channel.Type ) Then
    MsgBox channel.ContentHomeDirectory
  End If
Next
```

ContentID property

Gets the unique identifier of a channel's content.

Table

Syntax	*object*.**ContentID**
Parameters	N/A
Remarks	The ContentID property has the **Long** type. The ContentID property represents an identifier that uniquely distinguishes an object on a Server. This identifier is assigned by the system when the object is created and is never reused.
Example	This example gets the content ID associated with the first channel item:

```
Dim channel
Set channel = Transmitter.ChannelItems(1)
If ( "Channel" = channel.Type ) Then
  MsgBox channel.ContentID
End If
```

ContentSize property

Gets the size of a channel's content.

Table

Syntax	*object*.**ContentSize**
Parameters	N/A
Remarks	The ContentSize property has the **Double** type. The ContentSize property represents the estimated size, in bytes, of the content associated with the channel. It is mainly used for displaying size and time estimates to the administrator or Client user.

Example	This example gets the content size associated with the first channel item:
	
	`Dim channel`
	`Set channel = Transmitter.ChannelItems(1)`
	`If ( "Channel" = channel.Type ) Then`
	`  MsgBox channel.ContentSize`
	`End If`

Count property

Gets the number of items in a collection.

Table

Syntax	*object*.**Count**
Parameters	N/A
Remarks	The Count property has the **Long** type. The number of items in a collection is not always accurate. This depends on the collection type. If you need to iterate all items in a collection, use the _NewEnum property.

Example	This example displays the name of each item in the ChannelItems collection.
	
	`For i = 1 to ChannelItems.Count`
	`    MsgBox ChannelItems.Item(i).Name`
	`Next`

CurveType property

Gets the curve type value in a certificate configuration.

Table

Syntax	*object*.**CurveType**
Parameters	object An object expression that evaluates to an object that implements this property.
Remarks	The CurveType property has the *String* data type.

Example	This example gets the value from the configuration object for the first item in the collection:
	
	`Dim certificateConfigs, curveType`
	`Set certificateConfigs =`
	`Transmitter.Configurations("Certificate")`
	`curveType = certificateConfigs.Item(1).CurveType`

DailyInterval property

Gets and sets the interval between triggers, in days.

You must use the TriggerType property to set the trigger type before getting or setting any other trigger properties.

Table

Syntax	*object*.**DailyInterval** [= *value*]
Parameters	value A Short that specifies the interval between triggers, in days. 1 = every day, 2 = every other day, etc.
Remarks	The DailyInterval property has the **Short** type.

Table

Example	This example creates a trigger that fires every other day, starting on January 1, 2009:
	
	`Dim ta As afaria.TransmitterAccess`
	`Dim trans As afaria.Transmitter`
	`Dim trigger As afaria.trigger`
	`Set ta = CreateObject("Afaria.TransmitterAccess")`
	`Set trans = afaria.GetTransmitterFromAddress`
	`Set trigger = trans.GetTrigger`
	`trigger.TriggerType =`
	`Transmitter.Constant("cmTASK_TIME_TRIGGER_DAILY")`
	`trigger.BeginDate = CDate("January 1, 2009")`
	`trigger.DailyInterval = 2`

Date property

Gets the time and date associated with the object.

Table

Syntax	*object*.**Date**
Parameter	N/A
Remarks	The Date property has the **Date** type. The Date type is implemented using an 8-byte floating-point number. Days are represented by whole number increments starting with 30 December 1899, midnight as time zero. Hour values are expressed as the absolute value of the fractional part of the number. For example, midnight, 31 December 1899 is represented by 1.0. Similarly, 6 AM, 1 January 1900 is represented by 2.25, and midnight, 29 December 1899 is -1.0. However, 6 AM, 29 December 1899 is -1.25. For FailedSession, gets the date and time when the failed session occurred.

Example	The following example gets the date associated with the first failed session:
	
	`Dim dt`
	`dt = Transmitter.Configurations("Cleanup").FailedSessions(1).Date`

DayOfTheMonth property

Gets and sets the day of the month when the trigger fires.

You must use the TriggerType property to set the trigger type before getting or setting any other trigger properties.

Table

Syntax	*object*.**DayOfTheMonth** [= *value*]
Parameters	value A Short that specifies the day of the month when the trigger fires. 1 = first day of month, 2 = second day of month, etc.
Remarks	The DayOfTheMonth property has the **Short** type.
Example	This example creates a trigger that fires on the first day of each month, starting on January 1, 2009:

```
Dim ta As afaria.TransmitterAccess
Dim trans As afaria.Transmitter
Dim trigger As afaria.trigger
Set ta = CreateObject("Afaria.TransmitterAccess")
Set trans = afaria.GetTransmitterFromAddress
Set trigger = trans.GetTrigger
trigger.TriggerType =
Transmitter.Constant("cmTASK_TIME_TRIGGER_MONTHLYDATE")
trigger.BeginDate = CDate("January 1, 2009")
trigger.DayOfTheMonth = 1
```

DefApprovalDir property

Gets and sets the path for storing the virus definition files that await administor approval.

Table

Syntax	object.**DefApprovalDir** [=Value]
Parameters	Value A string value that specifies a directory on the Afaria server.
Remarks	If the directory name begins with .\ (example: .\Policy\AVFirewall\ClientHold), then the path is relative to the Afaria server's data directory.

Table

Example	This example sets the definition approval directory.
	
	`avfw.DefApprovalDir = "C:\MyAfariaData\AVFWDefHold"`
	
	This example gets the definition approval directory.
	
	`If avfw.DefApprovalDir.StartsWith(".\") Then`
	`  MsgBox("Def approval dir is Afaria\Data\" &`
	`avfw.DefApprovalDir.Substring(2))`
	`Else`
	`  MsgBox("Def approval dir is " & avfw.DefApprovalDir)`
	`End If`

DefaultConfiguration property

Gets and sets the default bandwidth throttling configuration.

Table

Syntax	*object*.**DefaultConfiguration** [= *value*]
Parameters	value A String that represents the configuration.
Remarks	The DefaultConfiguration property has the **String** data type.

Example	This example gets the value from the configuration object:
	
	`Dim config`
	`config =`
	`Transmitter.Configurations("BandwidthThrottling").DefaultConfiguration`

DefaultFailedSessionCleanupSchedule property

Gets the default schedule for automatically cleaning up failed sessions.

Table

Syntax	*object*.**DeletedFailedSessionCleanupSchedule**

Table

Parameters	N/A
Remarks	The DefaultFailedSessionCleanupSchedule property has the **Trigger** type. The DefaultFailedSessionCleanupSchedule property specifies how often the Server should clean up failed sessions.

Example	This example gets the Server's default failed session cleanup schedule:
	
	`Dim config`
	`Set config = Transmitter.Configurations("Cleanup")`
	`Msgbox config.DefaultFailedSessionCleanupSchedule.TriggerString`

DefaultHTTPPort property

Gets the default HTTP port for a Server.

Table

Syntax	*object*.**DefaultHTTPPort**
Parameters	N/A
Remarks	The property has the **Long** type.
Example	This example gets the default HTTP port number for a Server: `Msgbox Transmitter.HTTPConfiguration.DefaultHTTPPort`

DefaultHTTPSPort property

Gets the default HTTPS port for a Server.

Table

Syntax	*object*.**DefaultHTTPSPort**
Parameters	N/A
Remarks	The property has the **Long** type.

Table

Example	This example gets the default HTTPS port number for a Server: `Msgbox Transmitter.SSLCert.DefaultHTTPSPort`

DefaultSSLPort property

Gets the default SSL port for a Server.

Table

Syntax	*object*.**DefaultSSLPort**
Parameters	N/A
Remarks	The property has the **Long** type.
Example	This example gets the default SSL port number for a Server: `Msgbox Transmitter.SSLCert.DefaultSSLPort`

DefHoldForApproval property

Gets and sets the value indicating whether to hold new virus definition files to await administrator approval or to immediately release new files to current antivirus clients.

Table

Syntax	object.**DefHoldForApproval** [=Value]
Parameters	Value An boolean value that specifies the server action for new definition files. True to hold definitions for administrator approval. False to send definitions to clients on their next connection.

Table

Example	This example disables holding new virus definitions for approval.
	
	`avfw.DefHoldForApproval = False`
	
	This example gets the approval setting.
	
	`If avfw.DefHoldForApproval Then`
	`  MsgBox("Virus definitions are being held for approval")`
	`Else`
	`  MsgBox("Virus definitions are being delivered without approval")`
	`End If`

DeletedChannelCleanupSchedule property

Gets and sets the schedule for automatically cleaning up deleted channels.

Table

Syntax	*object*.**DeletedChannelCleanupSchedule** [= *Trigger*]
Parameters	Trigger A Trigger object that represents the new schedule.
Remarks	The DeletedChannelCleanupSchedule property has the **Trigger** type. The deleted channel update schedule specifies how often the Server should cleanup deleted channels.

Example	This example sets the value for the configuration object to once every two days:
	
	`Dim trigger`
	`Set trigger = Transmitter.GetTrigger`
	`trigger.TriggerType = Transmitter.Constant("cmTASK_TIME_TRIGGER_DAILY")`
	`trigger.DailyInterval = 2`
	`Transmitter.Configurations("Cleanup").DeletedChannelCleanupSchedule = trigger`

Description property

Gets and sets the description of an object.

Table

Syntax	*object*.**Description** [= *value*]
Parameters	value A String that represents the description.
Remarks	The Description property has the **String** type.
Example	This example gets the value from the configuration object: `Msgbox Transmitter.Configurations("Contact").Description`

DifferenceFiles property

Gets the DifferenceFiles collection.

Table

Syntax	*object*.**DifferenceFiles**
Parameters	N/A
Remarks	The DifferenceFiles property has the **DifferenceFiles** type. The DifferenceFiles object represents all the files in the Server's difference file cache.
Example	This example gets the entire collection from the configuration object: `Transmitter.Configurations("Difference").DifferenceFiles`

DisableMD5 property

Gets the value of the DisableMD5 value from the certificate settings configuration, which determines whether the MD5 ciphers are disabled or enabled for SSL encryption.

Table

Syntax	*object*.**DisableMD5**
Parameter	object An object expression that evaluates to an object that implements this property.
Remarks	**Boolean** type.
Example	

DomainListNames property

Gets and sets the names in the domain list on the Server Configuration Security page.

Table

Syntax	*object*.**DomainListNames**]
Parameters	N/A
Remarks	The DomainListNames property returns a variant array. It returns the list of domain names set on the Security page.

Example	This example lists all of the domain names listed on the Security page:
	`Dim domains ()`
	`domains () = Transmitter.Configurations("Security").DomainListNames`

DynamicData property

Gets one or more named data values.

Table

Syntax	*object*.**DynamicData** *Names*

Table

Parameters

Names
A String that represents the named data values. Separate names with a semicolon. For example, "TimeNow" or "TimeNow; TimeStarted; FailedSessions". Leading and trailing space characters are ignored. Case is not significant.

Remarks

The DynamicData property has the **VARIANT** type. This is a VARIANT that is a SAFEARRAY of VARIANTs. Each array element can be a different Automation-compatible type, such as Long, String, Boolean, etc. Each array element appears in the same order as requested in the Names string.
The first item in the array is at index 0.
If a named data value is unrecognized or its associated data is unavailable, the array element corresponding to the requested data will be an empty VARIANT. Scripting clients should use the IsEmpty() function to test for this case. C/C++ Clients should test the VARIANT's vt field for the VT_EMPTY flag.

For a Transmitter object, the following named data values are supported:

Table

Name	Type	Description
ActiveSessions	Long	The number of active sessions.
ComputerName	String	The name of the machine where the Server is installed.
DiffCacheUsed	Double	The total amount of disk space used by the File Difference Cache, in bytes.
DiskCapacity	Double	The total capacity of the disk where the Server is installed, in bytes.
DiskFreeSpace	Double	The amount of available space on the disk where the Server is installed, in bytes.
FailedSessions_<hours>	Long	The number of failed sessions in the specified last number of hours.For example, to find the number of failed sessions that occurred in the last day, use FailedSessions_24.
FailedSessionsSinceStartup	Long	The number of failed sessions since the Server service last started.

Table

FileCacheUsed	Double	The total amount of disk space used by the File Compression Cache, in bytes.
TimeNow	Date	The current date and time on the Server.
TimeStarted	Date	The date and time when the Server service was last started.
TotalSessions_<hours>	Long	The number of sessions that ran in the specified last number of hours.For example, to find the total number of sessions that ran in the last day, use TotalSessions_24.
TotalSessionsSinceStartup	Long	The number of sessions that ran since the Server service last started.

Table

Example

This example displays the Server's current date and time, and the number of sessions that ran on the Server in the last 24 hours:

```
Dim ta 'the TransmitterAccess object.
Set ta = CreateObject("Afaria.TransmitterAccess")
Dim t 'the Transmitter object.
Set t = ta.GetTransmitterFromAddress 'use default arguments to
get default transmitter.
Set ta = Nothing 'done with TransmitterAccess object.
Dim data
data = t.DynamicData( "TimeNow; TotalSessions" )
Dim s
If IsEmpty( data(0) ) Then
  MsgBox "Transmitter date and time unavailable"
Else
  MsgBox "Transmitter date and time: " & data(0)
End If
If IsEmpty( data(1) ) Then
  MsgBox "Number of Sessions unavailable"
Else
  MsgBox "Number of Sessions: " & data(1)
End If
```

DynamicDataSet property

Gets a DynamicDataSet object that represents a set of one or more named data values.

Table

Syntax	*object*.**DynamicDataSet** *Names*
Parameters	Names A String that represents the named data values. Separate names with a semicolon. For example, "TimeNow" or "TimeNow; TimeStarted; FailedSessions". Leading and trailing space characters are ignored. Case is not significant.
Remarks	The DynamicDataSet property has the **DynamicDataSet** type. See the DynamicData property for a description of supported data values.

Table

Example DynamicDataSet property

Thisexample uses VBScript to display Server dynamic data on an HTML page in Internet Explorer (requires IE 4.0 or later).

A TextArea HTML element is used to enter the names of the data set. Every two seconds, the GetData subroutine is called. A new DynamicDataSet object is created based on the names in the TextArea, and the current values associated with the data set are retrieved by calling the GetData property. Dynamic HTML is used to write the new data set values to the HTML page.

```
<HTML>

<!--
////////////////////////////////////////////////////////////////////////
////////////
// DynamicData.htm - test script for the DynamicData interface.
//
-->

<!--
////////////////////////////////////////////////////////////////////////
////////////
// VBScript
//
-->
<script language=vbscript>

Option Explicit

'////////////////////////////////////////////////////////////////////////
////////////
'// Global variables
'//

Dim oInterval 'the timer interval object.


Dim ta 'the TransmitterAccess object.
Set ta = CreateObject("Afaria.TransmitterAccess")
Dim t 'the Transmitter object.
Set t = ta.GetTransmitterFromAddress 'use default arguments to get
default transmitter.
Set ta = Nothing 'done with TransmitterAccess object.

Dim oDynamicDataSet
```

Example continued on next page

Table

Example DynamicData Set property continued

```
'////////////////////////////////////////////////////////////////////
///////
'// Event handlers
'//

Sub window_OnLoad
  '// This handler is called when the browser page finishes loading.

  '// Start off with some canned data so user can see something.
  '//
  Form1.txtNames.Value = "TimeNow; TimeStarted; FailedSessions_24; "    &
_
                         "FailedSessionsSinceStartup; ActiveSessions; " & _
                         "TotalSessions_24; TotalSessionsSinceStartup;" & _
                         "ComputerName; DiskCapacity; DiskFreeSpace;"   & _
                           "DiffCacheUsed; FileCacheUsed"
  Call GetData
End Sub

Sub window_OnUnLoad

  '// This handler is called when the window closes.
  window.clearInterval( oInterval )
End Sub
```

Example continued on next page

Table

Example DynamicData Set property continued

```
'////////////////////////////////////////////////////////////////////////
///////
'// Helper routines
'//

Sub GetData
  '// Stop the window timer
  '//
  window.clearInterval( oInterval )

  '// Reset the dynamic data set to the new named values.
  '//
  Dim names
  names = Form1.txtNames.Value
  Set oDynamicDataSet = Nothing
  Set oDynamicDataSet = t.DynamicDataSet( names )

  '// Get the latest values for the data set.
  '//
  Dim data
  data = oDynamicDataSet.GetData

  '// Format a string that contains the name and value of each
  '// dynamic data element, one per line.
  '//
  Dim s, nameArray, i
  nameArray = Split( names, ";", -1, 1 )
  For i = 0 To UBound( data )
    s = s & "<B>" & nameArray(i) & ": </B>"
    If IsEmpty( data(i) ) Then
      s = s & "(unavailable)"
    Else
      s = s & data(i)
    End If
    s = s & "<BR>"

  Next
```

Example continued on next page

Table

Example DynamicData Set property continued

```
    '// Now use DHTML to dynamically replace the <SPAN> element with
    '// the HTML we just created above. The browser will dynamically
    '// parse this new HTML and replace the <SPAN ID=idSpan> element
    '// with this new HTML. Cool!
    '//
    idSpan.innerHTML = s

    '// Restart the window timer to call the GetData subroutine once
    '// every two seconds.
    '//
    oInterval = window.setInterval( "GetData", 2000 )
End Sub

</script>

<BODY>
<H2>DynamicData<HR noshade></H2>

Enter the name(s) of the dynamic data you wish to display.<BR>
For multiple names, use a semicolon (;) between each name.

<FORM id=Form1>
  <TEXTAREA ROWS=5 COLS=50 NAME=txtNames WRAP=on>
  </TEXTAREA>
</FORM>

<B>Note:</B> Page automatically updates once every 2 seconds.
<P>
<SPAN ID=idSpan>this is where the dynamic data values appear</SPAN>

</BODY>
</HTML>
```

EnableBandwidthThrottling property

Gets and sets whether bandwidth throttling is enabled.

Table

Syntax	*object*.**EnableBandwidthThrottling** [= *value*]
Parameters	value A Long that represents the configuration.

Table

Remarks	The EnableBandwidthThrottling property has the **Long** data type.
Example	This example gets the value from the configuration object:

```
Dim configs
Dim enableThrottling
configs = Transmitter.Configurations("BandwidthThrottling")
enableThrottling = configs.EnableBandwidthThrottling
```

EnableCalibration property

Gets and set whether bandwidth throttling calibration is enabled.

Table

Syntax	*object*.**EnableCalibration** [= *value*]
Parameters	value A Long that represents the configuration.
Remarks	The EnableCalibration property has the **Long** data type.
Example	This example gets the value from the configuration object:

```
Dim configs
Dim enableCalibration
configs = Transmitter.Configurations("BandwidthThrottling")
enableCalibration = configs.EnableCalibration
```

EnableClientCert property

Gets and sets whether the server's clients are required to present a certificate to the server for client authentication.

Table

Syntax	*object*.**EnableClientCert** [= *value*]
Parameter	object An object expression that evaluates to an object that implements this property.

Table

Remarks	**Boolean** type.
Example	

Enabled property

Gets whether a tenant is enabled or disabled.

Table

Syntax	*object*.**Enabled**
Parameters	N/A
Remarks	The property has the **Boolean** data type.

Example	This example gets the value from the configuration object:
	`Dim configs`
	`Dim enabled`
	`configs = Transmitter.Configurations("TenantConfigurations")`
	`enabled = configs.Enabled`

EnableEventLogging property

Gets and set whether bandwidth throttling event logging is enabled.

Table

Syntax	*object*.**EnableEventLogging** [= *value*]
Parameters	value A Long that represents the configuration.
Remarks	The EnableEventLogging property has the **Long** data type.

Table

Example	This example gets the value from the configuration object:

```
Dim configs
Dim enableEventLogging
configs = Transmitter.Configurations("BandwidthThrottling")
enableEventLogging = configs.EnableEventLogging
```

EnableFIPS property

Gets and sets whether the server allows only FIPS ciphers for SSL encryption.

Table

Syntax	*object*.**EnableFIPS** [= *value*]
Parameter	object An object expression that evaluates to an object that implements this property.
Remarks	**Boolean** type.
Example	

EnableHTTP property

Gets and sets whether HTTP is enabled.

Table

Syntax	*object*.**EnableHTTP** [= *value*]
Parameters	value A Boolean that specifies whether HTTP is enabled.
Remarks	The EnableHTTP property has the **Boolean** type.
Example	This example sets the value for the configuration object:

```
Transmitter.Configurations("HTTP").EnableHTTP = True
```

EnableHTTPS property

Gets and sets whether HTTP for SSL is enabled.

Table

Syntax	*object*.**EnableHTTPS** [= *value*]
Parameters	value A Boolean that specifies whether HTTP is enabled.
Remarks	The EnableHTTPS property has the **Boolean** type.
Example	This example sets the value for the configuration object: `Transmitter.Configurations("HTTPS").EnableHTTPS = True`

EnableSSL property

Gets and sets whether SSL is enabled.

Table

Syntax	*object*.**EnableSSL** [= *value*]
Parameters	value A Boolean that specifies whether SSL is enabled.
Remarks	The EnableSSL property has the **Boolean** type.
Example	This example sets the value for the configuration object: `Transmitter.Configurations("Security").EnableSSL = True`

EnableUserAuthentication property

Gets and sets whether user authentication is enabled.

Table

Syntax	*object*.**EnableUserAuthentication** [= *value*]

Table

Parameters	value A Boolean that specifies whether User Authentication is enabled.
Remarks	The EnableUserAuthentication property has the **Boolean** type.
Example	This example sets the value for the configuration object: `Transmitter.Configurations("Security").EnableUserAuthentication = True`

EndDate property

Gets and sets the year, month, and day that the trigger deactivates.

You must use the TriggerType property to set the trigger type before getting or setting any other trigger properties.

Table

Syntax	*object*.**EndDate** [= *value*]
Parameters	value A Date that specifies the year, month, and day that a trigger deactivates.
Remarks	The EndDate property has the Date type.

Example	This example creates a trigger that fires once every day, starting on January 1, 2009, and ending on March 1, 2009:
	
	`Dim ta As afaria.TransmitterAccess`
	`Dim trans As afaria.Transmitter`
	`Dim trigger As afaria.trigger`
	`Set ta = CreateObject("Afaria.TransmitterAccess")`
	`Set trans = afaria.GetTransmitterFromAddress`
	`Set trigger = trans.GetTrigger`
	`trigger.TriggerType =`
	`Transmitter.Constant("cmTASK_TIME_TRIGGER_DAILY")`
	`trigger.BeginDate = CDate("January 1, 2009")`
	`trigger.EndDate = CDate("March 1, 2009")`

ExpDate property

Gets the expiration date value in a certificate configuration.

Table

Syntax	*object*.**ExpDate**
Parameters	N/A
Remarks	The ExpDate property has the **String** data type.

Example	This example gets the value from the configuration object for the first item in the collection:
	
	`Dim certificateConfigs, expDate`
	`Set certificateConfigs =`
	`Transmitter.Configurations("Certificate")`
	`expDate = certificateConfigs.Item(1).ExpDate`

ExpirationDate property

Gets the expiration date.

Table

Syntax	*object*.**ExpirationDate** [= *value*]
Parameters	N/A
Remarks	The ExpirationDate property has the **Date** type. For a Licenses object, this method will fail if there is no expiration date associated with the Server or the license is unlimited. VBScript clients should use On Error Resume Next to let the script continue in the event of a failure, as shown in the example.

Table

Example

This example displays the license expiration date for the default Server:

```
Dim t, ta
Set ta = CreateObject("Afaria.TransmitterAccess")
'Use default arguments to get default transmitter.
Set t = ta.GetTransmitterFromAddress
'NOTE: the ExpirationDate property will fail if the license is
'unlimited or something goes wrong. We must handle this
explicitly.
On Error Resume Next
Dim dt
dt = t.Licenses.ExpirationDate
'Explicitly test for success.
If IsDate( dt ) Then
  Msgbox "License expires on " & dt
End If
```

FailedSessionCleanupSchedule property

Gets and sets the schedule for automatically cleaning up failed sessions.

Table

Syntax

object.**FailedSessionCleanupSchedule** [= *Trigger*]

Parameters

Trigger
A Trigger object that represents the new schedule.

Remarks

The FailedSessionCleanupSchedule property has the **Trigger** type.
The failed session cleanup schedule specifies how often the Server should cleanup failed sessions.

Example

This example sets the value for the configuration object to once every two days:

```
Dim trigger
Set trigger = Transmitter.GetTrigger
trigger.TriggerType =
Transmitter.Constant("cmTASK_TIME_TRIGGER_DAILY")
trigger.DailyInterval = 2
Transmitter.Configurations("Cleanup").FailedSessionCleanupSchedule =
trigger
```

FailedSessions property

Gets the FailedSessions collection.

Table

Syntax	*object*.**FailedSession**
Parameters	N/A
Remarks	The FailedSessions property has the **FailedSessions** type. The FailedSessions object represents the collection of failed sessions associated with a Server. Each item in this collection is represented by a FailedSession object.
Example	This example gets the collection of all failed sessions: `Transmitter.Configurations("Cleanup").FailedSessions`

FarmID property

Gets the Farm ID for the server's farm. The FarmID property for all servers is the same as the master server's Server ID property.

Table

Syntax	*object*.**FarmID**
Parameters	N/A
Remarks	The property has the **String** data type.
Example	

FileSize property

Gets the size (in bytes) of a file.

Table

Syntax	*object*.**FileSize**

Table

Parameters	N/A
Remarks	The FileSize property has the **Double** type. The following table summarizes the results of using the FileSize property for each object listed:

Table

Object	Results
CachedFile	Gets the size of the compressed cached file. For files that are not compressible, a value of zero is returned.
DifferenceFile	Gets the total size of all stored versions of a file.
Document	Gets the size of the document. **Note**: This value is zero if the file is located on the client (as defined by the ClientIsSource property).
FileVersion	Gets the total size of all stored versions of a file.

Table

Example — This example displays the sizes of all compressed files in the Server's CachedFiles collection:

```
Dim cachedFiles, i
Set cachedFiles =
Transmitter.Configurations("Cache").CachedFiles
For i = 1 To cachedFiles.Count
    MsgBox cachedFiles.Item(i).FileSize
Next
```

FileVersionCount property

Gets the number of file versions available for a file.

Table

Syntax	*object*.**FileVersionCount**

Table

Parameters	N/A
Remarks	The FileVersionCount property has the **Long** type.

Example	This example displays the number of versions available for each file in the Server's DifferenceFiles collection:
	
	`Dim diffFiles, diffFile`
	`Set diffFiles =`
	`Transmitter.Configurations("Difference").DifferenceFiles`
	`For Each diffFile In diffFiles`
	`  MsgBox diffFile.FileVersionCount`
	`Next`

FileVersionInfo property

Gets the file version information for a file.

Table

Syntax	*object*.**FileVersionInfo**
Parameters	N/A
Remarks	The FileVersionInfo property has the **String** type. The file version information specifies the binary version number of a file. The binary version number consists of two 32-bit integers represented as a dotted decimal String--for example, "1.2.3.4" or "3.10".

Example	This example displays the version information of all files in the Server's DifferenceFiles collection:
	
	`Dim diffFiles, diffFile`
	`Set diffFiles =`
	`Transmitter.Configurations("Difference").DifferenceFiles`
	`For Each diffFile In diffFiles`
	`  MsgBox diffFile.FileVersionInfo`
	`Next`

FileVersions property

Gets the FileVersions collection.

Table

Syntax	*object*.**FileVersions**
Parameters	N/A
Remarks	The FileVersions property has the **FileVersions** type. The FileVersions object represents all the versions of a file in the Server's difference file cache.

Example	This example gets the collection of file versions for all files in the Server's DifferenceFiles collection:
	
	`Dim diffFiles, diffFile, fileVersions`
	`Set diffFiles =`
	`Transmitter.Configurations("Difference").DifferenceFiles`
	`For Each diffFile In diffFiles`
	`  Set fileVersions = diffFile.FileVersions`
	`Next`

FullName property

Gets the full path.

Table

Syntax	*object*.**FullName**
Parameters	N/A
Remarks	The FullName property has the **String** type. The following table summarizes the results of using the FullName property with the objects listed:

Table

Object	Results
CachedFile	Gets the full path to the cached file.

Table

ChannelItem	Gets the fully qualified name of the channel item, relative to the root folder. Note that this does not represent an actual file path.For example: "\Folder1\Channel3."
DifferenceFile	Gets the full path to the difference file.
FileVersion	Gets the full path to the difference file.
Transmitter	Gets the full path to the Server executable, for example: "C:\Program Files\Beyond\bin\XService.exe".

Table

Example	This example displays the full path to the Server: `MsgBox Transmitter.FullName`

See also **Name property on page 193** and **Path property on page 197**.

GetData property

Gets the current values for the data set.

Table

Syntax	*object*.**GetData**
Parameters	N/A
Remarks	The GetData property has the **VARIANT** type. This is a VARIANT that is a SAFEARRAY of VARIANTs. Each array element can be a different Automation-compatible type, such as Long, String, Boolean, etc. For more information, see the DynamicData property and the DynamicDataSet property.

Table

Example	This example uses a dynamic data set to display the Server's current date and time, and the number of sessions that ran on the Server in the last 24 hours:

```
Dim ta 'the TransmitterAccess object.
Set ta = CreateObject("Afaria.TransmitterAccess")
Dim t 'the Transmitter object.
Set t = ta.GetTransmitterFromAddress 'use default arguments to
get default transmitter.
Set ta = Nothing 'done with TransmitterAccess object.
Dim dataSet
Set dataSet = t.DynamicDataSet( "TimeNow; TotalSessions" )
Dim data
data = dataSet.GetData
Dim s
If IsEmpty( data(0) ) Then
  MsgBox "Server date and time unavailable"
Else
  MsgBox "Server date and time: " & data(0)
End If

If IsEmpty( data(1) ) Then
  MsgBox "Number of Sessions unavailable"
Else
  MsgBox "Number of Sessions: " & data(1)
End If
```

GetTrigger property

Gets an uninitialized Trigger object.

Table

Syntax	*object*.**GetTrigger**
Parameters	N/A
Remarks	The GetTrigger property has the **Trigger** type. The Trigger object represents start times, repetition criteria, and other schedule-related information. You must set the trigger type before getting or setting any other trigger properties.

Table

Example	This example creates an uninitialized Trigger object:
	
	`Dim Trig`
	`Set Trig = Transmitter.GetTrigger`

Hidden property

Gets and sets whether an object is hidden.

Table

Syntax	*object*.**Hidden** [= *value*]
Parameters	value A Boolean that specifies whether the object is hidden. A value of True hides the object. A value of False shows the object.
Remarks	The Hidden property has the **Boolean** type. System objects are automatically created and owned by the Server. These objects are typically not visible to the user.
Example	This example hides the channel: `channel.Hidden = True`

HitRate property

Gets the number of times the file was retrieved.

Table

Syntax	*object*.**HitRate**
Parameters	N/A
Remarks	The HitRate property has the **Long** type.

Table

Example	This example displays how many times each file in the Server's CachedFiles collection was retrieved:
	
	`Dim cachedFiles, i`
	`Set cachedfiles =`
	`Transmitter.Configurations("Cache").CachedFiles`
	`For i = 1 To cachedFiles.Count`
	`  MsgBox cachedFiles.Item(i).HitRate`
	`Next`

HTMLButtonImage property

Gets and sets the HTML button image associated with an object.

Changing any HTML property requires you to change the corresponding HTML code in your Web pages.

Table

Syntax	*object*.**HTMLButtonImage** [= *value*]
Parameters	value A String that specifies the path to the new button image.
Remarks	The HTMLButtonImage property has the **String** type. The HTMLButtonImage property is only valid when the HTMLControlType property is cmHTMLCTText or cmHTMLCTButton.

Example	This example gets the button image associated with all channel sets:
	
	`Dim channelItems`
	`Set channelItems = Transmitter.ChannelItems`
	`For Each item in channelItems`
	`  If ( "Channel Set" = item.Type ) Then`
	`  MsgBox item.HTMLButtonImage`
	`  End If`
	`Next`

HTMLButtonText property

Gets and sets the HTML button text associated with an object.

Changing any HTML property requires you to change the corresponding HTML code in your Web pages.

Table

Syntax	*object*.**HTMLButtonText** [= *value*]
Parameters	value A String that specifies the new button text.
Remarks	The HTMLButtonText property has the **String** type. The HTMLButtonText property is only valid when the HTMLControlType property is cmHTMLCTText or cmHTMLCTButton.
Example	This example gets the button text associated with all channel sets:

```
Dim channelItems
Set channelItems = Transmitter.ChannelItems
For Each item in channelItems
  If ( "Channel Set" = item.Type ) Then
    MsgBox item.HTMLButtonText
  End If
Next
```

HTMLCloseImmediately property

Gets and sets whether the Client status window closes immediately after a session completes.

Table

Syntax	*object*.**HTMLCloseImmediately** [= *value*]
Parameters	value A Boolean that specifies whether the Client status window closes immediately after a completed session. A value of True closes the status window immediately.
Remarks	The HTMLCloseImmediately property has the **Boolean** type.

Table

Example	This example gets the HTMLCloseImmediately property associated with all channel sets:
	
	`Dim channelItems`
	`Set channelItems = Transmitter.ChannelItems`
	`For each item in channelItems`
	`  If (“Channel Set” = Item.Type) then`
	`  Msgbox item.HTMLCloseImmediately`
	`  End if`
	`Next`

HTMLCode property

Gets the HTML code associated with an object.

Table

Syntax	*object*.**HTMLCode**
Parameters	N/A
Remarks	The HTMLCode property has the **String** type. The Server automatically generates the HTML code to run the channel or channel set on the Client from a Web page. Simply copy the generated HTML code into your Web page.

Example	This example gets the HTML code associated with all channel sets:
	
	`Dim channelItems`
	`Set channelItems = Transmitter.ChannelItems`
	`For Each item in channelItems`
	`  If ( "Channel Set" = item.Type ) Then`
	`  MsgBox item.HTMLCode`
	`  End If`
	`Next`

HTMLControlType property

Gets and sets the HTML control type associated with an object.

Changing any HTML property requires you to change the corresponding HTML code in your Web pages.

Table

Syntax	*object*.**HTMLControlType** [= *value*]
Parameters	value An enumeration of type *cmHTMLControlTypeEnum* that specifies how the channel or channel set published on a Web page runs on the Client.
Remarks	The HTMLControlType property has the **cmHTMLControlTypeEnum** type.
Example	This example gets the HTML control type associated with all channel sets:

```
Dim channelItems
Set channelItems = Transmitter.ChannelItems
For Each item in channelItems
  If ( "Channel Set" = item.Type ) Then
    MsgBox item.HTMLControlType
  End If
Next
```

HTMLDisableMessages property

Gets and sets whether messages are disabled on the Client.

Changing any HTML property requires you to change the corresponding HTML code in your Web pages.

Table

Syntax	*object*.**HTMLDisableMessages** [= *value*]
Parameters	value A Boolean that specifies whether messages are disabled on the Client. A value of *True* disables messages. A value of *False* enables messages.
Remarks	The HTMLDisableMessages property has the **Boolean** type.

Table

Example	This example gets the HTML disable messages property associated with all channel sets:
	
	`Dim channelItems`
	`Set channelItems = Transmitter.ChannelItems`
	`For Each item in channelItems`
	`  If ( "Channel Set" = item.Type ) Then`
	`  MsgBox item.HTMLDisableMessages`
	`  End If`
	`Next`

HTMLHideMessages property

Gets and sets whether messages are hidden on the Client.

Changing any HTML property requires you to change the corresponding HTML code in your Web pages.

Table

Syntax	*object*.**HTMLHideMessages** [= *value*]
Parameters	value A Boolean that specifies whether messages are hidden on the Client. A value of *True* hides messages. A value of *False* shows messages.
Remarks	The HTMLHideMessages property has the **Boolean** type.

Example	This example gets the HTML hide messages property associated with all channel sets:
	
	`Dim channelItems`
	
	`Set channelItems = Transmitter.ChannelItems`
	`For Each item in channelItems`
	`  If ( "Channel Set" = item.Type ) Then`
	`  MsgBox item.HTMLHideMessages`
	`  End If`
	`Next`

HTMLHideStatus property

Gets and sets the visibility of the Client status window.

Changing any HTML property requires you to change the corresponding HTML code in your Web pages.

Table

Syntax	*object*.**HTMLHideStatus** [= *value*]
Parameters	value A Boolean that specifies whether to show or hide the client status window. A value of *True* hides the status window. A value of *False* shows the status window.
Remarks	The HTMLHideMessages property has the **Boolean** type.
Example	This example gets the HTML hide status associated with all channel sets:

```
Dim channelItems
Set channelItems = Transmitter.ChannelItems
For Each item in channelItems
  If ( "Channel Set" = item.Type ) Then
    MsgBox item.HTMLHideStatus
  End If
Next
```

HTTPSPort property

Gets and sets the HTTPS port for the Server.

Table

Syntax	*object*.**HTTPSPort** [= *value*]
Parameters	value A Long that specifies the new HTTPS number
Remarks	The HTTPSPort property has the **Long** type.
Example	This example sets the HTTPS port number for a Server:

```
Transmitter.SSLCert.HTTPSPort = 443
```

ID property

Gets the identifier associated with an object.

Table

Syntax	*object.***ID**
Parameters	N/A
Remarks	The ID property has the **Long** type. The ID property represents an identifier that uniquely distinguishes an object on a Server. For ChannelItem and Content objects, the ID is assigned by the system when the object is created and is never reused.
Example	This example gets the ID associated with the first channel item: `Transmitter.ChannelItems(1).ID`

InventoryOptions property

Defines the type of inventory scan (hardware-only or both hardware and software) for an Inventory Manager channel, where "Inventory" refers to the ContentType property type.

Table

Syntax	object.**InventoryOptions** [=ScanType]
Parameters	ScanType A String that represents the type of scan. Valid scan types are: “Both” and “HW”.
Example	This example defines the scan type as both hardware and software: `Channel.InventoryOptions ="Both"`

IssuerAddress property

Gets the issuer address value in a certificate configuration.

Table

Syntax	*object*.**IssuerAddress**
Parameters	N/A
Remarks	The IssuerAddress property has the **String** data type.

Example	This example gets the value of issuer address in the first certificate configuration:
	
	`Dim certificateConfigs, issuerAddr`
	`Set certificateConfigs =`
	`Transmitter.Configurations("Certificate")`
	`issuerAddr = certificateConfigs.Item(1).IssuerAddress`

IssuerCommonName property

Gets the issuer common name value in a certificate configuration.

Table

Syntax	*object*.**IssuerCommonName**
Parameters	N/A
Remarks	The IssuerCommonName property has the **String** data type.

Example	This example gets the value of issuer common name in the first certificate configuration:
	
	`Dim certificateConfigs, issuerCommonName`
	`Set certificateConfigs =`
	`Transmitter.Configurations("Certificate")`
	`issuerCommonName = certificateConfigs.Item(1).IssuerCommonName`

IssuerCountry property

Gets the issuer country value in a certificate configuration.

Table

Syntax	*object*.**IssuerCountry**
Parameters	N/A
Remarks	The IssuerCountry property has the **String** data type.
Example	This example gets the value of issuer country in the first certificate configuration:

```
Dim certificateConfigs, issuerCountry
Set certificateConfigs =
Transmitter.Configurations("Certificate")
issuerCountry = certificateConfigs.Item(1).IssuerCountry
```

IssuerLocality property

Gets the issuer locality value in a certificate configuration.

Table

Syntax	*object*.**IssuerLocality**
Parameters	N/A
Remarks	The IssuerLocality property has the **String** data type.
Example	This example gets the value of issuer locality in the first certificate configuration:

```
Dim certificateConfigs, issuerLocality
Set certificateConfigs =
Transmitter.Configurations("Certificate")
issuerLocality = certificateConfigs.Item(1).IssuerLocality
```

IssuerOrgName property

Gets the issuer organization name value in a certificate configuration.

Table

Syntax	*object*.**IssuerOrgName**
Parameters	N/A
Remarks	The IssuerOrgName property has the **String** data type.

Example	This example gets the value of issuer organization name in the first certificate configuration:
	
	`Dim certificateConfigs, issuerOrgName`
	`Set certificateConfigs =`
	`Transmitter.Configurations("Certificate")`
	`issuerOrgName = certificateConfigs.Item(1).IssuerOrgName`

IssuerState property

Gets the issuer state value in a certificate configuration.

Table

Syntax	*object*.**IssuerState**
Parameters	N/A
Remarks	The IssuerState property has the **String** data type.

Example	This example gets the value of issuer state in the first certificate configuration:
	
	`Dim certificateConfigs, issuerState`
	`Set certificateConfigs =`
	`Transmitter.Configurations("Certificate")`
	`issuerState = certificateConfigs.Item(1).IssuerState`

IssuerUnit property

Gets the issuer unit value in a certificate configuration.

Table

Syntax	*object*.**IssuerUnit**
Parameters	N/A
Remarks	The IssuerUnit property has the **String** data type.

Example	This example gets the value of issuer unit in the first certificate configuration:
	
	`Dim certificateConfigs, issuerUnit`
	`Set certificateConfigs =`
	`Transmitter.Configurations("Certificate")`
	`issuerUnit = certificateConfigs.Item(1).IssuerUnit`

KeyType property

Gets the key type value in a certificate configuration.

Table

Syntax	*object*.**KeyType**
Parameters	N/A
Remarks	The KeyType property has the **String** data type.

Example	This example gets the value of key type in the first certificate configuration:
	
	`Dim certificateConfigs, keyType`
	`Set certificateConfigs =`
	`Transmitter.Configurations("Certificate")`
	`keyType = certificateConfigs.Item(1).KeyType`

LastAccessed property

Gets the date and time when an object was last accessed.

Table

Syntax	*object*.**LastAccessed**
Parameters	N/A
Remarks	The LastAccessed property has the **Date** type. The Date type is implemented using an 8-byte floating-point number. Days are represented by whole number increments starting with 30 December 1899, midnight as time zero. Hour values are expressed as the absolute value of the fractional part of the number. For example, midnight, 31 December 1899 is represented by 1.0. Similarly, 6 AM, 1 January 1900 is represented by 2.25, and midnight, 29 December 1899 is -1.0. However, 6 AM, 29 December 1899 is -1.25.

Example	This example displays the last accessed date of each compressed file in the Server's CachedFiles collection:
	
	`For i = 1 to CachedFiles.Count`
	`  MsgBox CachedFiles.Item(i).LastAccessed`
	`Next`

LastChecked property

Gets the date and time when an object was last synchronized with its original object.

Table

Syntax	*object*.**LastChecked**
Parameters	N/A
Remarks	The LastChecked property has the **Date** type. The Date type is implemented using an 8-byte floating-point number. Days are represented by whole number increments starting with 30 December 1899, midnight as time zero. Hour values are expressed as the absolute value of the fractional part of the number. For example, midnight, 31 December 1899 is represented by 1.0. Similarly, 6 AM, 1 January 1900 is represented by 2.25, and midnight, 29 December 1899 is -1.0. However, 6 AM, 29 December 1899 is -1.25.

Table

Example	This example displays the last checked date of each compressed file in the Server's CachedFiles collection:
	
	`For i = 1 to CachedFiles.Count`
	`  MsgBox CachedFiles.Item(i).LastChecked`
	`Next`

LastRefreshTime property

Gets the last time the cache was refreshed.

Table

Syntax	*object*.**LastRefreshTime**
Parameters	N/A
Remarks	The LastRefreshTime property has the **Date** type. The last refresh time specifies the last time the cache was refreshed.

Example	This example gets the LastRefreshTime of a CacheConfiguration object:
	
	`Dim config`
	`Set config = Transmitter.Configurations("Cache")`
	`Msgbox config.LastRefreshTime`

LastUpdated property

Gets the date and time of the most recent change made to an object.

Table

Syntax	*object*.**LastUpdated**
Parameters	N/A

Table

Remarks	The LastUpdated property has the **Date** type. The Date type is implemented using an 8-byte floating-point number. Days are represented by whole number increments starting with 30 December 1899, midnight as time zero. Hour values are expressed as the absolute value of the fractional part of the number. For example, midnight, 31 December 1899 is represented by 1.0. Similarly, 6 AM, 1 January 1900 is represented by 2.25, and midnight, 29 December 1899 is -1.0. However, 6 AM, 29 December 1899 is -1.25.

Example	This example gets the last updated time for all channel sets:
	
	`Dim channelItems, item`
	`Set channelItems = Transmitter.ChannelItems`
	`For Each item in channelItems`
	`  If ( "Channel Set" = item.Type ) Then`
	`  MsgBox item.LastUpdated`
	`  End If`
	`Next`

LDAPAssignmentNodeTypes property

Gets and sets LDAP assignment node types.

Table

Syntax	*object*.**LDAPSearchRoot**
Parameters	value List of cmLDAPNodeTypesEnum
Remarks	The LDAPAssignmentNodeTypes property has the **cmLDAPNodeTypesEnum** type.

Example	The following example gets the LDAP assignment node types:
	
	`Dim config`
	`Set config = Transmitter.Configurations("Security")`
	`MsgBox config.LDAPAssignmentNodeTypes`

LDAPSearchRoot property

Gets where to start searching for the LDAP server.

Table

Syntax	*object*.**LDAPSearchRoot**
Parameters	N/A
Remarks	The LDAPSearchRoot property has the **String** type.
Example	This example gets the location where to start searching for the LDAP server: `Transmitter.Configurations("Security").LDAPSearchRoot`

LicenseKey property

Gets the license key.

Table

Syntax	*object*.**LicenseKey**
Parameters	N/A
Remarks	The LicenseKey property has the **String** type.

Example	This example displays the license key for the default Server:
	`Dim t, ta`
	`Set ta = CreateObject("Afaria.TransmitterAccess")`
	`'Use default arguments to get default transmitter.`
	`Set t = ta.GetTransmitterFromAddress`
	`Msgbox t.Licenses.LicenseKey`

Licenses property

Gets the Licenses collection.

Table

Syntax	*object*.**Licenses**
Parameters	N/A
Remarks	The Licenses property has the **Licenses** type. The Licenses object represents the collection of licenses associated with a Server. Each license in this collection is represented by a LicensedComponent object.

Example	This example displays the type of each Server license in the Licenses collection:
	
	`For i = 1 to Licenses.Count`
	`  MsgBox Licenses.Item(i).Type`
	`Next`

MasterCopyID property

Gets the identifier associated with an object's master copy.

If the channel is a working copy of another channel, the MasterCopyID property is the ID of the original channel; otherwise, its value is zero (0).

Table

Syntax	*object*.**MasterCopyID**
Parameters	N/A
Remarks	The MasterCopyID property has the **Long** type. The MasterCopyID property represents an identifier that uniquely distinguishes an object's master copy on a Server. This identifier is assigned by the system when the object is created, and is never reused.

Example	This example gets the master copy ID associated with the first channel item:
	
	`Dim channel`
	`Set channel = Transmitter.ChannelItems(1)`
	`If ( "Channel" = channel.Type ) Then`
	`  MsgBox channel.MasterCopyID`
	`End If`

MaximumClientThroughput property

Gets and sets the maximum Client throughput value in a Bandwidth throttling configuration.

Table

Syntax	*object.* **MaximumClientThroughput [= value]**
Parameters	value A Long value representing the maximum Client throughput.
Remarks	The MaximumClientThroughput property has the **Long** data type. This property is typically used to

Example	This example gets the value for the maximum Client throughput from the first bandwidth throttling configuration:
	
	`Dim configs`
	`Dim maxThroughput`
	`Set configs = Transmitter.Configurations("BandwidthThrottling")`
	`Set maxThroughput = configs.Item(1).MaximumClientThroughput`

MinimumClientThroughput property

Gets and sets the minimum Client throughput value in a Bandwidth Throttling configuration.

Table

Syntax	*object.* **MinimumClientThroughput [= value]**
Parameters	value A Long value representing the minimum Client throughput.
Remarks	The MinimumClientThroughput property has the **Long** data type. This property is typically used to

Example	This example gets the value for the minimum Client throughput from the first bandwidth throttling configuration:
	
	`Dim configs`
	`Dim MinThroughput`
	`Set configs = Transmitter.Configurations("BandwidthThrottling")`
	`Set MinThroughput = configs.Item(1).MinimumClientThroughput`

MinutesDuration property

Gets and sets the number of minutes after the trigger starts that the trigger will remain active.

You must use the TriggerType property to set the trigger type before getting or setting any other trigger properties.

Table

Syntax	*object*.**MinutesDuration** [= *value*]
Parameters	value A Long that specifies the number of minutes after the trigger starts that the trigger will remain active. For example, if the trigger starts at 8:00 A.M. and you want to repeatedly fire the trigger until 5:00 P.M., there would be 540 minutes in the duration.
Remarks	The MinutesDuration property has the **Long** type.
Example	This example creates a trigger that fires once every day, starting at 1:00 AM on January 1, 2009 and lasting for 15 minutes:

```
Dim ta As afaria.TransmitterAccess
Dim trans As afaria.Transmitter
Dim trigger As afaria.trigger
Set ta = CreateObject("Afaria.TransmitterAccess")
Set trans = afaria.GetTransmitterFromAddress
Set trigger = trans.GetTrigger
trigger.TriggerType =
Transmitter.Constant("cmTASK_TIME_TRIGGER_DAILY")
trigger.BeginDate = CDate("January 1, 2009")
trigger.StartHour = 1
trigger.MinutesDuration = 15
```

MinutesInterval property

Gets and sets the number of minutes between consecutive triggers.

You must use the TriggerType property to set the trigger type before getting or setting any other trigger properties.

Table

Syntax	*object*.**MinutesInterval** [= *value*]

Table

Parameters	value A Long that specifies the number of minutes between consecutive triggers. This number is counted from the start of the previous trigger. For example, to fire a trigger every hour from 8:00 A.M. to 5:00 P.M., set this field to 60.
Remarks	The MinutesInterval property has the **Long** type.
Example	This example creates a trigger that fires every hour of every day, starting at 1:00 AM, on January 1, 2009:

```
Dim ta As afaria.TransmitterAccess
Dim trans As afaria.Transmitter
Dim trigger As afaria.trigger
Set ta = CreateObject("Afaria.TransmitterAccess")
Set trans = afaria.GetTransmitterFromAddress
Set trigger = trans.GetTrigger
trigger.TriggerType =
Transmitter.Constant("cmTASK_TIME_TRIGGER_DAILY")
trigger.BeginDate = CDate("January 1, 2009")
trigger.StartHour = 1
trigger.MinutesInterval = 60
```

MonthlyInterval property

Gets and sets the interval between triggers, in months.

You must use the TriggerType property to set the trigger type before getting or setting any other trigger properties.

Table

Syntax	*object*.**MonthlyInterval** [= *value*]
Parameters	value A Short that specifies the interval between triggers, in months. 1 = every month, 2 = every other month, etc.
Remarks	The MonthlyInterval property has the **Short** type.

Table

Example	This example creates a trigger that fires every other month, starting on January 1, 2009:
	
	`Dim ta As afaria.TransmitterAccess`
	`Dim trans As afaria.Transmitter`
	`Dim trigger As afaria.trigger`
	`Set ta = CreateObject("Afaria.TransmitterAccess")`
	`Set trans = afaria.GetTransmitterFromAddress`
	`Set trigger = trans.GetTrigger`
	`trigger.TriggerType =`
	`Transmitter.Constant("cmTASK_TIME_TRIGGER_MONTHLYINTERVAL")`
	`trigger.BeginDate = CDate("January 1, 2009")`
	`trigger.MonthlyInterval = 2`

MsgLogSettings property

Gets and sets the message logging policies in a logging configuration.

Table

Syntax	*object*.MsgLogSettings [=value]
Parameters	value An Integer value representing the MsgLogSettings.
Remarks	The MsgLogSettings property has the **Integer** data type. This property is typically used to

Example	This example gets the value for the message logging policies from the Logging configuration:
	
	`Dim msgLogSettings`
	`msgLogSettings =`
	`Transmitter.Configurations("Logging").MsgLogSettings`

Name property

Gets or gets and sets the name of an object. The Name property is typically the default property. Accordingly, you do not have to reference Name explicitly, as shown in the syntax.

The Name property does not get an object's drive and directory. To get the drive and directory, use the Path or FullName property.

Table

Syntax	*object*.**Name** [= *value*] ‾ or ‾ object
Parameters	value A String that represents the new name.
Remarks	The Name property has the **String** type. The Name property can get or set according to its object: • CachedFile – Gets • ChannelItem – Gets and sets; You cannot change the name of a Channel object if it is a working copy • ContactConfiguration – Gets and sets • DifferenceFile – Gets • Document – Gets • FileVersion – Gets • Profile – Gets • TenantConfiguration – Gets • Transmitter – Gets • TransmitterConfiguration – Gets and sets • TransmitterDescription – Gets • WorkObject – Gets and sets
Example	This example gets the name of the current Channel: `Channel.Name`

Parent property

Gets the parent of an object.

Table

Syntax	*object*.**parent**
Parameters	N/A
Remarks	The Parent property's type depends on the object you access the property from (see the table below). Use the Parent property to access the properties and methods of an object's parent. The following table shows the parent of each object:

Table

Object	Parent object
CachedFile CachedFiles	CacheConfiguration
Channel ChannelItem ChannelItems ChannelSet ChannelSetMembers	Transmitter
CacheConfiguration CleanupConfiguration ContactConfiguration Configuration Configurations DifferenceConfiguration HTTPConfiguration SecurityConfiguration TransmitterConfiguration	Transmitter
Content	Channel
DependentDocuments	Channel
DifferenceFile DifferenceFiles	DifferenceConfiguration
Document Documents	Channel

Table

FailedSession FailedSessions	CleanupConfiguration
FileVersion FileVersions	DifferenceFile
Folder	Transmitter
LicensedComponent	Transmitter
Licenses	Transmitter
WorkObjectEvent WorkObject	WorkObject
WorkObject	Channel (for a Work Object assigned to a Session Channel) Transmitter (for all other Work Objects)
WorkObjects	Channel (for Work Objects assigned to a Session Channel)Transmitter (for Work Objects associated with a Transmitter)

Table

Example	This example gets the parent of the Configurations collection: `Transmitter.Configurations.Parent`

ParentFolder property

Gets an object's parent folder.

Table

Syntax	*object*.**ParentFolder**
Parameters	N/A
Remarks	The ParentFolder property has the **Folder** type.
Example	This example gets the ID associated with the first channel item's parent folder: `Transmitter.ChannelItems(1).ParentFolder.ID`

PasswordEncoded property

Gets and sets the encoded password for an object.

Table

Syntax	*object*.**PasswordEncoded** [= *value*]
Parameters	value A String that represents the new encoded password.
Remarks	The PasswordEncoded property has the **String** type.
Example	This example gets the encoded password for all channels:

```
Dim channelItems
Set channelItems = Transmitter.ChannelItems
For Each item in channelItems
  If ( "Channel" = item.Type ) Then
    MsgBox item.PasswordEncoded
  End If
Next
```

PasswordPlain property

Sets the plain text password for an object.

Table

Syntax	*object*.**PasswordPlain** [= *value*]
Parameters	value A String that represents the new plain text password.
Remarks	The PasswordPlain property is a write-only property.
Example	This example sets the plain text password for all channels:

```
Dim channelItems
Set channelItems = Transmitter.ChannelItems
For Each item in channelItems
  If ( "Channel" = item.Type ) Then
    item.PasswordPlain = "secret"
  End If
Next
```

PasswordProtected property

Gets and sets whether the object is password protected.

Table

Syntax	*object*.**PasswordProtected** [= *value*]
Parameters	value A Boolean that specifies whether the object is password protected. A value of True enables password protection. A value of False disables password protection.
Remarks	The PasswordProtected property has the **Boolean** type.

Example	This example disables password protection for all channels:
	
	`Dim channelItems`
	`Set channelItems = Transmitter.ChannelItems`
	`For Each item in channelItems`
	`  If ( "Channel" = item.Type ) Then`
	`  item.PasswordProtected = False`
	`  End If`
	`Next`

Path property

Gets the path to an object. This path never ends with a backslash, unless the path has the format "C:\".

Table

Syntax	*object*.**Path**
Parameters	N/A
Remarks	The Path property has the **String** type. The Path property does not get an object's file name and extension. To get the file name and extension, use the Name or FullName property. The Path property does not end the path with a backslash, unless the path has the format "C:\". The following table summarizes the results of using the Path property with the objects listed:

Table

Object	Results

Table

CachedFile	Gets the path to the compressed file.
CachedFiles	Gets the path to the Server's compressed file cache.
ChannelItem	Gets the full name of the channel item's parent folder.
DifferenceFile	Gets the path to the difference file.
FileVersion	Gets the path to the difference file.
Transmitter	Gets the path to the Server's executable directory, for example: "C:\Program Files\Afaria\bin".

Table

Example — This example gets the path to the Server:

```
Dim PathString
PathString = Transmitter.Path
```

PercentDiskSpace property

Gets and sets the amount of disk space used by the Server's compressed file cache or the Server's difference file cache.

Table

Syntax	*object*.**PercentDiskSpace** [= *value*]
Parameters	value A Long that represents the maximum amount of disk space to use. Valid values are 1 to 100, inclusive.
Remarks	The PercentDiskSpace property has the **Long** type. The compressed file cache is located on the disk where the Server is installed. Use the Path property to determine where the Server is installed.
Example	This example sets the Server's compressed file cache size to 25% of the disk size:

```
Transmitter.Configurations("Cache").PercentDiskSpace = 25
```

PhoneNumber property

Gets or sets the phone number.

Table

Syntax	*object*.**PhoneNumber**
Parameters	Value A String that represents the phone number.
Remarks	The PhoneNumber property has the **String** type.
Example	This example gets the phone number of the Server's contact information: `Transmitter.Configurations("Contact").PhoneNumber`

Port property

Gets and sets the port number.

Table

Syntax	*object*.**Port** [=*value*]
Parameters	value A Long that specifies the new port number.
Remarks	The Port property has the **Long** type. The following table summarizes the results of using the Port property with the objects listed:

Table

Object	Results
HTTPConfiguration	Gets and sets the HTTP port number used by Clients to connect to the Server. The default value is 80.
TransmitterConfiguration	Gets and sets the TCP port number used by Clients to connect to the Server. A TCP port number uniquely identifies an application on a machine. Typically, port numbers 1 thru 1024 are reserved.

Table

Example	This example sets the TCP port number for a Server. `Transmitter.Configurations("Transmitter").Port = 3555`

Profiles property

Gets the Profiles collection, which you can enumerate to access individual Profile objects. Returns an obect which implements the IProfiles enumerator interface.

Table

Syntax	*object*.**Profiles**
Parameters	N/A
Example	

PubKey property

Gets the public key value in a certificate configuration.

Table

Syntax	*object*.**PubKey**
Parameters	N/A
Remarks	PubKey property has the **String** data type. This property is typically used to

Example	This example gets the value of public key in the first certificate configuration:
	
	`Dim certificateConfigs, pubKey`
	`Set certificateConfigs =`
	`Transmitter.Configurations("Certificate")`
	`pubKey = certificateConfigs.Item(1).PubKey`

Published property

Gets whether an object is published.

Publishing a working copy of a channel will overwrite and publish the original copy, even if the original is unpublished.

Table

Syntax	*object*.**Published**
Parameters	N/A
Remarks	The Published property has the **Boolean** type. You must publish a channel to make it available for Clients.

Example	This example publishes a channel if it is unpublished:
	
	`If NOT channel.Published Then`
	`  channel.SetPublish( True )`
	`End If`

ReadOnly property

Gets the ReadOnly state value in a bandwidth throttling configuration.

Table

Syntax	*object*.**ReadOnly**
Parameters	value A Long value representing the ReadOnly state.
Remarks	The ReadOnly property has the **Long** data type. This property is typically used to

Example	This example gets the value for the read-only state from the first bandwidth throttling configuration:
	
	`Dim configs`
	`Dim readOnly`
	`Set configs = Transmitter.Configurations("BandwidthThrottling")`
	`Set readOnly = configs.Item(1).ReadOnly`

RepLogSettings property

Gets and sets the replication logging policies in a logging configuration.

Table

Syntax	*object*.RepLogSettings [=value]
Parameters	value A Integer value representing the RepLogSettings.
Remarks	The RepLogSettings property has the **Integer** data type. This property is typically used to

Example	This example gets the value for the replication logging policies from the Logging configuration:
	
	`Dim repLogSettings`
	`repLogSettings =`
	`Transmitter.Configurations("Logging").RepLogSettings`

ReplyAddress property

Gets and sets the SMTP server reply address value in a BlackBerry configuration[1].

Table

Syntax	*object*.ReplyAddress [=value]
Parameters	value A String value representing the ReplyAddress.
Remarks	The ReplyAddress property has the **String** data type. This property is typically used to

Example	This example gets the value for the reply address from the BlackBerry configuration:
	
	`Dim replyAddress`
	`Set replyAddress =`
	`Transmitter.Configurations("BlackBerry").ReplyAddress`

1. In earlier Afaria releases, the BlackBerry Configuration object represented options and settings for configuration items relating to BlackBerry pagers, which included SMTP server configuration. The current release's BlackBerry support no longer requires the BlackBerry-specific items, but the product still supports using an SMTP server.

RunOnlyIfNewer property

Gets and sets whether a channel runs on a Client only when it has changed.

Some Content objects do not support this method. Use the AllowRunOnlyIfNewer property of the Content object to determine if this operation is allowed.

Table

Syntax	*object*.**RunOnlyIfNewer** [= *value*]
Parameters	value A Boolean that specifies whether the object runs only if changed. A value of True runs the object only if it changed.
Remarks	The RunOnlyIfNewer property has the **Boolean** type. This property allows you to minimize unnecessary downloads to the Client.
Example	This example sets a channel to run only if it has changed: `channel.RunOnlyIfNewer = True`

SendEncrypted property

Gets and sets whether a channel is sent to Clients using encryption.

The Microsoft CryptoAPI is used to establish a secure connection between the Server and Client. A Client that does not have CryptoAPI installed on their computer will not receive the channel contents.

Table

Syntax	*object*.**SendEncrypted** [= *value*]
Parameters	value A Boolean that specifies whether the object is sent using encryption. A value of True sends the object using encryption.
Remarks	The SendEncrypted property has the **Boolean** type.
Example	This example sets a channel to run using encryption: `channel.SendEncrypted = True`

SerialNumber property

Gets the serial number value in a certificate configuration.

Table

Syntax	*object*.**SerialNumber**
Parameters	N/A
Remarks	SerialNumber property has the **String** data type. This property is typically used to

Example	This example gets the value of the serial number in the first certificate configuration:
	
	`Dim certificateConfigs, serialNumber`
	`Set certificateConfigs =`
	`Transmitter.Configurations("Certificate")`
	`serialNumber = certificateConfigs.Item(1).SerialNumber`

ServerID property

Gets the Server ID for the current server. In a farm environment, the master server's FarmID property is the same as its ServerID property.

Table

Syntax	*object*.**ServerID**
Parameters	N/A
Remarks	The property has the **String** data type.
Example	

SessLogSettings property

Gets and sets the session logging policies in a logging configuration.

Table

Syntax	*object*.**SessLogSettings** [=*value*]
Parameters	value A Integer value representing the SessLogSettings.
Remarks	The SessLogSettings property has the **Integer** data type. This property is typically used to

Example	This example gets the value for the session logging policies from the Logging configuration:
	
	`Dim sessLogSettings`
	`sessLogSettings =`
	`Transmitter.Configurations("Logging").SessLogSettings`

SessionLimit property

Gets the session limit for a Server.

Table

Syntax	*object*.**SessionLimit**
Parameters	N/A
Remarks	Session Limit has the **Variant** type. It is Read Only.
Example	This example gets the Session Limit for a Server: `Msgbox.Transmitter.Licenses.SessionLimit`

SITSServerAddress property

Gets and sets the update server address for the antivirus and firewall component.

Table

Syntax	object.**SITSServerAddress** [=Value]
Parameters	Value A string value representing the address of the AV/Firewall update server.

Example	This example gets the value from the configuration object.
	
	`Dim updateServer`
	`Set updateServer =`
	`Transmitter.Configurations("AVFWSITS").SITSServerAddress`
	
	This example sets the AV/Firewall update server address.
	`avfw.SITSServerAddress = "avupdate.sybase.com"`

SMTPServer property

Gets and sets the SMTP server value in a BlackBerry configuration[2].

Table

Syntax	*object*.**SMTPServer** [=*value*]
Parameters	value A String value representing the SMTPServer.
Remarks	The SMTPServer property has the **String** data type. This property is typically used to

Example	This example gets the value for the SMTP server from the BlackBerry configuration:
	
	`Dim smtpServer`
	`Set smtpServer =`
	`Transmitter.Configurations("BlackBerry").SMTPServer`

2. In earlier Afaria releases, the BlackBerry Configuration object represented options and settings for configuration items relating to BlackBerry pagers, which included SMTP server configuration. The current release's BlackBerry support no longer requires the BlackBerry-specific items but does still support the SMTP server.

SMTPUserID property

Gets and sets the SMTP user ID value in a BlackBerry configuration[2].

Table

Syntax	*object*.**SMTPUserID** [=*value*]
Parameters	value A String value representing the SMTPUserID.
Remarks	The SMTPUserID property has the **String** data type. This property is typically used to

Example	This example gets the value for the SMTP user id from the BlackBerry configuration:
	
	`Dim smtpUserID`
	`Set smtpUserID =`
	`Transmitter.Configurations("BlackBerry").SMTPUserID`

SortMode property

Gets and sets the display sorting order.

Table

Syntax	*object*.**SortMode** [= *value*]
Parameters	value A Long that specifies the sort mode. This must be a value from the cmSortModeEnum enumeration.
Remarks	The SortMode property has the **cmSortModeEnum** type.
Example	Thisg example sorts a folder by name: `folder.SortMode = cmSortByName`

SourceFileName property

Gets the full path of the original file.

Table

Syntax	*object*.**SourceFileName**
Parameters	object An object expression that evaluates to an object that implements this property.
Remarks	The SourceFileName property has the **String** type. The following table summarizes the results of using the SourceFileName property with the objects listed:

Table

Object	Results
CachedFile	Gets the full path to the source file.
DifferenceFile	Gets the full path to the source file.
Document	Gets the full path to the document's source file.
FileVersion	Gets the full path to the source file.

Table

Example

This example displays the full path to the source file associated with the cached file:

```
MsgBox CachedFile.SourceFileName
```

SourceFileSize property

Gets the size (in bytes) of the original file.

Table

Syntax	*object*.**SourceFileSize**
Parameters	N/A

Table

Remarks	The SourceFileSize property has the **Double** type. The following table summarizes the results of using the SourceFileSize property with the objects listed:

Table

Object	Results
CachedFile	Gets the size of the source file associated with the cached file.
DifferenceFile	Gets the size of the source file associated with the difference file.
FileVersion	Gets the size of the source file associated with the difference file.

Table

Example	This example displays the source file sizes of the compressed file in the Server's CachedFiles collection:

```
For i = 1 to CachedFiles.Count
    MsgBox CachedFiles.Item(i).SourceFileSize
Next
```

SSLCert property

Gets an object that enables/disables SSL properties and displays certificate attributes.

Table

Syntax	*object*.**SSLCert**
Parameters	N/A
Remarks	The SSLCert property has the **SXSSSLCert object** type.
Example	This example gets the SSLCert property of a Transmitter object:

```
Dim SSLCertificate
Set SSLCertificate=Transmitter.SSLCert
```

SSLPort property

Gets and sets the SSL (Secure Sockets Layer) port number.

Table

Syntax	*object*.**SSLPort**
Parameters	value A Long that specifies the new SLL port number.
Remarks	The SSLPort property has the **Long** type.
Example	This example sets the SSL port number for a Server: `Transmitter.Configurations("Security").SSLPort = 29`

StartHour property

Gets and sets the hour of the day the trigger fires.

You must use the TriggerType property to set the trigger type before getting or setting any other trigger properties.

Table

Syntax	*object*.**StartHour** [= *value*]
Parameters	value A Short that specifies the hour of the day the trigger fires. This value is on a 24-hour clock; hours go from 0 to 23, inclusive.
Remarks	The StartHour property has the **Short** type.

Table

Example	This example creates a trigger that fires once every day, starting at 1:00 AM, on January 1, 2009:
	
	`Dim ta As afaria.TransmitterAccess`
	`Dim trans As afaria.Transmitter`
	`Dim trigger As afaria.trigger`
	`Set ta = CreateObject("Afaria.TransmitterAccess")`
	`Set trans = afaria.GetTransmitterFromAddress`
	`Set trigger = trans.GetTrigger`
	`trigger.TriggerType =`
	`Transmitter.Constant("cmTASK_TIME_TRIGGER_DAILY")`
	`trigger.BeginDate = CDate("January 1, 2009")`
	`trigger.StartHour = 1`

StartMinute property

Gets and sets the minute of the hour the trigger fires.

You must use the TriggerType property to set the trigger type before getting or setting any other trigger properties.

Table

Syntax	*object*.**StartMinute** [= *value*]
Parameters	value A Short that specifies the minute of the hour the trigger fires; minutes go from 0 to 59, inclusive.
Remarks	The StartMinute property has the **Short** type.

Table

Example	This example creates a trigger that fires once every day, starting at 1:32 AM, on January 1, 2009:

```
Dim ta As afaria.TransmitterAccess
Dim trans As afaria.Transmitter
Dim trigger As afaria.trigger
Set ta = CreateObject("Afaria.TransmitterAccess")
Set trans = afaria.GetTransmitterFromAddress
Set trigger = trans.GetTrigger
trigger.TriggerType =
Transmitter.Constant("cmTASK_TIME_TRIGGER_DAILY")
trigger.BeginDate = CDate("January 1, 2009")
trigger.StartHour = 1
trigger.StartMinute = 32
```

System property

Gets whether an object is a system object.

Table

Syntax	*object*.**System**
Parameters	N/A
Remarks	The System property has the **Boolean** type. System objects are automatically created and owned by the Server. These objects are typically not visible to the user and are not editable.

Table

Example	Example 1
	The following example enumerates all system channel items located in the root folder:
	
	`For Each channel In transmitter.ChannelItem`
	`  If True = channel.System Then`
	`  MsgBox "Channel Item " & channel.Name & " is a system object"`
	`  End If`
	`Next`
	
	Example 2
	Alternatively, you could explicitly enumerate system channel items as follows:
	
	`For Each channel In transmitter.ChannelItems( , cmFilterBySystem )`
	`  MsgBox "Channel Item " & channel.Name & " is a system object"`
	`Next`

TenantID property

Gets and sets the tenant ID to establish the tenant context for future operations. You can use this property with, or instead of, the TenantName property.

Table

Syntax	object.**TenantID** [=Value]
Parameters	Value An Integer value that specifies the ID for the tenant to use. Default value is 0, which represents the system tenant.
Example	This example the value for the tenant ID: `Transmitter.TenantID`

TenantName property

Gets and sets the tenant name to use as the context for future operations. You can use this property with, or instead of, the TenantID property.

Table

Syntax	object.**TenantName** [=Value]
Parameters	Value A BSTR value that specifies the name of the tenant to use. Value is case sensitive.
Example	This example sets the value for the tenant name to establish a tenant context for the subsequent task of creating and Inventory Manager Channel for the tenant:

```
Option explicit

Dim TransmitterAccess, Transmitter, ChannelItems, Flags,
ChannelItem

'Create the Afaria Object
set TransmitterAccess =
CreateObject("Afaria.TransmitterAccess")

'Set the Transmitter
set Transmitter = TransmitterAccess.GetTransmitterFromAddress

'Set the Tenant context of the transmitter
Transmitter.TenantName = "CJ"

'Gets the ChannelItems collection with all channels in the
collection
Flags = Transmitter.Constant("cmFilterByAll")
Set ChannelItems = Transmitter.ChannelItems( "", Flags )

'Adds a new Inventory Manager Channel named "Win32 HW Scan" to
the Sybase tenant
set ChannelItem = ChannelItems.Add("Win32 HW Scan", "Channel",
"\", "Inventory", "Win32")

'Sets the Inventory Manager Channel to the hardware only scan
option
ChannelItem.InventoryOptions = "HW"

'Sets the channel description

ChannelItem.Description = "A Hardware Scan Only Inventory
Manager Channel for Windows clients"
```

ThrottleDownPercentage property

Gets and sets the Throttle Down Percentage value in a Bandwidth Throttling configuration.

Table

Syntax	*object*.**ThrottleDownPercentage** [= *value*]
Parameters	value A Long value representing the ThrottleDownPercentage.
Remarks	The ThrottleDownPercentage property has the **Long** data type. This property is typically used to

Example	This example gets the value for the throttle down percentage from the first bandwidth throttling configuration:
	
	`Dim configs`
	`Dim throttleDown`
	`Set configs = Transmitter.Configurations("BandwidthThrottling")`
	`Set throttleDown = configs.Item(1).ThrottleDownPercentage`

ThrottleDownThreshold property

Gets and sets the throttle down threshold value in a Bandwidth Throttling configuration.

Table

Syntax	*object*.**ThrottleDownThreshold** [= *value*]
Parameters	value A Long value representing the ThrottleDownThreshold.
Remarks	The ThrottleDownThreshold property has the **Long** data type. This property is typically used to

Example	This example gets the value for the throttle down threshold from the first bandwidth throttling configuration:
	
	`Dim configs`
	`Dim throttleDown`
	`Set configs = Transmitter.Configurations("BandwidthThrottling")`
	`Set throttleDown = configs.Item(1).ThrottleDownThreshold`

ThrottleDownWaitTime property

Gets and sets the throttle down wait time value in a bandwidth throttling configuration.

Table

Syntax	*object*.**ThrottleDownWaitTime** [= *value*]
Parameters	value A Long value representing the ThrottleDownWaitTime.
Remarks	The ThrottleDownWaitTime property has the **Long** data type. This property is typically used to

Example	This example gets the value for the throttle down wait time from the first bandwidth throttling configuration:
	
	`Dim configs`
	`Dim throttleDown`
	`Set configs = Transmitter.Configurations("BandwidthThrottling")`
	`Set throttleDown = configs.Item(1).ThrottleDownWaitTime`

Transmitter property

Gets the Transmitter object.

Table

Syntax	*object*.**Transmitter**
Parameters	N/A
Remarks	The Transmitter property has the **Transmitter** type. The Transmitter object represents a Server on a particular machine. It provides access to other objects in the object model and provides methods and properties that affect the Server.

Example	This example gets the Transmitter object, using the Channel object:
	
	`Dim Trans`
	`Set Trans = Channel.Transmitter`

TransmitterDescriptions property

Gets the TransmitterDescriptions collection.

Table

Syntax	*object*.**TransmitterDescriptions**
Parameters	N/A
Remarks	The TransmitterDescriptions property has the **TransmitterDescriptions** type. The TransmitterDescriptions object is a collection object that represents descriptions of the Servers that can be accessed from the local workstation. Each item in this collection is represented by a TransmitterDescription object.

Example	This example displays the name of each Server that can be accessed from the local workstation:
	
	`Dim ta`
	`Set ta = CreateObject("Afaria.TransmitterAccess")`
	`Dim td`
	`For Each td In ta.TransmitterDescriptions`
	`  MsgBox td.Name`
	`Next`

TransmitterState property

Returns a value that specifies the state of the Server service.

Table

Syntax	*object*.**TransmitterState**
Parameters	N/A
Remarks	The TransmitterState property has the **cmTransmitterStateEnum** type.

Example	This example starts the Server if it is stopped:
	
	`Dim Trans`
	`Set Trans = CreateObject( "Afaria.Transmitter" )`
	`If Trans.TransmitterState = cmStateStopped Then`
	`  Trans.Start`
	`End If`

TriggerFlags property

Gets and sets the trigger's behavior.

You must use the TriggerType property to set the trigger type before getting or setting any other trigger properties.

Table

Syntax	*object*.**TriggerFlags** [= *value*]
Parameters	value A Long value that describes the trigger's behavior. This is a combination of one or more cmTriggerFlagsEnum values.
Remarks	The TriggerFlags property has the **Long** type that is a combination of one or more cmTriggerFlagsEnum values.
Example	This example creates an inactive trigger that fires once every day:

```
Dim ta As afaria.TransmitterAccess
Dim trans As afaria.Transmitter
Dim trigger As afaria.trigger
Set ta = CreateObject("Afaria.TransmitterAccess")
Set trans = afaria.GetTransmitterFromAddress
Set trigger = trans.GetTrigger
trigger.TriggerType =
Transmitter.Constant("cmTASK_TIME_TRIGGER_DAILY")
trigger.TriggerFlags = Transmitter.Constant("cmTASK_TRIGG
```

TriggerString property

Gets the trigger in the form of a string.

You must use the TriggerType property to set the trigger type before getting or setting any other trigger properties.

Table

Syntax	*object*.**TriggerString**
Parameters	N/A
Remarks	The TriggerString property has the **String** type.

Table

Example	This example creates a trigger that fires once every day, then displays a string version of the trigger:

```
Dim ta As afaria.TransmitterAccess
Dim trans As afaria.Transmitter
Dim trigger As afaria.trigger
Set ta = CreateObject("Afaria.TransmitterAccess")
Set trans = afaria.GetTransmitterFromAddress
Set trigger = trans.GetTrigger
trigger.TriggerType =
Transmitter.Constant("cmTASK_TIME_TRIGGER_DAILY")
Msgbox trigger.TriggerString
```

TriggerType property

Gets and sets the type of trigger.

You must use the TriggerType property to set the trigger type before getting or setting any other trigger properties.

Table

Syntax	*object*.**TriggerType** [=*Type*]
Parameters	Type An enumeration of type cmTriggerTypeEnum that specifies the trigger type.
Remarks	The TriggerType property is an enumeration of type **cmTriggerTypeEnum**.
Example	This example creates a trigger that fires once every day:

```
Dim ta As afaria.TransmitterAccess
Dim trans As afaria.Transmitter
Dim trigger As afaria.trigger
Set ta = CreateObject("Afaria.TransmitterAccess")
Set trans = afaria.GetTransmitterFromAddress
Set trigger = trans.GetTrigger
trigger.TriggerType =
Transmitter.Constant("cmTASK_TIME_TRIGGER_DAILY")
```

Type property

Gets the type of an object.

Table

Syntax	*object*.**Type**
Parameters	N/A
Return Values	The following table lists what the Type property returns for each object:

Table

Object	Type returned
ApplicationContent	Application
CacheConfiguration	Cache
CachedFile	A value from the cmCachedEntryEnum enumeration.
Channel	Channel
ChannelItem	Generic
ChannelSet	ChannelSet
CleanupConfiguration	Cleanup
Configuration	Generic
ContactConfiguration	Contact
Content	Generic
DifferenceConfiguration	Difference
DifferenceFile	A value from the cmCachedEntryEnum enumeration.
DocumentContent	Document
Folder	Folder
HTTPConfiguration	HTTP
InventoryContent	Inventory

Table

LicensedComponent	Clients Connections Session Manager Software Manager
SecurityConfiguration	Security
SendList	SendList
SessionContent	Session
SoftwareContent	Software
TransmitterConfiguration	Transmitter
TransmitterContent	Transmitter
WorkList	WorkList
WorkObject	Generic

Table

Remarks — The Type property is a **String** for all objects except as noted above. For example, the Type property of a ContactConfiguration object is the string "Contact." Used with the CachedFile object, the Type property is an enumeration of type cmCachedEntryEnum.

Example — This example gets the type of the currently opened Channel:

```
Dim channeltype
channeltype = Channel.Type
```

Unit property

Gets and sets the value of the server configuration's Unit value for client authentication. The unit value corresponds to a client SSL certificate's Organizational Unit value.

Table

Syntax	*object*.**Unit**
Parameter	N/A

Table

Remarks	**String** type. The Unit field for client authentication.
Example	

UseLDAP property

Gets whether LDAP (Lightweight Directory Access Protocol) is used for authentication.

Table

Syntax	*object*.**UseLDAP**
Parameters	N/A
Remarks	The UseLDAP property has the **Boolean** type.

Example	This example determines whether LDAP is in use for a Server:
	`If Transmitter.Configurations("Security").UseLDAP Then`
	`  Msgbox "LDAP is in use"`
	`End If`

UserAddress property

Gets the user address value in a certificate configuration.

Table

Syntax	*object.***UserAddress**
Parameters	N/A
Remarks	UserAddress property has the **String** data type. This property is typically used to

Table

Example	This example gets the value of the user address in the first certificate configuration:

```
Dim certificateConfigs, userAddress
Set certificateConfigs =
Transmitter.Configurations("Certificate")
userAddress = certificateConfigs.Item(1).UserAddress
```

UserAssignmentTimeout property

Gets and sets the amount of time that a client's cached assignment information remains valid.

Table

Syntax	*object.***UserAssignmentTimeout** [= *value*]
Parameters	value A Long that specifies the new timeout, in minutes.
Remarks	The UserAssignmentTimeout property has the **Long** type.
Example	This example sets the timeout for Client assignment information to one hour:

```
Transmitter.Configurations("Security").UserAssignmentTimeout = 60
```

UserAuthenticationRenew property

Gets and sets the amount of time that a Client's cached authentication information nearing its timeout can be automatically renewed.

Table

Syntax	*object.***UserAuthenticationRenew** [= *value*]
Parameters	value A Long that specifies the new timeout, in minutes.
Remarks	The UserAuthenticationRenew property has the **Long** type.

Table

Example	This example sets the timeout for automatically renewing Client authentication information to one hour: `Transmitter.Configurations("Security").UserAuthenticationRenew = 60`

UserAuthenticationTimeout property

Gets and sets the amount of time that a Client's cached authentication information remains valid.

Table

Syntax	*object*.**UserAuthenticationTimeout** [= *value*]
Parameters	value A Long that specifies the new timeout, in minutes.
Remarks	The UserAuthenticationTimeout property has the **Long** type.
Example	This example sets the timeout for Client authentication information to one hour: `Transmitter.Configurations("Security").UserAuthenticationTimeout = 60`

UserCommonName property

Gets the user common name value in a certificate configuration.

Table

Syntax	*object*.**UserCommonName**
Parameters	N/A
Remarks	UserCommonName property has the String data type. This property is typically used to

Table

Example	This example gets the value of the user common name in the first certificate configuration:
	
	`Dim certificateConfigs, userCommonName`
	`Set certificateConfigs =`
	`Transmitter.Configurations("Certificate")`
	`userCommonName = certificateConfigs.Item(1).UserCommonName`

UserCountry property

Gets the user country value in a certificate configuration.

Table

Syntax	*object*.**UserCountry**
Parameters	N/A
Remarks	UserCountry property has the **String** data type. This property is typically used to

Example	This example gets the value of the user country in the first certificate configuration:
	
	`Dim certificateConfigs, userCountry`
	`Set certificateConfigs =`
	`Transmitter.Configurations("Certificate")`
	`userCountry = certificateConfigs.Item(1).UserCountry`

UserLocality property

Gets the user locality value in a certificate configuration.

Table

Syntax	*object*.**UserLocality**
Parameters	N/A
Remarks	UserLocality property has the **String** data type. This property is typically used to

Table

Example	This example gets the value of the user locality in the first certificate configuration:
	`Dim certificateConfigs, userLocality`
	`Set certificateConfigs =`
	`Transmitter.Configurations("Certificate")`
	`userLocality = certificateConfigs.Item(1).UserLocality`

UserName property

Gets the name of the user associated with the object.

Table

Syntax	*object*.**UserName**
Parameters	N/A
Remarks	The UserName property has the **String** type. For FailedSession, gets the name of the user at the Client that executed the failed session.

Example	This example gets the name of the user associated with the first failed session:
	`Dim name`
	`name =`
	`Transmitter.Configurations("Cleanup").FailedSessions(1).UserName`

UserOrgName property

Gets the user organization name value in a certificate configuration.

Table

Syntax	*object*.**UserOrgName**
Parameters	N/A
Remarks	UserOrgName property has the **String** data type. This property is typically used to

Table

Example	This example gets the value of the user organization name in the first certificate configuration:
	
	`Dim certificateConfigs, userOrgName`
	`Set certificateConfigs =`
	`Transmitter.Configurations("Certificate")`
	`userOrgName = certificateConfigs.Item(1).UserOrgName`

UserState property

Gets the user state value in a certificate configuration.

Table

Syntax	*object*.**UserState**
Parameters	N/A
Remarks	UserState property has the **String** data type. This property is typically used to

Example	This example gets the value of the user state in the first certificate configuration:
	
	`Dim certificateConfigs, userState`
	`Set certificateConfigs =`
	`Transmitter.Configurations("Certificate")`
	`userState = certificateConfigs.Item(1).UserState`

UserUnit property

Gets the user unit value in a certificate configuration.

Table

Syntax	*object*.**UserUnit**
Parameters	N/A
Remarks	UserUnit property has the **String** data type. This property is typically used to

Table

Example	This example gets the value of the user unit in the first certificate configuration:
	
	`Dim certificateConfigs, userUnit`
	`Set certificateConfigs =`
	`Transmitter.Configurations("Certificate")`
	`userUnit = certificateConfigs.Item(1).UserUnit`

ValidDate property

Gets the valid date value in a certificate configuration.

Table

Syntax	*object*.**ValidDate**
Parameters	N/A
Remarks	ValidDate property has the **String** data type. This property is typically used to

Example	This example gets the value of the valid date in the first certificate configuration:
	
	`Dim certificateConfigs, validDate`
	`Set certificateConfigs =`
	`Transmitter.Configurations("Certificate")`
	`validDate = certificateConfigs.Item(1).ValidDate`

ValidDaysOfMonth property

Gets and sets the day of the month a trigger fires.

You must use the TriggerType property to set the trigger type before getting or setting any other trigger properties.

Table

Syntax	*object*.**ValidDaysOfMonth** [=*value*]

Table

Parameters	value A Long value that describes the days of the month the trigger fires. This is a combination of one or more cmDaysOfTheMonthEnum values.
Remarks	The ValidDaysOfMonth property has the **Long** type that is a combination of one or more cmDaysOfTheMonthEnum values.
Example	This example creates a trigger that fires every first and third day of the month:

```
Dim ta As afaria.TransmitterAccess
Dim trans As afaria.Transmitter
Dim trigger As afaria.trigger
Set ta = CreateObject("Afaria.TransmitterAccess")
Set trans = afaria.GetTransmitterFromAddress
Set trigger = trans.GetTrigger
trigger.TriggerType = cmTASK_TIME_TRIGGER_MONTHLYDATE
trigger.ValidDaysOfMonth = cmTASK_FIRST + cmTASK_THIRD
```

ValidDaysOfWeek property

Gets and sets the days of the week the trigger fires.

You must use the TriggerType property to set the trigger type before getting or setting any other trigger properties.

Table

Syntax	*object*.**ValidDaysOfWeek** [=*value*]
Parameters	value A Short value that describes the days of the week the trigger fires. This is a combination of one or more cmDaysOfTheWeekEnum values.
Remarks	The ValidDaysOfWeek property has the **Short** type that is a combination of one or more cmDaysOfTheWeekEnum values.

Table

Example — This example creates a trigger that fires every Sunday, Wednesday, and Friday:

```
Dim ta As afaria.TransmitterAccess
Dim trans As afaria.Transmitter
Dim trigger As afaria.trigger
Set ta = CreateObject("Afaria.TransmitterAccess")
Set trans = afaria.GetTransmitterFromAddress
Set trigger = trans.GetTrigger
trigger.TriggerType = cmTASK_TIME_TRIGGER_WEEKLY
trigger.ValidDaysOfWeek = cmTASK_SUNDAY + cmTASK_WEDNESDAY +
cmTASK_FRID
```

ValidMonths property

Gets and sets the month(s) when the trigger fires.

You must use the TriggerType property to set the trigger type before getting or setting any other trigger properties.

Table

Syntax — *object*.**ValidMonths** [=*value*]

Parameters — value
A Short value that describes the month(s) the trigger fires. This is a combination of one or more cmMonthsOfTheYearEnum values.

Remarks — The ValidMonths property has the **Short** type that is a combination of one or more cmMonthsOfTheYearEnum values.

Example — This example creates a trigger that fires on the months of January and March:

```
Dim ta As afaria.TransmitterAccess
Dim trans As afaria.Transmitter
Dim trigger As afaria.trigger
Set ta = CreateObject("Afaria.TransmitterAccess")
Set trans = afaria.GetTransmitterFromAddress
Set trigger = trans.GetTrigger
trigger.TriggerType = cmTASK_TIME_TRIGGER_MONTHLYDATE
trigger.ValidMonths = cmTASK_JANUARY + cmTASK_MARCH
```

Value property

Gets the value of an object.

Table

Syntax	*object*.**Value**
Parameters	N/A
Remarks	The Value property has the **String** type. For Licensed Component objects, the licensed value. If the license value is a number, it is interpreted as the upper limit for the license type, such as the number of concurrent sessions allowed; otherwise, the license value indicates the availability of an add-on product, such as "Limited" or "Licensed."
Example	This example gets the upper limit of the number of concurrent sessions allowed on a Server: `Transmitter.Licenses("Connections").Value`

Version property

Gets the version number of the Server.

Table

Syntax	*object*.**Version**
Parameters	N/A
Remarks	The Version property has the **String** type. The version information specifies the binary version number of the Server. The binary version number consists of two 32-bit integers represented as a dotted decimal String -- for example, "1.2.3.4" or "3.10".

Example	This example gets the version of the Server:
	
	`Dim Ver`
	`Ver = Transmitter.Version`

VisibilityWindowBegin property

Gets and sets the object's visibility window start time.

The order of operations is important. It is an error to set the visibility window time without first enabling it. Similary, it is an error to get the visibility window time if it is disabled. See the example below for how to correctly set and get the visibility window time.

Table

Syntax	*object*.**VisibilityWindowBegin** [= *value*]
Parameters	value A Date that specifies the object's visibility window start time.
Remarks	The VisibilityWindowBegin property has the **Date** type. You can make folders and published channels visible to Clients for only a set period of time. When the visibility window "closes" for a channel or folder, it disappears from the Channel Viewer but remains on the Server. To allow Clients to view the channel or folder again, simply reset the visibility window. Use the VisibilityWindowBegin and VisibilityWindowEnd properties to set and get the start and end times for the visibility window. Use the VisibilityWindowBeginEnabled and VisibilityWindowEndEnabled properties to enable and disable the visibility start and end times.

Example	Example 1
	The following example sets the channel's visibility window time:
	
	`channel.VisibilityWindowBeginEnabled = True`
	`channel.VisibilityWindowBegin = CDate("January 1, 2009")`
	`channel.VisibilityWindowEndEnabled = True`
	`channel.VisibilityWindowEnd = CDate("January 15, 2009")`
	
	Example 2
	The following example gets the channel's visibility window time:
	
	`If channel.VisibilityWindowBeginEnabled Then`
	`  Msgbox "Visibility Window starts on " &`
	`channel.VisibilityWindowBegin`
	`Else`
	`  Msgbox "Visibility Window start time is not enabled."`
	`End If`
	`If channel.VisibilityWindowEndEnabled Then`
	`  Msgbox "Visibility Window ends on " &`
	`channel.VisibilityWindowEnd`
	`Else`
	`  Msgbox "Visibility Window end time is not enabled."`
	`End If`

VisibilityWindowBeginEnabled property

Gets and sets whether the object's visibility window start time is enabled.

The order of operations is important. It is an error to set the visibility window time without first enabling it. Similary, it is an error to get the visibility window time if it is disabled. See the example below for how to correctly set and get the visibility window time.

Table

Syntax	*object*.**VisibilityWindowBeginEnabled** [= *value*]
Parameters	value A Boolean that specifies whether the object's visibility window start time is enabled.
Remarks	The VisibilityWindowBeginEnabled property has the **Boolean** type. You can make folders and published channels visible to Clients for only a set period of time. When the visibility window "closes" for a channel or folder, it disappears from the Channel Viewer but remains on the Server. To allow Clients to view the channel or folder again, simply reset the visibility window. Use the VisibilityWindowBegin and VisibilityWindowEnd properties to set and get the start and end times for the visibility window. Use the VisibilityWindowBeginEnabled and VisibilityWindowEndEnabled properties to enable and disable the visibility start and end times.

Example	Example 1
	This example sets the channel's visibility window time:
	
	`channel.VisibilityWindowBeginEnabled = True`
	`channel.VisibilityWindowBegin = CDate("January 1, 2009")`
	`channel.VisibilityWindowEndEnabled = True`
	`channel.VisibilityWindowEnd = CDate("January 15, 2009")`
	
	Example 2
	This example gets the channel's visibility window time:
	
	`If channel.VisibilityWindowBeginEnabled Then`
	`  Msgbox "Visibility Window starts on " &`
	`channel.VisibilityWindowBegin`
	`Else`
	`  Msgbox "Visibility Window start time is not enabled."`
	`End If`
	`If channel.VisibilityWindowEndEnabled Then`
	`  Msgbox "Visibility Window ends on " &`
	`channel.VisibilityWindowEnd`
	`Else`
	`  Msgbox "Visibility Window end time is not enabled."`
	`End If`

VisibilityWindowEnd property

Gets and sets the object's visibility window end time.

The order of operations is important. It is an error to set the visibility window time without first enabling it. Similary, it is an error to get the visibility window time if it is disabled. See the example below for how to correctly set and get the visibility window time.

Table

Syntax	*object*.**VisibilityWindowEnd** [= *value*]
Parameters	value A Date that specifies the object's visibility window end time.
Remarks	The VisibilityWindowEnd property has the **Date** type. You can make folders and published channels visible to Clients for only a set period of time. When the visibility window "closes" for a channel or folder, it disappears from the Channel Viewer but remains on the Server. To allow Clients to view the channel or folder again, simply reset the visibility window. Use the VisibilityWindowBegin and VisibilityWindowEnd properties to set and get the start and end times for the visibility window. Use the VisibilityWindowBeginEnabled and VisibilityWindowEndEnabled properties to enable and disable the visibility start and end times.
Example	Example 1 This example sets the channel's visibility window time:

```
channel.VisibilityWindowBeginEnabled = True
channel.VisibilityWindowBegin = CDate("January 1, 2009")
channel.VisibilityWindowEndEnabled = True
channel.VisibilityWindowEnd = CDate("January 15, 2009")
```

Example 2
This example gets the channel's visibility window time:

```
If channel.VisibilityWindowBeginEnabled Then
  Msgbox "Visibility Window starts on " &
channel.VisibilityWindowBegin
Else
  Msgbox "Visibility Window start time is not enabled."
End If
If channel.VisibilityWindowEndEnabled Then
  Msgbox "Visibility Window ends on " &
channel.VisibilityWindowEnd
Else
  Msgbox "Visibility Window end time is not enabled."
End If
```

VisibilityWindowEndEnabled property

Gets and sets whether the object's visibility window end time is enabled.

The order of operations is important. It is an error to set the visibility window time without first enabling it. Similary, it is an error to get the visibility window time if it is disabled. See the example below for how to correctly set and get the visibility window time.

Table

Syntax	*object*.**VisibilityWindowEndEnabled** [= *value*]
Parameters	value A Boolean that specifies whether the object's visibility window end time is enabled.
Remarks	The VisibilityWindowEndEnabled property has the **Boolean** type. You can make folders and published channels visible to Clients for only a set period of time. When the visibility window "closes" for a channel or folder, it disappears from the Channel Viewer but remains on the Server. To allow Clients to view the channel or folder again, simply reset the visibility window. Use the VisibilityWindowBegin and VisibilityWindowEnd properties to set and get the start and end times for the visibility window. Use the VisibilityWindowBeginEnabled and VisibilityWindowEndEnabled properties to enable and disable the visibility start and end times.

Example	Example 1
	This example sets the channel's visibility window time:
	
	`channel.VisibilityWindowBeginEnabled = True`
	`channel.VisibilityWindowBegin = CDate("January 1, 2009")`
	`channel.VisibilityWindowEndEnabled = True`
	`channel.VisibilityWindowEnd = CDate("January 15, 2009")`
	
	Example 2
	This example gets the channel's visibility window time:
	
	`If channel.VisibilityWindowBeginEnabled Then`
	`  Msgbox "Visibility Window starts on " &`
	`channel.VisibilityWindowBegin`
	`Else`
	`  Msgbox "Visibility Window start time is not enabled."`
	`End If`
	`If channel.VisibilityWindowEndEnabled Then`
	`  Msgbox "Visibility Window ends on " &`
	`channel.VisibilityWindowEnd`
	`Else`
	`  Msgbox "Visibility Window end time is not enabled."`
	`End If`

WeeklyInterval property

Gets and sets the interval between triggers, in weeks.

You must use the TriggerType property to set the trigger type before getting or setting any other trigger properties.

Table

Syntax	*object*.**WeeklyInterval** [=*value*]
Parameters	value A Short that specifies the interval between triggers, in weeks. 1 = every week, 2 = every other week, etc.
Remarks	The WeeklyInterval property has the **Short** type.

Example	This example creates a trigger that fires every other week, starting on January 1, 2009:
	
	`Dim ta As afaria.TransmitterAccess`
	`Dim trans As afaria.Transmitter`
	`Dim trigger As afaria.trigger`
	`Set ta = CreateObject("Afaria.TransmitterAccess")`
	`Set trans = afaria.GetTransmitterFromAddress`
	`Set trigger = trans.GetTrigger`
	`trigger.TriggerType =`
	`Transmitter.Constant("cmTASK_TIME_TRIGGER_WEEKLY")`
	`trigger.BeginDate = CDate("January 1, 2009")`
	`trigger.WeeklyInterval = 2`

WeekOfTheMonth property

Gets and sets the week of the month when the trigger fires.

You must use the TriggerType property to set the trigger type before getting or setting any other trigger properties.

Table

Syntax	*object*.**WeeklyInterval** [=*value*]
Parameters	value A Short value that describes the week of the month when the trigger fires. This is a single cmWeekOfTheMonthEnum value.

Table

Remarks	The WeekOfTheMonth property has the **Short** type that is a single cmWeekOfTheMonthEnum value.
Example	This example creates a trigger that fires on Monday and Friday of the third week of every third month:

```
Dim ta As afaria.TransmitterAccess
Dim trans As afaria.Transmitter
Dim trigger As afaria.trigger
Set ta = CreateObject("Afaria.TransmitterAccess")
Set trans = afaria.GetTransmitterFromAddress
Set trigger = trans.GetTrigger
trigger.TriggerType = cmTASK_TIME_TRIGGER_MONTHLYDOW
trigger.WeekOfTheMonth = cmTASK_THIRD_WEEK
trigger.DaysOfTheWeek = cmTASK_MONDAY + cmTASK_FRIDAY
trigger.ValidMonths = cmTASK_JANUARY + cmTASK_APRIL +
cmTASK_JULY + cmTASK_OCTOBER
```

WorkingCopyID property

Gets the identifier associated with an object's working copy.

If the channel is a working copy of another channel, the WorkingCopyID property has a value of zero (0); otherwise, it is the ID of the original channel.

Table

Syntax	*object*.**WorkingCopyID**
Parameters	N/A
Remarks	The WorkingCopyID property has the **Long** type. The WorkingCopyID property represents an identifier that uniquely distinguishes an object's working copy on a Server. This identifier is assigned by the system when the object is created, and is never reused.
Example	This example gets the working copy ID associated with the first channel item:

```
Dim channel
Set channel = Transmitter.ChannelItems(1)
If ( "Channel" = channel.Type ) Then
  MsgBox channel.WorkingCopyID
End If
```

WorkObjectName property

Gets the name of the work object associated with the channel.

Table

Syntax	*object*.**WorkObjectName**
Parameters	N/A
Remarks	The WorkObjectName property has the **String** type. An empty string is returned if no work object is associated with the channel.

Example	This example gets the name of the work object associated with the first channel item:
	
	`Dim channel`
	`Set channel = Transmitter.ChannelItems(1)`
	`If ( "Channel" = channel.Type ) Then`
	`  MsgBox channel.WorkObjectName`
	`End If`

6

Methods

In general, you use methods to get to functionality, which entails everything you can do to the content.

About methods

Methods are actions you take against objects. For example, to add a file to the Server's compressed file cache, use the Add method of the CachedFiles object.

For more background about methods, **see "Properties and Methods" on page 20.**

Add method (CachedFiles)

Adds a file to the CachedFiles collection.

This method fails if the Server service is not running.

Table

Syntax	*object.*Add Filename [, *Type* = cmCacheNormal]
Parameters	Filename A String that represents the fully qualified name of the file to add to the cache. Use a UNC name if the file is located on a remote machine. Type (Optional) An enumeration of type cmCachedEntryEnum that specifies when the file is automatically purged from the cache. The default type is cmCacheNormal.
Remarks	The Add method is an asynchronous operation. To get a CachedFile object representing the newly added file, call the Item method and specify the source file's fully qualified name.
Example	This example adds a file to the Server's compressed file cache: `Transmitter.Configurations("Cache").CachedFiles.Add "c:\demo\bigfile.dat", cmCacheSticky`

Add method (ChannelItems)

Adds a channel item to the ChannelItems collection.You can use this method to create a channel, channel set or folder item.

Table

Syntax	object.Add Name, Type, ParentFolder, ContentType, ClientType

Table

Parameters	Name A String that represents the name for the new channel item, as defined by the Name property. It is an error to specify a name of an existing channel, channel set, or folder item. Type A String that represents the channel item type, as defined by the Name property. The only valid channel item types are: “Channel”, “Channel Set”, and “Folder”. ParentFolder A String that represents the parent folder for the new channel item. It is an error to specify a destination folder that does not exist or does not represent an actual folder item. Use the string “\” or “” to specify the root folder. ContentType A String that represents the channel item's content type. The valid content item types are: "Configuration", "Inventory", and "Session". ("Application", "Document", "Software", and "Transmitter" are not valid.) This parameter is ignored for all channel item types except "Channel." ClientType A String that represents the channel item's client type. The valid client types are: "Win32", "WinCE", "Palm", "Java", "BlackBerry", "Symbian", and "Smartphone". Multiple client types may be specified by using a comma delimiter (example: “Win32, Smartphone, Symbian”). This parameter applies only for channel item type “Channel” and is ignored by all others. Return Values The Add method returns the ChannelItem object representing the new channel item.
Example	This example adds an Inventory Manager channel named “Inv Win32” to the channel tree’s root folder: `ChannelItems.Add (“Inv Win32”,”Channel”,”\”,”Inventory”,”Win32”)`

Add method (ChannelSetMembers)

Adds a channel to a ChannelSetMembers collection.

Table

Syntax	*object*.**Add** *Channel*

Table

Parameters	Channel A Variant that is a Long, String, or Channel object that represents a channel. • If you specify a *Long*, it must be the ID of the channel (as defined by the ID property). • If you specify a *String*, it must be the name of the channel (as defined by the FullName property). • If you specify a *Channel object*, it must represent a valid channel.
Return Values	The Add method returns the Channel object representing the channel added.

Example	This example adds a channel to a channel set:
	`Dim channel`
	`channelSet.ChannelSetMembers.Add  channel`

Add method (DifferenceFiles)

Adds a file to the DifferenceFiles collection.

This method fails if the Server service is not running.

Table

Syntax	*object*.**Add *Filename*** [, *Type* = cmCacheNormal]
Parameters	Filename A String that represents the fully qualified name of the file to add to the difference file cache. Use a UNC name if the file is located on a remote machine. Type (Optional) An enumeration of type cmCachedEntryEnum that specifies when the file is automatically purged from the cache. The default type is cmCacheNormal.
Return Values	The Add method is an asynchronous operation. To get a DifferenceFile object representing the newly added file, call the Item method and specify the source file's fully qualified name.
Example	The following example adds a file to the Server's difference file cache: `Transmitter.Configurations("Difference").DifferenceFiles.Add "c:\demo\bigfile.dat", cmCacheSticky`

AddCertificate method

Adds the named certificate.

Table

Syntax	object.**AddCertificate** Name
Parameters	Name The unqualified name of the certificate. CertificateFileName String of file name to add.
Remarks	Adds an existing certificate to the Server.
Example	The following example adds the certificate, “name”, to the collection: `Transmitter.Configurations("Certificate").AddCertificate “name”`

AddAssignment method

Adds a user group or Afaria client group to a profile’s assigment list.

Table

Syntax	object.**AddAssignment** [DomainName] [,GroupName] [,GroupType]
Parameters	DomainName A string that contains the domain name for the user group. Group Name A string that contains the Afaria client group name. Group Type A string that contains the Afaria client group type.
Example	This example adds client group “WindowsVista” to the current profile: `profile.AddAssignment (“WindowsVista”,”dynamic”)` See “Profile object” on page 82 for an extended example.

AddChannel method

Adds a channel or channel set to a profile's allowed channel list.

Table

Syntax	object.**AddChannel** channelItem
Parameters	channelItem A string that contains the fully qualified name of the channel or channel set to add.
Example	This example adds a channel to the current profile: `profile.AddChannel ("Inv Win32")` See "Profile object" on page 82 for an extended example.

AddMonitorAction method

Adds a monitor-action pair to a profile's client action list.

Table

Syntax	object.**AddMonitorAction** MonitorName, Action, ActionDefinitionXml, Enabled
Parameters	Some parameters for this method require detailed syntax and definitions. They provide the same capability as the product UI, which uses several UI controls to define the action. You are advised to use the UI to practice adding model monitor-action pairs and analyzing them as they exist in database table A_PRFL_TRIGGER_ACTION before attempting to add them programatically. MonitorName String containing the monitor name. Reference column TriggerID. Action String containing the action: Log, ExecuteProgram, ExecuteScript, or RunChannel. Reference column ActionTypeID. ActionDefinitionXml String containing the action qualifiers, according to action type. Reference column ActionDefinition. Enabled Boolean indicating whether to add the monitor-action pair in an enabled state.

Table

Example	This example adds a monitor with a run channel action. `prf.AddMonitorAction "Schedule Monitor", "RunChannel", "<Channel waitForCompletion=""False"" origChannelId=""105"" origTransmitterId=""ko$o"" /><Criteria><Connection type=""None"" /></Criteria><ErrorRetries count=""0"" intervalMinutes=""0"" />", False` See "Profile object" on page 82 for an extended example with additional action types.

AssociateCertificate method

Associates a certificate with a key file.

Table

Syntax	*object*.**AssociateCertificate**
Parameters	CertificateFile A string that represents the file path and name of the Server's certificate file. KeyFile A string that represents the file path and name of the corresponding private key file. AssociatedFile A string that represents the file name that will contain the Server's identity. The file name, path, and extension are pre-determined.
Remarks	Associates a certificate with a key file.

Example	This example associates a Certificate with a key file:
	
	`DimConfig`
	`SetConfig = Transmitter.Configurations("CertificateGeneration")`
	`Config.AssociateCertificate "C:\thwate.crt", "C:\1024RSAkey", "C:\ProgramFiles\Afaria\Data\Certs\Server.crt"`

ChangePassword method

Changes the password for the server's SSL certificate.

Table

Syntax	*object*.**ChangePassword**
Parameters	N/A
Example	

CheckVersion method

Determines if the given version number is compatible with that of the object model implementation.

Table

Syntax	object.**CheckVersion** Major, Minor
Parameters	Major A Short that represents the major version number. Minor A Short that represents the minor version number.
Return Values	The CheckVersion method returns one of the following values: • *True*. Indicates that the version numbers are compatible. • *False*. Indicates that the version numbers are incompatible.
Remarks	This method is primarily intended for typeless clients (VBScript, JavaScript, etc.) that need some way to perform version management. Although type libraries are tagged with a major and minor version number, most typeless clients do not have direct access to the object's type library or its version number. To allow typeless clients to check the version number, the Transmitter object supports the CheckVersion method.

Example	This example shows how to execute conditional code based on version information.
	
	`If Transmitter.CheckVersion( 1, 3 ) Then`
	`  'We are using a version 1.3 compatible implementation,`
	`  'so it is safe to invoke version 1.3 methods:`
	`  Transmitter.NewMethod  'invoke hypothetical version 1.3 method`
	`End If`

ClearAllFailedSessions method

Resets all the Server's failed session information.

Table

Syntax	*object*.**ClearAllFailedSessions**
Parameters	N/A
Remarks	A Server keeps track of failed sessions in order to perform failed session restarts. Use the ClearAllFailedSessions method to force all failed sessions to restart their communications from the beginning rather than from the point at which the session was interrupted. This is particularly useful when a channel continues to fail despite several attempts to correct it.

The channel will restart from the beginning the next time a Client connects to the Server. All currently running channels are not affected.

Table

Example	The following example clears all the Server's failed session information: `Transmitter.Configurations("Cleanup").ClearAllFailedSessions`

CopyToFolder method

Copies object to a folder.

Table

Syntax	object.**CopyToFolder** [Folder] [, Name] [, Recursive = False]

Table

Parameters	Folder (Optional) A Variant that is a Long, String, or Folder object that represents the destination folder for the new channel item. If this parameter is missing, the destination folder is the parent folder of the object. It is an error to specify a destination folder that does not exist or does not represent an actual folder item. If you specify a Long, it must be the ID of the destination folder (as defined by the ID property). If you specify a String, it must be the name of the destination folder (as defined by the FullName property). Use the string "\" to specify the root folder. If you specify a Folder object, it must represent a valid folder. Name (Optional) A String that represents the name for the new channel item (as defined by the Name property). If this parameter is missing or has a value of zero (0), a new name is automatically generated by the system. It is an error to specify a name of an existing channel item. Recursive (Optional) A Boolean specifying whether to copy all sub-items associated with the source object. This parameter is only valid for Folder objects.

Default is non-recursive copy. Recursive copy is not currently supported. It is an error to specify True for this parameter.

Table

Remarks	When you copy an object to a folder, the original object is unaffected. When specifying the destination folder, use the folder ID or Folder object instead of the FullName (the latter requires a lookup, which is slower).
Example	This example copies a channel to the "\Folder1\CriticalChannels" folder: `channel.CopyToFolder("\Folder1\CriticalChannels")`

CopyToFolderEx method

Copies object to a folder for a specific tenant.

Table

Syntax	object.**CopyToFolderEx** [Folder] [, Name] [, Recursive = False] [,TenantID]

Table

Parameters	Folder (Optional) A Variant that is a Long, String, or Folder object that represents the destination folder for the new channel item. If this parameter is missing, the destination folder is the parent folder of the object. It is an error to specify a destination folder that does not exist or does not represent an actual folder item. If you specify a Long, it must be the ID of the destination folder (as defined by the ID property). If you specify a String, it must be the name of the destination folder (as defined by the FullName property). Use the string "\" to specify the root folder. If you specify a Folder object, it must represent a valid folder. Name (Optional) A String that represents the name for the new channel item (as defined by the Name property). If this parameter is missing or has a value of zero (0), a new name is automatically generated by the system. It is an error to specify a name of an existing channel item. Recursive (Reserved for later use)(Optional) A Boolean specifying whether to copy all sub-items associated with the source object. This parameter is only valid for Folder objects. TenantID (Optional) An Integer value that specifies the ID for the tenant to use. Default value is 0, which represents the system tenant.

Default is non-recursive copy. Recursive copy is not currently supported. It is an error to specify True for this parameter.

Table

Remarks	When you copy an object to a folder, the original object is unaffected. When specifying the destination folder, use the folder ID or Folder object instead of the FullName (the latter requires a lookup, which is slower).
Example	This example copies a channel to the "\Folder1\CriticalChannels" folder in the system tenant: `channel.CopyToFolder("\Folder1\CriticalChannels",0)`

CreateFrom method

Creates a new BandwidthThrottlingConfiguration object from the data structure passed in.

Table

Syntax	object.**CreateFrom** OriginalName, CopyToName

Table

Parameters	OriginalName A String that represents the name of an existing BandwidthThrottlingConfiguration. CopyToName A String that represents the name for the new BandwidthThrottlingConfiguration.
Return values	The CreateFrom method returns a new BandwidthThrottlingConfiguration object.
Example	This example creates a new configuration object from an existing one: `Transmitter.Configurations("BandwidthThrottling").CreateFrom "OrgConfig", "NewConfig"`

Delete method

Deletes (permanently removes) the object.

Table

Syntax	object.**Delete**
Parameters	N/A
Remarks	Once an object is deleted, you should not call any of its methods or properties. Deleting an object effectively removes any underlying resources associated with the object.

You cannot delete a Channel object if it has a working copy.

Table

Example	This example deletes the first file from the compressed file cache: `Transmitter.Configurations("Cache").CachedFiles(1).Delete`

DeleteCertificate method

Deletes the named certificate.

Table

Syntax	object.**DeleteCertificate** Name
Parameters	Name The unqualified name of the certificate. CertificateFileName String of file name to delete.
Remarks	Once an object is deleted, you should not call any of its methods or properties. Deleting an object effectively removes any underlying resources associated with the object.
Example	The following example deletes the certificate, “name”, from the collection: `Transmitter.Configurations("Certificate").DeleteCertificate “name”`

EmptyCache method

Deletes all the files from the Server's compressed file cache or difference file cache.

Table

Syntax	object.**EmptyCache**
Parameters	N/A
Remarks	The file compression cache is used to store compressed files that are frequently sent to clients. The file difference cache is used to store different versions of files that are frequently sent to Clients.
Example	This example deletes all the files from the Server’s compressed file cache: `Transmitter.Configurations("Cache").EmptyCache`

Folder method

Returns the folder containing the ChannelItems collection.

Table

Syntax	object.**Folder**

Table

Parameters	N/A
Remarks	
Example	This example returns the pointer to the folder for the current ChannelItems collection: `ChannelItems.Folder`

GenerateCertificateEx method

Generates a new certificate request.

Table

Syntax	*object*.**GenerateCertificateEx** *CryptionType, CryptionLength, CommonName, Organization, OrganizationUnit, StreetAddress, Locality, State, Country, RequestFile, PrivateKeyFile, Password*

Table

Parameters	CryptionType An integer representing the type of certificate key to generate: RSA key pair is the only supported type. CryptionLength An integer representing the length in the range of 512 to 2048 bits. CommonName A string representing DNS address for an Internet server. Organization A string representing ISO registered top-level company name. OrganizationUnit A string representing a department within a company. StreetAddress A string representing the street address for a company. Locality A string representing the city where a company is located. State A string representing the full name of the state, province, or territory where a company is located. Country A string representing the two-letter ISO code for the nation where a company is located. RequestFile A string representing the file path and name where the generated certificate request is to be stored. PrivateKeyFile A string representing the file path and name where the generated private key is to be stored. Password A string representing the password to associate with the certificate.
Remarks	Generates a new certficate.

Example	This example creates a Certificate request:
	
	`DimConfig`
	`SetConfig = Transmitter.Configurations("CertificateGeneration")`
	`Config.GenerateCertificate 22, 1024, "63.127.43.21",`
	`"MyCompany", "Sales", "2432 Ricker Road", "Alpharetta", "GA",`
	`"US", "C:\CertReq.crq", "C:\1024RSA.key", "pass1234"`

GetCertificates method

Gets the certificate for the Server.

Table

Syntax	*object*.**GetCertificates**
Parameters	N/A
Remarks	Gets the certificate for a Server
Example	This example gets the Certificate for a Server: `Transmitter.SSLCert.GetCertificates`

GetItemByID method

Gets a specified object from a collection by its ID.

Table

Syntax	object.**GetItemByID** ID
Parameters	ID A Long representing the object's ID (as defined by its ID property).
Return Values	The GetItemByID method returns the object.
Remarks	Use the GetItemByID method when you know nothing about an object except its ID.

Example	This example gets the object representing the channel item with an ID of 12 located in the root folder:
	
	`Dim item`
	`Set item = transmitter.ChannelItems.GetItemByID( 12 )`

GetTransmitterFromAddress method

Gets a Transmitter object that represents a Server located at the specified address.

Table

Syntax	object.**GetTransmitterFromAddress** [, Address = "<default>"]

Table

Parameters	Address (Optional) A String that represents the Transmitter address. This can be a UNC machine name or a network IP address.
Remarks	If you do not specify an address, or you use the string "<default>", the returned Transmitter object represents the default Server for your workstation.

The Transmitter object uses a significant amount of computer resources. In addition, it may take several seconds to create this object. Automation Clients are advised to use a single Transmitter object that is referenced for the life of the application. The only time you should create multiple Transmitter objects is if you need to simultaneously control multiple Servers.

Table

Example	This example stops the Server service for the default (local) Server.
	
	`Dim ta As Afaria.TransmitterAccess`
	`Dim t As Afaria.Transmitter`
	`Set ta = CreateObject( "Afaria.TransmitterAccess" )`
	`Set t = ta.GetTransmitterFromAddress`
	`t.Stop`

GetTransmitterFromAddress2 method

Gets a Server object that represents a Server located at the specified address.

The Transmitter object uses a significant amount of computer resources. In addition, it may take several seconds to create this object. Automation Clients are advised to use a single Transmitter object that is referenced for the life of the application. The only time you should create multiple Transmitter objects is if you need to simultaneously control multiple Servers.

Table

Syntax	*object*.**GetTransmitterFromAddress2** [, *Address* = "<default>"]
Parameters	Address (Optional) A String that represents the Server address. This can be a UNC machine name or a network IP address. UserContext A string of the user contenxt to use to gain permission to get a Server. Transmitter Returned Transmitter object.
Remarks	If you do not specify an address, or you use the string "<default>", the returned Transmitter object represents the default Server for your workstation.

Table

Example	This example stops the Server service for the default (local) Server:
	
	`Dim ta As Afaria.TransmitterAccess`
	`Dim t As Afaria.Transmitter`
	`Set ta = CreateObject( "Afaria.TransmitterAccess" )`
	`Set t = ta.GetTransmitterFromAddress2`
	`t.Stop`

InitInstance method

Initializes a CertificateConfiguration object from the name of the certificate file.

Table

Syntax	*object*.**InitInstance** CertificateFileName
Parameters	CertificateFileName String of file containing certificate information
Remarks	CertificateFileName: String of file name containing certificate information.

Example	This example initializes a CertificateConfiguration object from the name of the Certificate file:
	
	`Dim certConfigs`
	`Set certConfigs =`
	`Transmitter.Configurations("CertificateConfigurations")`
	`certConfigs.Item(1).InitInstance <full path of filename containing certificate information>`

IsClientTypeLicensed method

Gets if a Client type is licensed.

Table

Syntax	*object*.**IsClientTypeLicensed**

Table

Parameters	Component Long that represents the Client type to check: LIC45_CLIENT_WIN32 = 0; LIC45_CLIENT_PALM = 1; LIC45_CLIENT_WINCE = 2; LIC45_CLIENT_BLACKBERRY = 3; LIC45_CLIENT_JAVA = 4; LIC45_CLIENT_SYMBIAN = 5;
Remarks	The IsClientTypeLicensed function returns a Long.
Example	This example displays how to determine if a Client type is licensed: `Msgbox Transmitter.Licenses.IsClientTypeLicensed dwComponent`

IsProductLicensedForAnyClientType method

Gets if a product is licensed for any Client type.

Table

Syntax	*object.***IsProductLicensedForAnyClientType**
Parameters	Component Long that represents the product type to check: LIC45_CLIENT_WIN32 = 0; LIC45_CLIENT_PALM = 1; LIC45_CLIENT_WINCE = 2; LIC45_CLIENT_BLACKBERRY = 3; LIC45_CLIENT_JAVA = 4; LIC45_CLIENT_SYMBIAN = 5;
Remarks	The IsProductLicensedForAnyClientType function returns a Long.
Example	This example displays how to determine if a product is licensed for any Client type: `Msgbox Transmitter.Licenses.IsProductLicensedForAnyClientType dwComponent`

Item method

Gets a specified object from a collection.

Table

Syntax	*object*.**Item** *Index* or object.Index
Parameters	Index A Variant that is a Long or String representing the appropriate object. • If you specify a Long, the Item method fetches the object by its one-based index in the collection. • If you specify a String, it must be one of the strings described in the following table:

Table

Object	String
BandwidthThrottlingConfiguration	The BandwidthThrottlingConfiguration object name as described by the Description property.
CachedFile	The compressed file's fully qualified name (as defined by the SourceFileName property).
CertificateConfiguration	
ChannelItem ChannelSetMember	The channel item's name. Use the item's fully qualified name (as defined by the FullName property) to reference a specific item; otherwise, use the item's name (as defined by the Name property) to reference the first item with that name, regardless of its location.
Configuration	The configuration's name (as defined by the Type property).
DependentDocument	The document name (as defined by the Name or SourceFileName property).
DifferenceFile	The difference file's fully qualified name (as defined by the SourceFileName property).
Document	The document name (as defined by the Name or SourceFileName property).
FailedSession	The failed session's name (as defined by the ChannelName property.
FileVersion	Not supported--you must use a Long to specify the index.
LicensedComponent	The license's type (as defined by the Type property).

Table

TransmitterDescription	Not supported--you must use a Long to specify the index.
WorkObjectEvent	Not supported.
WorkObject	The work object's name (as defined by the Name property).

Table

Remarks

If you specify numbers for index, do not store these for later use because the indices might change as objects are added or removed.
The Item method is the default. Accordingly, you don't have to reference Item explicitly, as shown in the syntax.

Example

This example displays the types of all Server configurations:

```
Sub DisplayTransmitterConfigurations( )
  Dim i
  for i = 1 To Transmitter.Configurations.Count
    MsgBox Transmitter.Configurations.Item(i).Type
  next
End Sub
```

MoveToFolder method

Moves object to a folder.

Table

Syntax

object.**MoveToFolder** Folder

Parameters

Folder
A Variant that is a Long, String, or Folder object that represents the destination folder.It is an error to specify a destination folder that does not exist or does not represent an actual folder item.

- If you specify a Long, it must be the ID of the destination folder (as defined by the ID property).
- If you specify a String, it must be the name of the destination folder (as defined by the FullName property). Use the string "\" to specify the root folder.
- If you specify a Folder object, it must represent a valid folder.

Remarks

When specifying the destination folder, use the folder ID or Folder object instead of the FullName (the latter requires a lookup, which is slower).

Example

This example moves a channel to the root folder:

```
channel.MoveToFolder("\")
```

RefreshCache method

Refreshes the files from the Server's compression cache.

Table

Syntax	*object*.**RefreshCache**
Parameters	N/A
Remarks	The RefreshCache method should be called when the underlying content has changed (for example, when the data in a file changes). Refreshing the cache ensures that the Server updates the cache and efficiently serves Client requests.

Example	This example refreshes the Server's compression cache:
	
	`Dim config`
	`Set config = Transmitter.Configurations("Cache")`
	`Msgbox config.RefreshCache`

RefreshContent method

Refreshes a channel's content.

Table

Syntax	*object*.**RefreshContent**
Parameters	N/A
Remarks	The RefreshContent method should be called when the underlying content has changed (for example, when the data in a file changes). Refreshing the content ensures that clients receive up-to-date information.

Some Content objects do not support this method. Use the AllowRefresh property of the Content object to determine if this operation is allowed. This property is usually a read-only property of the content, but some Content objects allow you to change the value of this property (e.g., DocumentContent objects).

Table

Example	This example refreshes the content associated with a channel named "Price List." The channel is located in the root folder: `transmitter.ChannelItems("Price List").RefreshContent`

Remove method

Removes the item from a collection.

Table

Syntax	*object*.**Remove** *Index*
Parameters	Index A Variant that is a Long or String representing the appropriate object. • If you specify a Long, the Item method fetches the object by its one-based index in the collection. Only the String variant is applicable to a BandwidthThrottlingConfiguration. • If you specify a String, it must be one of the strings described in the following table:

Table

Object	String
BandwidthThrottlingConfiguration	The BandwidthThrottlingConfiguration object name as described by the Description property.
CachedFile	The fully qualified name of the source file associated with the compressed file (as defined by the SourceFileName property).
ChannelItem	The channel name. Use the fully qualified name (as defined by the FullName property) or the item's name (as defined by the Name property) to reference the item in the current folder. **Note**: You cannot delete a Channel object if it has a working copy.
ChannelSetMember	The channel name. Use the fully qualified name (as defined by the FullName property) to reference a specific item; otherwise, use the item's name (as defined by the Name property) to reference the first item with that name, regardless of its location. **Note**: Removing a channel from a channel set does not delete the channel from the system.
DependentDocument	The document name (as defined by the Name or SourceFileName property). **Note**: Adding and removing a dependent document does not remove the document from the channel--it only removes it from the dependency list.
DifferenceFile	The fully qualified name of the source file associated with the difference file (as defined by the SourceFileName property).
Documents	The document name (as defined by the Name or SourceFileName property).

Table

FailedSession	The name of the channel (as defined by the ChannelName property).
FileVersion	Not supported--you must use a Long to specify the index.
LicensedComponent	The license's type (as defined by the Type property).
TransmitterDescription	Not supported--you must use a Long to specify the index.
WorkObjectEvent	Not supported.
WorkObject	The work object's name (as defined by the Name property). **Note**: Removing a work object from a Session channel does not delete the work object from the system--it only unassigns the work object from the channel. All assignment changes take effect immediately, regardless of the current edit mode

If you specify an Object, it must be a valid object in the collection as described in the following table:

Collection	Object
ChannelSetMembers	Channel **Note**: Removing a channel from a channel set does not delete the channel from the system.

Table

Remarks

If you specify numbers for index, do not store these for later use because the indices might change as objects are added or removed.
Unless otherwise specified, removing an item from a collection also destroys the item (most items cannot exist outside their collection). For example, removing a CachedFile object from the CachedFiles collection also removes the cached file from the Server's file compression cache. In addition, you cannot remove or delete system objects (as defined by the System property).

Example

This example removes the first file from the compressed file cache:

```
Transmitter.Configurations("Cache").CachedFiles.Remove(1)
```

The Count property of a collection changes after the Remove method is called.

RemoveAll method

Removes all items from a collection.

Table

Syntax	*object.***RemoveAll**
Parameters	N/A
Remarks	Unless otherwise specified, removing an item from a collection also destroys the item (most items cannot exist outside their collection). For example, removing a CachedFile object removes the associated file from the Server's file compression cache. See the Remove method for more information.
Example	This example removes all the files from the compressed file cache: `Transmitter.Configurations("Cache").CachedFiles.RemoveAll`

The Count property of a collection changes after the RemoveAll method is called.

RemoveAllAssignments method

Removes all user groups and Afaria client groups from a profile's assigment list.

Table

Syntax	object.**RemoveAllAssignments**
Parameters	N/A
Example	This example removes all assignments from the current profile: `profile.RemoveAllAssignments ()` See "Profile object" on page 82 for an extended example.

RemoveAllChannels method

Removes all explicitly allowed channels from the profile. See “RemoveChannel method” on page 266 to learn about explicit and implicit allowed channels.

Table

Syntax	object.**RemoveAllChannels**
Parameters	N/A
Example	This example removes all channels from the current profile: `profile.RemoveAllChannels ()` See “Profile object” on page 82 for an extended example.

RemoveAssignment method

Removes a user group or Afaria client group from a profile’s assigment list.

Table

Syntax	object.**RemoveAssignment** [DomainName] [,GroupName] [,GroupType]
Parameters	DomainName A string that contains the domain name for the user group. Group Name A string that contains the Afaria client group name. Group Type A string that contains the Afaria client group type. • System • Client • Local • Domain • LDAPOU • LDAPOBJ
Example	This example removes local client group “Administrators” from current profile: `profile.RemoveAssignment "", "Administrators" , "Local"` See “Profile object” on page 82 for an extended example.

RemoveChannel method

Removes an explicitly allowed channel or channel set from profile's allowed channels list.

An explicitly allowed channel is one that is manually added to a profile's allowed channels list, which is distinct from a implicitly allowed channel. An implicitly allowed channel is one that is present on the allowed channels list by virtue of its status as the profile's default channel or as a member of a monitor-action pair on the client actions list.

A channel that is on an allowed channels list may have both explicit and implicit membership. Removing such a channel using the RemoveChannel method removes only the explicit membership.

Table

Syntax	object.**RemoveChannel** channelItem
Parameters	channelItem A string that contains the fully qualified name of the channel or channel set to remove.
Example	This example removes a channel from the current profile: `profile.RemoveChannel("\CJAFARIA\Config\BB Config")` See "Profile object" on page 82 for an extended example.

RemoveChannelByID method

Removes an explicitly allowed channel or channel set from profile's allowed channels list. See "RemoveChannel method" on page 266 to learn about explicit and implicit allowed channels.

Table

Syntax	object.**RemoveChannelByID** originalTransmitterID, originalChannelID
Parameters	originalTransmitterID A string that contains the ID of the server on which the channel was originally created. originalChannelID An integer that contains the ID of the channel from the server on which the channel was originally created.

Example	This example a channel using a variable as the parameter value.
	
	`chanID = chanItem.ID`
	`profile.RemoveChannelByID (chanID)`
	
	See "Profile object" on page 82 for an extended example.

RemoveMonitorAction method

Removes a monitor-action pair from a profile's client action list.

Table

Syntax	object.**RemoveMonitorAction** MonitorName, Action, ActionDefinitionXml
Parameters	Some parameters for this method require detailed syntax and definitions. You are advised to analyze sample monitor-action pairs as they exist in database table A_PRFL_TRIGGER_ACTION before attempting to remove them programatically. MonitorName String containing the monitor name. Reference column TriggerID. Action String containing the action: Log, ExecuteProgram, ExecuteScript, or RunChannel. Reference column ActionTypeID. ActionDefinitionXml String containing the action qualifiers, according to action type. Reference column ActionDefinition.
Example	This example removes a monitor-action pair from the current profile. `prf.RemoveMonitorAction "Directory Monitor", "ExecuteProgram", "<Program waitForCompletion=""False"" name=""Notepad"" parameters="""" /><Criteria><Connection type=""None"" /></Criteria><ErrorRetries count=""0"" intervalMinutes=""0"" />"` See "Profile object" on page 82 for an extended example.

ResetAddress method

Resets an address to its default value.

Table

Syntax	*object*.**ResetAddress**
Parameters	N/A
Remarks	Resets the network IP address used by Clients to connect to the Server. The address can be a machine name, such as companyname.com, or a numeric IP address, such as 128.56.22.8.
Example	This example resets the Server's address to its default value: `Transmitter.Configurations("Transmitter").ResetAddress`

ResetAll method

Resets options and settings to their default values.

You must stop and restart the Server to use any new settings.

Table

Syntax	*object*.**ResetAll**
Parameters	N/A
Remarks	Resets *a subset* of the Server settings and options to their default values.

Table

Example	This example resets all the Server's cleanup settings to their default values: `Transmitter.Configurations("Cleanup").ResetAll`

ResetChannelUpdateSchedule method

Resets the schedule for automatically updating channel contents.

You must stop and restart the Server to use the new setting.

Table

Syntax	*object*.**ResetChannelUpdateSchedule**
Parameters	N/A
Remarks	The channel update schedule specifies how often the Server should refresh channel content. The Server refreshes channel content to ensure that the most current version of data is available to clients. The default value is once every day.
Example	This example resets the Server's channel update schedule to its default value: `Transmitter.Configurations("Cleanup").ResetChannelUpdateSchedule`

See also **ChannelUpdateSchedule property on page 135**.

ResetDeletedChannelCleanupSchedule method

Resets the schedule for automatically cleaning up deleted channels.

You must stop and restart the Server to use the new setting.

Table

Syntax	*object*.**ResetDeletedChannelCleanupSchedule**
Parameters	N/A
Remarks	The deleted channel update schedule specifies how often the Server should cleanup deleted channels. The default value is once every day.
Example	This example resets the Server's deleted channel cleanup schedule to its default value: `Transmitter.Configurations("Cleanup").ResetDeletedChannelCleanupSchedule`

ResetFailedSessionCleanupSchedule method

Resets the schedule for automatically cleaning up failed sessions.

You must stop and restart the Server to use the new setting.

Table

Syntax	*object*.**ResetFailedSessionCleanupSchedule**
Parameters	N/A
Remarks	The failed session cleanup schedule specifies how often the Server should cleanup failed sessions. The default value is once every day.
Example	This example resets the Server's failed session cleanup schedule to its default value: `Transmitter.Configurations("Cleanup").ResetFailedSessionCleanupSchedule`

ResetName method

Resets a name to its default value.

Table

Syntax	*object*.**ResetName**
Parameters	N/A
Remarks	Resets the Server's name.
Example	This example resets the Server's address to its default value: `Transmitter.Configurations("Transmitter").ResetAddress`

ResetPort method

Resets a port to its default value.

You must stop and restart the Server to use any new settings.

Table

Syntax	*object*.**ResetPort**
Parameters	N/A
Remarks	The following table summarizes the results of using the ResetPort method with the objects listed: *Table* Object — Results HTTPConfiguration — Resets the Server's HTTP port to its default value. TransmitterConfiguration — Resets the Server's TCP port to its default value. *Table*
Example	This example resets the Server's HTTP port to its default value: `Transmitter.Configurations("HTTP").ResetPort`

SetDailyTrigger method

Sets a trigger to fire at 12:00 AM every N day(s), starting today.

Table

Syntax	*object*.**SetDailyTrigger** *Interval*
Parameters	Interval A Short that represents the interval, in days. Valid values are 1 to 32767, inclusive.
Remarks	This method provides an easy way to create a simple daily trigger.

Example	This example creates a trigger that fires once every three days:
	
	`Dim ta As afaria.TransmitterAccess`
	`Dim trans As afaria.Transmitter`
	`Dim trigger As afaria.trigger`
	`Set ta = CreateObject("Afaria.TransmitterAccess")`
	`Set trans = afaria.GetTransmitterFromAddress`
	`Set trigger = trans.GetTrigger`
	`trigger.SetDailyTrigger = 3`

SetDefaultHTTPSPort method

Sets the default HTTPS port for a Server.

Table

Syntax	*object*.**DefaultHTTPSPort**
Parameters	N/A
Remarks	Sets the default HTTPS port for a Server
Example	This example sets the default HTTPS port for a Server: `Transmitter.SSLCert.DefaultHTTPSPort`

SetDefaultSSLPort method

Sets the default SSL port for a Server.

Table

Syntax	*object*.**DefaultSSLPort**

Table

Parameters	N/A
Remarks	Sets the default SSL port for a Server
Example	This example sets the default SSL port for a Server:

```
Transmitter.SSLCert.DefaultSSLPort
```

SetHourlyTrigger method

Sets a trigger to fire at 12:00 AM every N hour(s), starting today.

Table

Syntax	*object*.**SetHourlyTrigger** *Interval*
Parameters	Interval A Short that represents the interval, in hours. Valid values are 1 to 23, inclusive.
Remarks	This method provides an easy way to create a simple hourly trigger.
Example	This example creates a trigger that fires once every three hours:

```
Dim ta As afaria.TransmitterAccess
Dim trans As afaria.Transmitter
Dim trigger As afaria.trigger
Set ta = CreateObject("Afaria.TransmitterAccess")
Set trans = afaria.GetTransmitterFromAddress
Set trigger = trans.GetTrigger
trigger.SetHourlyTrigger = 3
```

SetPassword method

Associates a password with the server's SSL certificate.

Table

Syntax	*object*.**SetPassword**
Parameters	N/A
Example	

SetPublish method

Sets whether an object is published.

Publishing a working copy of a channel will overwrite and publish the original copy, even if the original is unpublished.

Table

Syntax	*object*.**SetPublish** *Published*
Parameters	Published A Boolean that specifies whether the object is published. A value of *True* publishes the object. A value of *False* unpublishes the object.
Remarks	You must publish a channel to make it available for Clients.
Example	This example publishes a channel: `channel.SetPublish( True )`

SetWorklistFile method

Creates a worklist and assigns it to a Session Manager channel, where “Session” refers to the ContentType property type.

Table

Syntax	object.**SetWorklistFile** WorklistFile, WorklistName
Parameters	WorklistFile A String that represents the name of an existing file that contains a worklist. It is an error to specify a worklist file that does not exist or does not represent an actual worklist file. WorklistName A String that represents the name of the worklist to be assigned to this Session Manager channel. It is an error to specify a worklist that exists unless *bOverWrite* is True.
Example	This example creates a worklist for the current channel: `channel.SetWorklistFile (“c:\worklist.evf”,”Worklist”)`

SetWorklist method

Assigns an existing worklist to a Session Manager channel, where “Session” refers to the ContentType property type.

Table

Syntax	object.**SetWorklist** WorklistName
Parameters	WorklistName A String that represents the name of the worklist to assign to the Session Manager channel. It is an error to specify a worklist that does not exist or does not represent an actual worklist item.
Example	This example assigns a worklist to the current channel: `channel.SetWorklist (“Worklist”)`

Start method

Starts the Server service.

Table

Syntax	*object*.**Start**
Parameters	N/A
Remarks	If the Server is already started when this method is called, the method does nothing and succeeds.

Example	This example starts the default (local) Server service:
	
	`Dim ta As Afaria.TransmitterAccess`
	`Dim t As Afaria.Transmitter`
	`Set ta = CreateObject( "Afaria.TransmitterAccess" )`
	`Set t = ta.GetTransmitterFromAddress`
	`t.Start`

Stop method

Stops the Server service.

Table

Syntax	*object*.**Stop**

Table

Parameters	N/A
Remarks	If the Server is already stopped when this method is called, the method does nothing and succeeds.

Example	This example stops the default (local) Server service:
	
	`Dim ta As Afaria.TransmitterAccess`
	`Dim t As Afaria.Transmitter`
	`Set ta = CreateObject( "Afaria.TransmitterAccess" )`
	`Set t = ta.GetTransmitterFromAddress`
	`t.Stop`

7

Events

An event is an action or occurrence, often generated by the user, to which a VBScript event handler or other Automation controller can respond.

About events

An event is an action or occurrence, often generated by the user, to which a VBScript event handler or other Automation controller can respond. For example, if when the Server service starts or stops, the Server fires the OnTransmitterStateChanged event of the Transmitter object.

In a VBScript event handler, you can respond to an event by taking some action. For instance, you could restart the Server service by calling the Start method of the Transmitter object.

To respond to an event, add the appropriate event handler to your Automation controller or VBScript macro. You can add this event handler to multiple VBScript macro files. For example, if you add an event handler for the OnTransmitterStateChanged event to two loaded VBScript macro files, when the Server service begins its shutdown sequence, both of these event handlers will be called¯one at a time.

To add a VBScript event handler, find the appropriate event by using the previous table, copy the sample event handler from the example, paste the handler into your script, insert the appropriate code into the handler, and then reload the script.

For example, you could take some action just before the Server service stops by using the OnTransmitterStateChanged event. To use this event, go to the description of the event and copy the sample event handler:

```
Dim Trans
Sub Trans_OnTransmitterStateChanged( oldState, newState, theTransmitter )
 ' Add code here to handle state change
End Sub
```

Then, paste this code into your macro file, insert the code for handling the event, and then reload the macro file.

OnTransmitterStateChanged event

Occurs when the Server service changes state..

Table

Syntax	*object*_**OnTransmitterStateChanged** *oldState, newState, theTransmitter*
Parameters	object An object expression that evaluates to an object that implements this property. oldState A Long that specifies the previous state of the Server service. This is a value from the cmTransmitterStateEnum enumeration. newState A Long that specifies the current state of the Server service. This is a value from the cmTransmitterStateEnum enumeration. theTransmitter The Transmitter object that fires this event. Non-scripting clients must explicitly release this object; scripting Clients automatically release this object when it goes out of scope.
Remarks	The Server fires this event whenever the Server service changes state.

Example	Following is a sample event handler for the OnTransmitterStateChanged event:
	
	`Dim Trans`
	`Sub Trans_OnTransmitterStateChanged( oldState, newState, theTransmitter )`
	`  ' Add code here to handle state change`
	`End Sub`

8

Enumerations

You use Enumerations whenever a property or parameter can have one of a fixed set of possible values.

About enumerations

An enumerated type is a user-defined type consisting of a set of named constants called enumerators, which need not have unique values. Treat the name of each enumerator as a constant. Use Enumerations whenever a property or parameter can have one of a fixed set of possible values. Keep the two important points in mind when using enumerations:

- Automation Clients such as VBA and VBScript do not actually perform runtime checking to ensure proper use of enumerated values.
- Script engines for VBScript and JavaScript do not expose enumerations because they have no concept of data types or namespaces (a reason why they can stay so small.) The scripting environments only see enumerated types as Long values.

To solve the lack of enumeration support in scripting environments, the Object Model exposes the Constants object, which contains a set of properties, one for each constant or enumeration used in the Object Model.

cmAuthenticationEnum

Specifies the authentication service to use when opening a connection to a remote Server.

The value can be one of the following constants:

Table

cmAuthnNone	No authentication.
cmAuthnDCEPrivate	DCE private key authentication.
cmAuthnDCEPublic	DCE public key authentication.
cmAuthnDECPublic	DEC public key authentication (reserved for future use).
cmAuthnWINNT	NT LM SSP (NT Security Service).
cmAuthnDefault	The system default authentication service. Windows NT 4.0 defaults to DCE private key authentication (cmAuthnDCEPrivate).

cmAuthenticationLevelEnum

Specifies the authentication level to use when opening a connection to a remote Server.

The value can be one of the following constants:

Table

cmAuthnLevelNone	No authentication.
cmAuthnLevelConnect	Authenticates only when the Client establishes a relationship with the Server. Datagram transports always use cmAuthnLevelPkt instead.
cmAuthnLevelCall	Authenticates only at the beginning of each remote procedure call when the server receives the request. Datagram transports use cmAuthnLevelPkt instead.
cmAuthnLevelPkt	Authenticates that all data received is from the expected Client.
cmAuthnLevelPktIntegrity	Authenticates and verifies that none of the data transferred between Client and Server has been modified.
cmAuthnLevelPktPrivacy	Authenticates all previous levels and encrypts the argument value of each remote procedure call.

cmAuthorizationEnum

Specifies the authorization service to use when opening a connection to a remote Server.

The value can be one of the following constants:

Table

cmAuthzNone	Server performs no authorization.
cmAuthzName	Server performs authorization based on the Client's principal name.
cmAuthzDCE	Server performs authorization checking using the Client's DCE privilege attribute certificate (PAC) information, which is sent to the Server with each remote procedure call made using the binding handle. Generally, access is checked against DCE access control lists (ACLs).

cmCachedEntryEnum

Specifies the cache entry type. When an entry is created in a cache, the entry type specifies when the entry is automatically purged from the cache.

The value can be one of the following constants:

Table

cmCacheNormal	Normal cache entry; may be deleted to recover space for new entries.
cmCacheStable	Stable cache entry; may be deleted to recover space for new entries only when there are no more entries of type cmCacheNormal.
cmCacheSticky	Sticky cache entry; entry will never be automatically removed from the cache.

cmChannelFilterEnum

Specifies which items to include in the ChannelItems collection.

The value can be one of the following constants:

Table

cmFilterByAll	Enumerate all items.
cmFilterByChannels	Enumerate channels.
cmFilterByChannelSets	Enumerate channel sets.
cmFilterByFolders	Enumerate folders.
cmFilterByNonSystem	Enumerate non-system items.
cmFilterBySystem	Enumerate system items.

cmDaysOfTheMonthEnum

Specifies the days of the month.

This is a combination of one or more of the following constants:

Table

cmTASK_FIRST	Day 1 of the month.
cmTASK_SECOND	Day 2 of the month.
cmTASK_THIRD	Day 3 of the month.

Table

cmTASK_FOURTH	Day 4 of the month.
cmTASK_FIFTH	Day 5 of the month.
cmTASK_SIXTH	Day 6 of the month.
cmTASK_SEVENTH	Day 7 of the month.
cmTASK_EIGHTH	Day 8 of the month.
cmTASK_NINTH	Day 9 of the month.
cmTASK_TENTH	Day 10 of the month.
cmTASK_ELEVENTH	Day 11 of the month.
cmTASK_TWELFTH	Day 12 of the month.
cmTASK_THIRTEENTH	Day 13 of the month.
cmTASK_FOURTHEENTH	Day 14 of the month.
cmTASK_FIFTEENTH	Day 15 of the month.
cmTASK_SIXTEENTH	Day 16 of the month.
cmTASK_SEVENTEENTH	Day 17 of the month.
cmTASK_EIGHTEENTH	Day 18 of the month.
cmTASK_NINETEENTH	Day 19 of the month.
cmTASK_TWENTIETH	Day 20 of the month.
cmTASK_TWENTYFIRST	Day 21 of the month.
cmTASK_TWENTYSECOND	Day 22 of the month.
cmTASK_TWENTYTHIRD	Day 23 of the month.
cmTASK_TWENTYFOURTH	Day 24 of the month.
cmTASK_TWENTYFIFTH	Day 25 of the month.
cmTASK_TWENTYSIXTH	Day 26 of the month.
cmTASK_TWENTYSEVENTH	Day 27 of the month.
cmTASK_TWENTYEIGHTH	Day 28 of the month.
cmTASK_TWENTYNINTH	Day 29 of the month.
cmTASK_THIRTIETH	Day 30 of the month.
cmTASK_THIRTHYFIRST	Day 31 of the month.

cmDaysOfTheWeekEnum

Specifies the days of the week.

This is a combination of one or more of the following constants:

Table

cmTASK_SUNDAY	Sunday
cmTASK_MONDAY	Monday
cmTASK_TUESDAY	Tuesday
cmTASK_WEDNESDAY	Wednesday
cmTASK_THURSDAY	Thursday
cmTASK_FRIDAY	Friday
cmTASK_SATURDAY	Saturday
cmTASK_WEEKDAYS	Monday through Friday, inclusive.
cmTASK_WEEKENDS	Saturday and Sunday

cmDocumentAttributeEnum

Specifies the attributes for a file in a Document channel.

The value can be a sum of any one or more of these constants (default is zero):

Table

cmDocumentHidden	Document is hidden.
cmDocumentClientIsSource	Document is located at the Client.

cmEventOptionsEnum

Specifies the workobject event options. Not all options are valid for all events. For detailed information about each event, including examples, parameters, and options, **see "Work Object Events" on page 293.**

The value can be one or more of the following constants:

Table

cmEvtOptCheckNewer	*Check If Newer*. Transfers a file only if the source file has a more recent date and time stamp than the destination file.
cmEvtOptCheckUpdatesOnly	*Check Updates Only*. Instructs the Server to transfer only files with different sizes or dates.
cmEvtOptCondFalse	*Conditional False*. Executes the event only if the previous event failed or was a "no execute."
cmEvtOptCondTrue	*Conditional True*. Executes the event only if the previously executed event was successful.
cmEvtOptCritical	*Critical Event*. Ends the session if this event fails. A failure is an event that executes but does not finish successfully. Events that do not execute because of conditional options are not considered failures and do not terminate the session.
cmEvtOptDeleteAfter	*Delete After*. Deletes the source file after the file has been transferred. This option also works with other events such as the Append event.
cmEvtOptIncludeSubdir	*Include Subdirectories*. Includes subdirectories with the event. For a registry event, includes registry subkeys.
cmEvtOptMakeTargetPath	*Make Target Path*. Establishes a target path for the event and creates directories when necessary.
cmEvtOptNoCompression	*Turn Compression Off*. Turns off file compression. Use this option for files that do not compress well.
cmEvtOptNoOverwrite	*Do Not Overwrite*. Will not copy an existing file if its name matches that of the destination file.
cmEvtOptNotRequired	*Not Required For Successful Session*. Indicates that this event does not have to execute successfully for the Server to log the session as successful. The Server logs sessions as successful when every event was processed without failure. Completed sessions process all events in the session, but individual events may have failed.

Table

cmEvtOptSafeTransfer	Safe Transfer. Does not create a destination file until it has been successfully transferred. This option instructs the Server to use a hidden temporary file until the file transfer completes. Once complete, the Server renames the temporary file to the destination filename. For unsuccessful transfers, the temporary file remains hidden so the transfer can continue if a retry is executed. Safe transfer ensures that no corruption occurs as a result of an incomplete file transfer.

cmHTMLControlTypeEnum

Specifies the HTML control type. The HTML control type property determines how the channel or channel set published on a Web page runs on the client.

The value can be one of the following constants:

Table

cmHTMLCTText	The channel appears as a hyperlink. Use the HTMLButtonText property to set the text for the hyperlink (default is the channel name).
cmHTMLCTButton	The channel appears as a standard button. Use the HTMLButtonText property to set the button text (default is the channel name).
cmHTMLCTImage	The channel appears as a bitmap. Use the HTMLButtonImage property to set the image path.
cmHTMLCTOnLoad	The Client immediately begins to run the channel when the Web page appears. This item does not appear on the Web page.

cmImpersonationLevelEnum

Specifies the impersonation level to use when you connect to a remote Server.

The value can be one of the following constants:

Table

cmImpLevelAnonymous	(Not supported in this release.) The Client is anonymous to the Server. The Server process cannot obtain identification information about the Client and it cannot impersonate the Client.

Table

cmImpLevelIdentify	The Server can obtain the Client's identity. The Server can impersonate the Client for ACL checking, but cannot access system objects as the Client. This information is obtained when the connection is established, not on every call.
cmImpLevelImpersonate	The Server process can impersonate the Client's security context while acting on behalf of the Client. This information is obtained when the connection is established, not on every call.
cmImpLevelDelegate	The Server process can impersonate the Client's security context while acting on behalf of the client. The Server process can also make outgoing calls to other Servers while acting on behalf of the Client. This information is obtained when the connection is established, not on every call.

cmLDAPNodeTypesEnum

Specifies LDAP Node Types.

The value can be one of the following constants:

Table

cmLDAPNoNodeType	Internal uninitialized state. If LDAP setup completed properly, this should never be seen or used.
cmLDAPOUMemebership	Only OU memberships are considered for assignment checks
cmLDAPOUAndGroupMembership	Both OU and group memberships are considered for assignment checks

cmMonthsOfTheYearEnum

Specifies the months of the year.

This is a combination of one or more of the following constants:

Table

cmTASK_JANUARY	January
cmTASK_FEBRUARY	February
cmTASK_MARCH	March
cmTASK_APRIL	April

Table

cmTASK_MAY	May
cmTASK_JUNE	June
cmTASK_JULY	July
cmTASK_AUGUST	August
cmTASK_SEPTEMBER	September
cmTASK_OCTOBER	October
cmTASK_NOVEMBER	November
cmTASK_DECEMBER	December

cmSortModeEnum

Specifies the sort mode.

The value can be one of the following constants:

Table

cmSortManual	Manual (default)
cmSortByName	By name
cmSortByFolderThenName	By name, with folders always displayed first

cmTransmitterStateEnum

Specifies the state of the Server service.

The value can be one of the following constants:

Table

cmStateStopped	The service is not running.
cmStateStartPending	The service is starting.
cmStateStopPending	The service is stopping.
cmStateRunning	The service is running.
cmStateContinuePending	The service continue is pending.
cmStatePausePending	The service pause is pending.

Table

cmStatePaused	The service is paused.

cmTriggerFlagsEnum

Specifies the trigger's behavior.

This is a combination of one or more of the following constants:

Table

cmTASK_TRIGGER_FLAG_HAS_END_DATE	The trigger's end date is valid. If this flag is not set, the end date data is ignored and the trigger will be valid indefinitely.
cmTASK_TRIGGER_FLAG_KILL_AT_DURATION_END	Task will be terminated at the end of the active trigger's lifetime.
cmTASK_TRIGGER_FLAG_DISABLED	Trigger is inactive.
cmTASK_TRIGGER_FLAG_DURATION_IS_END_TIME	The trigger's lifetime is determined by its start time and duration.

cmTriggerTypeEnum

Specifies the trigger type.

The value can be one of the following constants:

Table

cmTASK_TIME_TRIGGER_ONCE	Trigger is set to run a single time.
cmTASK_TIME_TRIGGER_DAILY	Trigger is set to run on a daily interval.
cmTASK_TIME_TRIGGER_WEEKLY	Trigger is set to run on specific days of the week on a weekly interval.
cmTASK_TIME_TRIGGER_MONTHLYDATE	Trigger is set to run on a specific day(s) of the month.
cmTASK_TIME_TRIGGER_MONTHLYDOW	Trigger is set to run on specific days, weeks, and months.
cmTASK_TIME_TRIGGER_MONTHLYINTERVAL	Trigger is set to run on a specific day of the month on a monthly interval.
cmTASK_EVENT_TRIGGER_ON_IDLE	Trigger is set to run when the system becomes idle.

Table

cmTASK_EVENT_TRIGGER_AT_SYSTEMSTART	Trigger is set to run at system startup.
cmTASK_EVENT_TRIGGER_AT_LOGON	Trigger is set to run when a user logs on.
cmTASK_EVENT_TRIGGER_ONLINE	Trigger is set to run when the user logs onto the Internet (not implemented).
cmTASK_EVENT_TRIGGER_OFFLINE	Trigger is set to run when the user logs off the Internet (not implemented).

cmWeekOfTheMonthEnum

Specifies the week of the month.

This is a combination of one or more of the following constants:

Table

cmTASK_FIRST_WEEK	First week of the month
cmTASK_SECOND_WEEK	Second week of the month
cmTASK_THIRD_WEEK	Third week of the month
cmTASK_FOURTH_WEEK	Fourth week of the month
cmTASK_LAST_WEEK	Last week of the month

cmWorkObjectEventEnum

Specifies the work object event.

Work object events are organized by functional categories:

- **File/Disk Operations**. These events perform file-level data exchange, administration, and information gathering on the Server and the Client.
- **Variables**. These events manipulate placeholders whose contents you control. You can use the predefined Session Manager variables or create your own user-defined variables. User-defined variables can be used in all work objects contained within an individual Session Manager channel. You can also use these events to perform system registry tasks.
- **Session Control**. These events control how Session Manager structures and progresses through the work object's list of events. These events include conditional statements and events that stop the work object, session, and connection.

- **Miscellaneous**. These events display save file and message dialogs, execute programs, send commands to other Windows computers, and run events in an external file.

The value can be one of the following constants:

Table

File/Disk Operations	
cmEvtAppendFile	*Append File*. Add the contents of one or more files to the end of another file.
cmEvtCheckFile	*Check File*. Compare file sizes and dates.
cmEvtCopyFile	*Copy File*. Copy files to another location.
cmEvtDeleteFile	*Delete File*. Remove files from disk.
cmEvtDirectory	*Directory Listing*. Save a directory listing to a file.
cmEvtFileStat	*File Status*. Determine if a file exists.
cmEvtFindFile	*Find File*. Find a file or a directory.
cmEvtGetFile	*Get File from Client*. Get files from the Client to the Server.
cmEvtMakeDir	*Make Directory*. Create a new directory.
cmEvtRemoveDir	*Remove Directory*. Remove a directory from disk.
cmEvtRenameFile	*Rename File*. Modify the names of files.
cmEvtSendFile	*Send File to Client*. Send files from the Server to the Client.
cmEvtWait	*Wait for File to Exist*. Wait for a file to exist.
Variables	
cmEvtCreateRegKey	*Create Registry Key*. Create a key in the Windows Registry.
cmEvtDeleteRegKey	*Delete Registry Key*. Remove a key from the Windows Registry.
cmEvtDeleteRegValue	*Delete Registry Value*. Remove a value from the Windows Registry.
cmEvtDeleteVarFile	*Delete Variable File*. Remove variables from an INI file.
cmEvtGetRegValue	*Get Registry Value*. Get a value from the Windows Registry.
cmEvtIncrement	*Increment Variable*. Increment or Decrement a variable.
cmEvtReadVarFile	*Read Variable File*. Read variables from an INI file.
cmEvtSearchRegistry	*Search Registry*. Search the Windows Registry for a key or a value.
cmEvtSetRegValue	*Set Registry Value*. Set a value in the Windows Registry.
cmEvtSetVariable	*Set Variable*. Create a variable to use when defining events.
cmEvtTestVariable	*Test Variable*. Test a built-in or user-defined variable.
cmEvtUpdateVarFile	*Update Variable File*. Update variables in an INI file.
Session Control	

Table

cmEvtComment	*Comment*. Insert a comment into the event list.
cmEvtDisconnect	*Disconnect*. Disconnect from the Client.
cmEvtElseIf	*Else*. Conditionally execute an alternate block of events.
cmEvtEndIf	*End If*. Mark the end of an IF block.
cmEvtEndRepeat	*End Repeat*. Mark the end of a REPEAT block.
cmEvtEndSession	*End Session*. Stop executing events and end the session.
cmEvtEndWorkobject	*End Work Object*. Stop executing events in this event list.
cmEvtIfTrue cmEvtIfFalse cmEvtIfLessThan cmEvtIfGreaterThan cmEvtIfEqual cmEvtIfLessThanEqual cmEvtIfGreaterThanEqual	*If*. conditionally execute a block of events.
cmEvtRepeatIfTrue cmEvtRepeatIfFalse cmEvtRepeatIfLessThan cmEvtRepeatIfGreaterThan cmEvtRepeatIfEqual cmEvtRepeatIfLessThanEqual cmEvtRepeatIfGreaterThanEqual	*Repeat*. Execute a block of events repeatedly.
Miscellaneous	
cmEvtExecute	*Execute Program*. Run an application program.
cmEvtLogMsg	*Message*. Log a customized user message to the Server Log.
cmEvtNotify	*Notify Program*. Send a message to a program on a local or remote machine.
cmEvtFileSaveDialog	*File Save Dialog*. Display a dialog that allows a user to save received files.
cmEvtInsertWorklist	*Insert Worklist*. Execute another worklist's events.

For more information, see SendList object, and WorkList object. For more detailed information about each of these events, including examples, parameters, and syntax, see Work Object Events.

9

Work Object Events

A work object event represents the smallest unit of work in a worklist and sendlist. Some events are valid for worklists only; others are valid for both worklists and sendlists. Sendlist objects are optimized for sending files and creating directories. Worklist objects provide more control over session activity.

About work object events

A work object event represents the smallest unit of work in a worklist and sendlist. Some events are valid for worklists only; others are valid for both worklists and sendlists. Sendlist objects are optimized for sending files and creating directories. Worklist objects provide more control over session activity.

Work object events are listed in the cmWorkObjectEventEnum enumeration, and are organized by functional categories:

- **File/Disk Operations**. These events perform file-level data exchange, administration, and information gathering on the Server and the Client.
- **Variables**. These events manipulate placeholders whose contents you control. You can use the predefined Session Manager variables or create your own user-defined variables. User-defined variables can be used in all work objects contained within an individual Session Manager channel. You can also use these events to perform system registry tasks.
- **Session Control**. These events control how Session Manager structures and progresses through the work object's list of events. These events include conditional statements and events that stop the work object, session, and connection.
- **Miscellaneous**. These events display save file and message dialogs, execute programs, send commands to other Windows NT computers, and run events in an external file.

The following sections provide detailed information about each work object event, including examples, parameters, and syntax.

File/Disk operations

These work object events perform file-level data exchange, administration, and information gathering on the Server and the Client.

Append event

Table

Enumeration	cmEvtAppendFile
Applies to	Worklist Object
Description	Use the Append event to add the contents of one or more files to the end of another file.
Parameters	[Param1] Source Filename or Wildcard The path name, file name, or wildcard parameter for one or more files to be appended to the destination file. *Example*: "C:\Docs*.*" [Param2] Target Filename Specifies the name of the file to which the source file is being added. *Example*: "C:\DailyDocs\Daily.txt"
Options	Delete after Make target path (default) Not required for successful session Critical event Conditional (True/False)
Execute	On Server On Client (default)
Remarks	If the destination file exists prior to this example, then the file is not appended to itself and it is not included in the list of source files.
Returned Value	N/A

Check File event

Table

Enumeration	cmEvtCheckFile
Applies to	Worklist and Sendlist Objects
Description	The Check File event compares the time, date, and file size for a Server and Client file. This event is often used to test the state of a file before a transfer event.
Parameters	[Param1] Server Filename The drive, path, and file name of the Server file to be compared with the Client file. *Example*: "C:\Doc\Daily.doc" [Param2] Client Filename Specifies the drive, path, and file name for the Client file. *Example*: "D:\Docs\ClientDaily.doc"

Table

Options	Check if newer Not required for successful session Critical event Conditional (True/False)
Execute	N/A
Remarks	The "Check updates only" option is implicitly set.
Returned Value	N/A

Copy File event

Table

Enumeration	cmEvtCopyFile
Applies to	Worklist Objects
Description	The Copy File event duplicates one or more files to another file name or directory.
Parameters	[Param1] Source Filename or Wildcard Specifies the path, file name, or wildcard parameters for one or more files to copy. This event is unsuccessful if the source file does not exist or if the wildcard parameter does not locate any files. *Example*: "C:\Docs*.*" [Param2] Target Filename or Wildcard The path, file name, or directory for the file or directory that will receive the copied files. This value should be a file if the source field is a file or a directory if the source field is a wildcard parameter. *Example*: "C:\Save*.*"
Options	Make target path (default) Not required for successful session Critical event Conditional (True/False)Include subdirectories
Execute	On Server On Client (default)
Remarks	Supports "Include Subdirectories" option so that subdirectories of the Source file spec will be searched and any files matching the Source file spec will also be copied. Supports "Make Target Path" option so that the Target file(s), including any subdirectories in the file spec, that do not exist will be created.
Returned Value	N/A

Delete File event

Table

Enumeration	cmEvtDeleteFile
Applies to	Worklist Objects
Description	The Delete File event permanently removes one or more files from the Server or Client.
Parameters	[Param1] Filename or Wildcard The path, file name, or wildcard parameter for one or more files to delete. *Example*: "C:\Docs*.doc"

Table

Options	Not required for successful session Critical event Conditional (True/False) Include subdirectories
Execute	On Server On Client (default)
Remarks	Supports "Include Subdirectories" option so that subdirectories of the Source file spec will be searched and any files matching the Source file spec will also be deleted.
Returned Value	N/A

Directory Listing event

Table

Enumeration	cmEvtDirectory
Applies to	Worklist Objects
Description	The Directory Listing event copies the list of files in a directory into an output file on the Server. The output file is text and has the same format as a DOS DIR command.
Parameters	[Param1] Server filename for output Instructs the event to create a file at this location on the Server. Enter the directory, path, and file name that will contain the directory listing. The event replaces the file if it already exists. *Example*: "C:\Listings\Dirlist.txt" [Param2] Directory wildcard Specifies the path or wildcard to use to get the directory listing. End the path with a backslash (\) to list the contents of a directory; otherwise, the event only lists the directory name. *Example*: "C:\DailyDocs*.sav"
Options	Make target path (default) Not required for successful session Critical event Conditional (True/False) Include subdirectories
Execute	On Server On Client (default)
Remarks	Supports "Include Subdirectories" option so that subdirectories of the Server file spec will be searched and any files matching the file spec will also be copied. Supports "Make Target Path" option so that the Server file(s), including any subdirectories, in the file spec that do not exist will be created.
Returned Value	N/A

File Status event

Table

Enumeration	cmEvtFileStat
Applies to	Worklist Objects
Description	The File Status event determines whether a file exists at the specified location. Use this event to set the conditional value to true or false based on a file's presence. This event most often precedes a conditional event or an event with the Conditional option enabled.

Table

Parameters	[Param1] Filename The Server attempts to locate a file at the specified path and file name. *Example*: "C:\Docs\Daily.doc"
Options	Not required for successful session Critical event Conditional (True/False)
Execute	On Server On Client (default)
Remarks	N/A
Returned Value	N/A

Find File event

Table

Enumeration	cmEvtFileStat
Applies to	Worklist Objects
Description	The File Status event determines whether a file exists at the specified location. Use this event to set the conditional value to true or false based on a file's presence. This event most often precedes a conditional event or an event with the Conditional option enabled.
Parameters	[Param1] Filename The Server attempts to locate a file at the specified path and file name. *Example*: "C:\Docs\Daily.doc"
Options	Not required for successful session Critical event Conditional (True/False)
Execute	On Server On Client (default)
Remarks	N/A
Returned Value	N/A

Get File from Client

Table

Enumeration	cmEvtGetFile
Applies to	Worklist Objects
Description	The Get File from Client event locates one or more files on the Client and transfers them to the specified location on the Server. Wildcards used with this event will retrieve a group of files whose names have something in common or that are in the same directory.
Parameters	[Param1] Target Server filename or wildcard The path, file name, directory, or wildcard parameters for the file or directory that will receive the transferred file. *Example*: "C:\Updates*.*" [Param2] Source Client filename or wildcard Specifies the path, file name or wildcard parameters for the file(s) to transfer. *Example*: "C:\Files*.*"

Table

Options	Do not overwrite Safe transfer (default) Check updates only (default) Check if newer Delete after Make target path (default) Not required for successful session Turn compression off Critical event Conditional (True/False) Include subdirectories
Execute	N/A
Remarks	Supports "Include Subdirectories" option so that subdirectories of the source file spec will be searched and any files matching the file spec will be transferred. Supports "Make Target Path" option so that the server files, including any directories, in the file spec that do no exist will be created.
Returned Value	N/A

Make Directory event

Table

Enumeration	cmEvtMakeDir
Applies to	Worklist and Sendlist Objects
Description	The Make Directory event creates a new Client or Server directory. As part of a sendlist object, this event creates the directory only if necessary.
Parameters	[Param1] Directory path Specifies the path and directory name of the new directory. *Example*: "C:\Docs\Save"
Options	Not required for successful session Critical event Conditional (True/False)
Execute	On Server On Client (default)
Remarks	The "Make target path" option is implicitly set.
Returned Value	N/A

Remove Directory event

Table

Enumeration	cmEvtRemoveDir
Applies to	Worklist Objects
Description	The Remove Directory event deletes a Client or Server directory. The directory must be empty of files before it can be removed.
Parameters	[Param1] Directory path The path and name of the directory to be removed. *Example*: "C:\Old"
Options	Not required for successful session Critical event Conditional (True/False) Include subdirectories (default)
Execute	On Server On Client (default)
Remarks	Supports "Include Subdirectories" option so that empty subdirectories of the directory will also be removed. A directory is not deleted if it contains any files.
Returned Value	N/A

Rename File event

Table

Enumeration	cmEvtRenameFile
Applies to	Worklist Objects
Description	The Rename File event moves files or changes the name of one or more files on either the Server or Client.
Parameters	[Param1] Source Old filename or wildcard Specifies one or more source files to move or rename. *Example*: "C:\Docs*.*" [Param2] Target New filename or wildcard Enter the path and new file name, or the wildcard when more than one file is involved. Enter a directory to move one or more files without changing their names. *Example*: "C:\Save*.*"
Options	Make target path (default) Not required for successful session Critical event Conditional (True/False) Include subdirectories
Execute	On Server On Client (default)
Remarks	Supports "Include Subdirectories" option so that subdirectories of the Source file spec will be searched and any files matching the file spec field will be renamed. Supports "Make Target Path" option so that the Target file(s), including any directories, in the file spec that do not exist will be created.
Returned Value	N/A

Send File to Client

Table

Enumeration	cmEvtSendFile
Applies to	Worklist and Sendlist Objects
Description	The Send File to Client event transfers one or more Server files to a file or directory on the Client. Using wildcards with this event transfers a group of Server files whose names have something in common or are in the same directory.
Parameters	[Param1] Source Server filename or wildcard Indicates which directory or files to send to the Client. Enter the file name, path name, or directory on the Server. *Example*: "C:\Files*.*" [Param2] Target Client filename or wildcard Places one or more Server files in this location at the Client. Specify the file name, wildcard parameter, or directory for the Client file(s). *Example*: "C:\Updates*.*"
Options	Do not overwrite Safe transfer (default) Check updates only (default) Check in newer Delete after Make target path (default) Not required for successful session Turn compression off Critical event Conditional (True/False) Include subdirectories
Execute	N/A
Remarks	Supports "Include Subdirectories" option so that subdirectories of the Source file spec will be searched and any files matching the file spec field will be sent. Supports "Make Target Path" option so that the Target file(s) including any directories in the file spec that do not exist will be created.
Returned Value	N/A

For multiple files, the Server always assumes that the destination is a directory. For single files, directories must use a backslash " \ " as the last character in the path name, otherwise the Server assumes the name to be the destination file name.

Wait for File to Exist event

Table

Enumeration	cmEvtWait
Applies to	Worklist Objects
Description	The Wait for File to Exist event instructs the session to pause until a Client or Server file exists, or until a specified amount of time elapses, whichever comes first.
Parameters	[Param1] Filename or wildcard Identifies the Client or Server file to locate. *Example*: "C:\Daily.doc" [Param2] Wait time (mm:ss) Specify the time, in minutes and seconds, to wait for the file to exist. Times may range from 0 to 59:59 (zero to 59 minutes and 59 seconds). *Example*: "0" or ":23" or "23:34"
Options	Delete after Not required for successful session Critical event Conditional (True/False)
Execute	On Server On Client (default)
Remarks	Consider using the File Status event if your wait time is close to 0.
Returned Value	N/A

Variables

These events manipulate placeholders whose contents you control. You can use the predefined Session Manager variables or create your own user-defined variables. User-defined variables can be used in all work objects contained within an individual Session Manager channel. You can also use these events to perform system registry tasks.

Create Registry Key event

Table

Enumeration	cmEvtCreateRegKey
Applies to	Worklist Objects
Description	The Create Registry Key event creates a new key in the registry.
Parameters	[Param1] Root key\key1\keyN The complete path and name of the key to be added. *Example*: "HKLM\Software\Key"
Options	Make target path Not required for successful session Critical event Conditional (True/False)
Execute	On Server On Client (default)
Remarks	This event will fail if the parameter is not a valid registry path or if the specified key already exists.
Returned Value	N/A

Delete Registry Key event

Table

Enumeration	cmEvtDeleteRegKey
Applies to	Worklist Objects
Description	The Delete Registry Key event removes a key from the registry.
Parameters	[Param1] Root key\key1\keyN The complete path and name of the registry key to be deleted. *Example*: "HKLM\Software\Key"
Options	Not required for successful session Critical event Conditional (True/False) Include subkeys
Execute	On Server On Client (default)
Remarks	This event fails if the parameter is not a valid registry path or if the parent key from which the key would have been deleted does not exist.
Returned Value	N/A

Delete Registry Value event

Table

Enumeration	cmEvtDeleteRegValue
Applies to	Worklist Objects
Description	The Delete Registry Value event removes a value from the registry.
Parameters	[Param1] Root key\key1\keyN\[value] The complete path and name of the registry value to be deleted. *Example*: "HKLM\Software\Key\Value"
Options	Not required for successful session Critical event Conditional (True/False)
Execute	On Server On Client (default)
Remarks	This event fails if the parameter is not a valid registry path or if the parent key from which the key would have been deleted does not exist.
Returned Value	N/A

Delete Variable File event

Table

Enumeration	cmEvtDeleteVarFile
Applies to	Worklist Objects
Description	The Delete Variable File event removes a value entry from a variable file (*.ini) on the Server or Client.
Parameters	[Param1] Filename The path and file name of the file from which an entry is to be removed. *Example*: "C:\Variables.ini" [Param2] User variable name The name of the user-defined variable for which the value entry is being removed. *Example*: "<%[Section].VarName>"
Options	Not required for successful session Critical event Conditional (True/False)
Execute	On Server On Client (default)
Remarks	N/A
Returned Value	N/A

Get Registry Value event

Table

Enumeration	cmEvtGetRegValue
Applies to	Worklist Objects
Description	The Get Registry Value event gets the value of a specified registry value on Client or Server and makes it available in a specified user-defined variable.
Parameters	[Param1] User variable name The user-defined variable to receive the registry value. *Example*: "<%TempValueFromRegistry>" [Param2] Root key\key1\keyN\[value] The registry path. *Example*: "<HKLM\Software\Key\Value"
Options	Not required for successful session Critical event Conditional (True/False)
Execute	On Server On Client (default)
Remarks	The system also accepts HKLM, HKCU, HKCR, and HKU as abbreviations.
Returned Value	N/A

Increment Variable event

Table

Enumeration	cmEvtIncrement
Applies to	Worklist Objects
Description	The Increment Variable event modifies the value of the specified user variable by the specified amount (positive or negative).
Parameters	[Param1] User variable name The user-defined variable to be incremented by the specified amount. *Example*: "<%Count>" [Param2] Amount The positive or negative amount by which the variable is to be incremented. *Example*: "1"
Options	Conditional (True/False)
Execute	N/A
Remarks	Increment amounts must be positive or negative whole numbers with no separator characters. For example, 5000, not 5,000.

Table

Returned Value	N/A

Read Variable file

Table

Enumeration	cmEvtReadVarFile
Applies to	Worklist Objects
Description	The Read Variable File event sets variable(s) by reading value(s) from a Windows text file.
Parameters	[Param1] Filename Specifies the INI file whose value(s) are to be set as a user variable. *Example*: "C:\Variables.ini" [Param2] User variable name The user-defined variable whose value is to be determined by the specified INI file. *Example 1*: "<%[MySectionName].MySectionVar>" Read MySectionVar entry in MySectionName section of the INI file. *Example 2*: "<%[MySectionName].*>" Read all entries in the MySectionName section of the INI file. The variable name format is <%[MySectionName].EntryName> where EntryName is the name on the left side of the equal sign. *Example 3*: "<%*>" Read all entries in all sections of the INI file. The variable name format is <%[MySectionName].EntryName> where SectionName is the name of the INI file section and EntryName is the name on the left of the equal sign.
Options	Not required for successful session Critical event Conditional (True/False)
Execute	On Server On Client (default)
Remarks	Examples 2 and 3 above work only on the Server, not on the Client.
Returned Value	N/A

Search Registry event

Table

Enumeration	cmEvtSearchRegistry
Applies to	Worklist Objects
Description	The Search Registry event searches the registry on the Client or Server for the specified key or value and places the value found into the specified user-defined variable.
Parameters	[Param1] User variable name The user-defined variable whose value is to be set by the specified registry key. *Example*: "<%FullRegPath>" [Param2] Root key\key1\keyN or value The registry whose value is to be used for the specified user variable. *Example*: "<HKLM\Software\Key\Value"
Options	Not required for successful session Critical event Conditional (True/False) Include subkeys (default)
Execute	On Server On Client (default)
Remarks	N/A
Returned Value	N/A

Set Registry Value event

Table

Enumeration	cmEvtSetRegValue
Applies to	Worklist Objects
Description	The Set Registry Value event sets a specified registry value to the string specified.
Parameters	[Param1] Root key\key1\keyN\[value] The complete name of the value to be set. *Example*: "HKLM\Software\Key\Value" [Param2] Variable or value The user-defined variable whose value is to be set by the specified registry key, or the value to use. *Example*: "C:\Temp"
Options	Make target path (default) Not required for successful session Critical event Conditional (True\False)

Table

Execute	On Server On Client (default)
Remarks	N/A
Returned Value	N/A

Set Variable event

Table

Enumeration	cmEvtSetVariable
Applies to	Worklist Objects
Description	The Set Variable event creates user-defined variables. Once a user variable is defined, it may be used anywhere in a session, including other worklist objects. A user-defined variable does not preserve its data across sessions, except during a restart.
Parameters	[Param1] User variable name Specifies the name for this user-defined variable. The default value is <%VariableName>. *Example*: "<%MyVariable>" [Param2] Value or @indirect file Sets the variable's value or specifies the name of the file that contains the value. When using a file, remember to precede the path and file name with an "@". *Example*: "@C:\NewValue.txt"
Options	Not required for successful session Critical event Conditional (True/False)
Execute	N/A
Remarks	This event works only with text (*.txt) files.
Returned Value	N/A

The variable names should be unique throughout all worklists in a session. Using the Set Variable event on a previously defined variable will change the value. Although this may be needed for some applications, it can lead to unexpected results and side effects across worklists.

Test Variable event

Table

Enumeration	cmEvtTestVariable
Applies to	Worklist Objects
Description	The Test Variable event enables you to test predefined variables by comparing the variable to a known value. The session evaluates the variable and the result is compared to the specified value.

Table

Parameters	[Param1] Variable(s) and/or text The variable or text string to evaluate and compare with the field below. This field may contain up to 260 characters. *Example*: "<%MyVar>" [Param2] Variable(s) and/or text The variable or text string to compare with the value in the first field. This field may contain up to 260 characters. *Example*: "<%MyTestVar>" or "TestText"
Options	Not required for successful session Critical event Conditional (True/False)
Execute	N/A
Remarks	Testing numbers may not give useful results. For example, Test <%Version> <= "1.30" if <%Version> contains "1.4", the comparison will evaluate to True since the comparison stops when the decimal point is identified: "1" = "1".
Returned Value	N/A

Update Variable File event

Table

Enumeration	cmEvtUpdateVarFile
Applies to	Worklist Objects
Description	The Update Variable File event allows user-defined variables to be saved to a Windows INI file on the Client and Server.
Parameters	[Param1] Filename The path and directory of the INI file. *Example*: "C:\Variables.ini" [Param2] User variable name The user-defined variable to be saved to the specified INI file. *Example*: "<%[Section].MyVariable>"
Options	Make target path (default) Not required for successful session Critical event Conditional (True/False)
Execute	On Server On Client (default)
Remarks	Supports "Make Target Path" option so that the file(s), including any directories, in the file spec that do not exist will be created.

Table

Returned Value	N/A

Session Control

These events control how Session Manager structures and progresses through the work object's list of events. These events include conditional statements and events that stop the work object, session, and connection.

Comment event

Table

Enumeration	cmEvtComment
Applies to	Worklist and Sendlist Objects
Description	The Comment event is a non-executable event used to add comments to a worklist or sendlist or to separate event blocks with a blank line. Comment events are ignored at session execution time, but the comment text is displayed in Session Manager.
Parameters	[Param1] Text Box Enter the comment text (up to 251 characters, including line breaks) that you want inserted into the worklist or sendlist. The comment text may span several lines. In VBScript, use Chr(10) to represent a line break. Example: "//" + Chr(10) + "// This is a comment" + Chr(10) + "//"
Options	N/A
Execute	N/A
Remarks	N/A
Returned Value	N/A

Disconnect event

Table

Enumeration	cmEvtDisconnect
Applies to	Worklist Objects
Description	The Disconnect event disconnects the link between the Client and Server.
Parameters	N/A
Options	Conditional (True/False)
Execute	N/A
Remarks	N/A

Table

Returned Value	N/A

Else event

Table

Enumeration	cmEvtElseIf
Applies to	Worklist Objects
Description	The Else conditional event is used in combination with an If event to control the execution of a block of events.
Parameters	N/A
Options	N/A
Execute	N/A
Remarks	N/A
Returned Value	N/A

End If event

Table

Enumeration	cmEvtEndIf
Applies to	Worklist Objects
Description	The End If conditional event is used in combination with other If events to control the execution of a block of events. Place the End If event at the very end of each If block to end the If clause.
Parameters	N/A
Options	N/A
Execute	N/A
Remarks	N/A
Returned Value	N/A

The <, >, <=, and >= operators in Session Manager events compare only integers or strings. Comparisons of integers are terminated by the first non-numeric character. For example, 128.56.22.8 would equal 128.46.22.8 since the comparison stops after the first three digits.

End Repeat event

Table

Enumeration	cmEvtEndRepeat
Applies to	Worklist Objects

Table

Description	The End Repeat conditional event is used with the Repeat event to mark the end of a Repeat block of events. Place End Repeat events at the end of each Repeat event.
Parameters	N/A
Options	N/A
Execute	N/A
Remarks	N/A
Returned Value	N/A

End Session event

Table

Enumeration	cmEvtEndSession
Applies to	Worklist Objects
Description	The End Session event is used to stop a session and disconnect the communication line before the session reaches the end of the worklist or the end of the session. This event is useful for stopping execution in a specific condition rather than continuing the operation or initiating a communication.
Parameters	N/A
Options	Not required for successful session Conditional (True/False)
Execute	N/A
Remarks	N/A
Returned Value	N/A

End Work Object event

Table

Enumeration	cmEvtEndWorkobject
Applies to	Worklist Objects
Description	The End Work Object event ends the currently executing work object. This event does not terminate the link between the Client and the Server, unless there are no more work objects in the session. If there are more work objects to be executed for the session, then the next work objects in the list will be executed.
Parameters	N/A

Table

Options	Not required for successful session Conditional (True/False)
Execute	N/A
Remarks	N/A
Returned Value	N/A

If event

Table

Enumeration	cmEvtIf*
Applies to	Worklist Objects
Description	The If conditional event controls the execution of a block of events in a session. A block of events begins with an If event and ends with an End If event. If the condition specified is true, then all events up to the next Else or End IF event will be executed.
Parameters	For cmEvtIfTrue and cmEvtIfFalse, N/A. For cmEvtIfLessThan, cmEvtIfGreaterThan, cmEvtIfEqual, cmEvtIfLessThanEqual, cmEvtIfGreaterThanEqual: [Param1] Condition A session variable, number, or string that represents the condition to test. *Example*: "<%NumFiles>" [Param2] Condition A session variable, number, or string that represents the condition to test. *Example*: "12"
Options	Not required for successful session Conditional (True/False)
Execute	N/A
Remarks	N/A
Returned Value	N/A

The <, >, <=, and >= operators in Session Manager events compare only integers or strings. Comparisons of integers are terminated by the first non-numeric character. For example, 128.56.22.8 would equal 128.46.22.8 since the comparison stops at the first decimal point.

Repeat event

Table

Enumeration	cmEvtRepeatIf*
Applies to	Worklist Objects
Description	The Repeat event conditionally repeats a block of events. A Repeat block begins with the Repeat event and ends with an End Repeat event. Repeat if previous event is false allows the events to execute if the previous event failed. Repeat if previous event is true allows the events to repeat if the previous event was successful.
Parameters	[Param1] Condition A session variable, number, or string that represents the condition to test. *Example*: "<%NumFiles>" [Param2] Condition A session variable, number, or string that represents the condition to test. *Example*: "12" [Param3] Maximum Timeout The maximum amount of time the Repeat event may execute repeatedly. The value may range in minutes and seconds from 00:00 to 59:59. *Example*: "12" or ":23" or "1:34" [Param4] Inactivity Timeout The maximum amount of time that execution of the event continues when no file transfer occurs. The value may range in minutes and seconds from 00:00 to 59:59. *Example*: "12" or ":23" or "1:34" [Param5] Max Repeats The maximum number of iterations of this Repeat event. Select from 0 (no repeats) to 99 repetitions. Execution stops after the event has been repeated the maximum number of times. *Example*: "5"
Options	Not required for successful session Conditional (True/False)
Execute	N/A
Remarks	[Param1] and [Param2] are ignored for the cmEvtRepeatIfTrue and cmEvtRepeatIfFalse events.
Returned Value	N/A

If no limit is set for timeouts or repeats, a session could become caught in an endless loop. The <, >, <=, and >= operators in Session Manager events compare only integers or strings. Comparisons of integers are terminated by the first non-numeric character. For example, 128.56.22.8 would equal 128.46.22.8 since the comparison stops at the first decimal point.

Miscellaneous events

These events display save file and message dialog boxes, execute programs, send commands to other Windows NT computers, and run events in an external file.

Execute Program event

Table

Enumeration	cmEvtExecute
Applies to	Worklist Objects
Description	The Execute Program event provides the same capability as the DOS command line for running programs. This event launches the program via the information in the Command Line field.
Parameters	[Param1] Command line The path and name of the application's executable file. Include command line options after the file name. **Example**: "notepad.exe myfile.txt"
Options	Not required for successful session Conditional (True/False)
Execute	On Server On Client (default)
Remarks	If you launch a program on the Server, the program will not be visible on the Windows desktop because the program is launched by Afaria, which normally runs as an NT service. To make the program visible, you must configure the Server service to run as the System Account, and you must select the Allow Service to Interact with Desktop option on the Windows Services applet. For security reasons, this is not recommended.
Returned Value	N/A

Message event

Table

Enumeration	cmEvtLogMsg
Applies to	Worklist Objects
Description	The Message event displays a message in the status dialog at the Client or a log message to Server Log.
Parameters	[Param1] Message text or @indirect file Specifies the text of the message to display or the fname of the file that contains the message text. *Example*: "This is a test message" or "@C:\Messages\Message1.txt"
Options	Not required for successful session Critical event Conditional (True/False)
Execute	On Server On Client (default)
Remarks	N/A

Table

Returned Value	N/A

Notify Program event

Table

Enumeration	cmEvtNotify
Applies to	Worklist Objects
Description	The Notify Program event sends a message to the specified named pipe or mailslot on the Server.
Parameters	[Param1] Server Named Pipe or Mail Slot Specifies the pipe name or mailslot on the Server to be notified. *Example*: "\\.\pipe\name" or "\\.\mailslot\name" [Param2] Notify text or @indirect file Specifies the text of the message to send or the filename of the file that contains the message text. *Example*: "this is a test" or "@C:\Notifications\Notify1.txt"
Options	Not required for successful session Critical event Conditional (True/False)
Execute	N/A
Remarks	N/A
Returned Value	N/A

File Save Dialog event

Table

Enumeration	cmEvtFileSaveDialog
Applies to	Worklist Objects
Description	The File Save Dialog event displays a dialog at the Client that prompts the Client to save the specified directory or file. Typically, files are transferred to the Client's temp directory, then this event is used to enable the Client to specify a destination for the files.
Parameters	[Param1] Filename or directory Enter the file name to save or the directory that contains the files to save. *Example*: "C:\Files*.*"
Options	Not required for successful session Critical event Conditional (True/False)
Execute	N/A
Remarks	If the specified directory does not exist or if this event follows a send event with multiple files, an error message appears at the Client instead of the File Save dialog.

Table

Returned Value	N/A

Insert Worklist event

Table

Enumeration	cmEvtInsertWorklist
Applies to	Worklist Objects
Description	The Insert Worklist event enables you to insert one or more events from an external worklist file into a worklist's list of events.
Parameters	[Param1] Worklist filename or @indirect file The filename of the file that contains the worklist file. *Example*: "C:\Files\Worklist.evf" or "C:\Indirect\Insert.ind"
Options	Not required for successful session Critical event Conditional (True/False)
Execute	N/A
Remarks	N/A

10

Afaria Web Services

The Afaria Web services enable you to initiate activity with your Afaria Server without requiring access to the Afaria Administrator application. The services expose methods for controlling actions that you would otherwise perform on the Afaria Administrator application.

Using the Web services

Afaria public web services are hosted by your Afaria Administrator's web server, and reside in the virtual directory you created for Afaria during installation. Services provide methods for accessing specific Afaria features without requiring access to the Afaria Administrator application. You can use these services to include Afaria features in your own programmatic implementation.

Service assemblies are stored in the following path:

http://<localhost>/<virtual_directory>/PublicWebServices/<service_type>/<service_name>.asmx

according to the following path definitions:

- <localhost> — The name of your Afaria Administrator server
- <virtual_directory> — The virtual directory that you defined for Afaria during installation
- <service_type> — The Afaria Public Web service type
- <service_name> — The Afaria Public Web service name

Your Afaria solution includes the following public web services:

- Platform service: outbound notification — Services that provide methods for making outbound notifications
- Platform service: SMS gateway — Services that provide methods for sending a variety of Afaria message types via the Afaria SMS gateway
- Component service: Security Manager — Services that provide methods for Data Security Manager recovering a temporary password for handheld and Windows Data Security Manager Clients

Platform service: outbound notification

Afaria includes the following platform service:

AfariaNotificationRequest

Service — AfariaNotificationRequest

The implementation for the service is located in the following assembly file:

\<virtual_directory>\\PublicWebServices\Outbound\AfariaNotificationRequest.asmx

The service includes the following methods:

NotifyClientsByClientGroupName

NotifyClientsByClientUID

NotifyClientsByClientGUID

NotifyClientsByClientAddress

CancelNotifications

GetNotificationStatus

Method: NotifyClientsByClientGroupName

Uses the list of Client groups to notify the Clients to run the given channel. Will return a valid NotifyContext object if trackStaus is true, otherwise null is returned. The NotifyContext object is necessary if it is desired to track the status of or cancel this notification request.

Parameters

`string afariaServerAddress`	Afaria Server address or hostname.
`string userName`	Name to be logged in Afaria message log.
`string channelName`	Name of the channel the Clients should run.
`string[] clientGroupNameList`	Array of Client group names to be notified.
`bool trackStatus`	Set to true if it is desired to check the status of or cancel this notification request.

Return

`NotifyContext`	Null if trackStatus is false, otherwise an object that must be passed to the GetNotificationStatus and CancelNotifications methods to have them apply to this notification request.

Method: NotifyClientsByClientUID

Uses the list of Client UIDs to notify the Clients to run the given channel or to deliver a remote wipe command. Will return a valid NotifyContext object if trackStaus is true, otherwise null is returned. The NotifyContext object is necessary if it is desired to track the status of or cancel this notification request.

Parameters

`string afariaServerAddress`	Afaria Server address or hostname.
`string userName`	Name to be logged in Afaria message log.
`string channelName`	To define a channel to run, insert the name of the channel the Clients should run. To deliver a remote wipe command, insert one of the following remote wipe variations: • Hard reset — $\$CMD:WIPEAPPDATA • Hard reset, erase external data card — $\$CMD:WIPEALLDATA • Hard reset, block synchronization — $\$CMD:WIPEAPPDATA+EAS • Hard reset, erase external data card, block synchronization — $\$CMD:WIPEALLDATA+EAS
`string[] clientUID_List`	Array of Client UIDs to be notified.
`bool trackStatus`	Set to true if it is desired to check the status of or cancel this notification request.

Return

`NotifyContext`	Null if trackStatus is false, otherwise an object that must be passed to the GetNotificationStatus and CancelNotifications methods to have them apply to this notification request.

Method: NotifyClientsByClientGUID

Uses the list of Client GUIDs to notify the Clients to run the given channel or to deliver a remote wipe command. Will return a valid NotifyContext object if trackStaus is true, otherwise null is returned. The NotifyContext object is necessary if it is desired to track the status of or cancel this notification request.

Parameters

`string afariaServerAddress`	Afaria Server address or hostname.
`string userName`	Name to be logged in Afaria message log.

`string channelName`	To define a channel to run, insert the name of the channel the Clients should run. To deliver a remote wipe command, insert one of the following remote wipe variations: • Hard reset — $\$CMD:WIPEAPPDATA • Hard reset, erase external data card — $\$CMD:WIPEALLDATA • Hard reset, block synchronization — $\$CMD:WIPEAPPDATA+EAS • Hard reset, erase external data card, block synchronization — $\$CMD:WIPEALLDATA+EAS
`string[] clientGUID_List`	Array of Client GUIDs to be notified.
`bool trackStatus`	Set to true if it is desired to check the status of or cancel this notification request.

Return

`NotifyContext`	Null if trackStatus is false, otherwise an object that must be passed to the GetNotificationStatus and CancelNotifications methods to have them apply to this notification request.

Method: NotifyClientsByClientAddress

Uses the list of IP addresses to notify the Clients to run the given channel. Will return a valid NotifyContext object if trackStaus is true, otherwise null is returned. The NotifyContext object is necessary if it is desired to track the status of or cancel this notification request.

Parameters

`string afariaServerAddress`	Afaria Server address or hostname.
`string userName`	Name to be logged in Afaria message log.
`string channelName`	Name of the channel the Clients should run.
`string[] clientAddressList`	Array of Client addresses (IP or hostname) to be notified.
`bool trackStatus`	Set to true if it is desired to check the status of or cancel this notification request.

Return

`NotifyContext`	Null if trackStatus is false, otherwise an object that must be passed to the GetNotificationStatus and CancelNotifications methods to have them apply to this notification request.

Method: CancelNotifications

Cancels the remaining notifications from the notification request represented by the supplied NotifyContext object.

Parameters

`NotifyContext notifyContext` — The object returned by the notification request that you wish to cancel.

Method: GetNotificationStatus

Gets the status for the notification request represented by the supplied NotifyContext object.

Parameters

`NotifyContext notifyContext` — The object returned by the notification request for which you desire status.

Return

`NotifyStatus` — An object containing the current status of the notification request associated with the supplied NotifyContext object as follows:

NotifyStatus object
Properties

`uint TotalClientsToNotify` – Total number of Clients to be notified by this notification request.

`uint NumCurrentNotifyClient` – Sequence number of the current Client being notified.

`string CurrentClientName` – Name of the current Client being notified.

`string CurrentClientAction` – Description of the action currently taking place on the current Client if there is one or on the notification request as a whole if there is not.

Service notes

Please note the following caveats:

- When you are notifying Clients by Client GUID, you will need to enclose this value in { } (curly brackets) in order for it to be recognized as a GUID.

- By default, Anonymous Access is turned off for everything in the Afaria virtual directory and Windows Integrated Authentication is turned on.
- For HTML, you are not required to setup any network credentials because Internet Explorer will pass on the logon user's credentials with the HTTP GET or POST request. This will be a user name and password that has administrative rights for the Afaria Server. You will also need to specify the domain for the user name. Otherwise you will need to turn on Anonymous Access for the PublicWebServices directory.
- For applications, you are required to setup the network credentials on the Webservice object. Refer to System.Net.NetworkCredential class in the MSDN Library. You can also turn on Anonymous Access in IIS only for the PublicWebServices folder in the virtual directory.

Platform service: SMS gateway

Afaria includes the following platform service:

SMSGatewayWebService

This service provides a programmatic interface for using the Afaria SMS gateway and its defined Short Message Service Centers (SMSCs) for many of Afaria's client deployment features, including Over-The-Air (OTA) notifications, Open Mobile Alliance Client Provisioning (OMA CP) notifications.

Service — SMSGatewayWebService

The implementation for the service is located in the following assembly file:

\<virtual_directory>\PublicWebServices\SMSGateway\SMSGatewayWebService.asmx

Methods are implemented using proprietary data structures and Microsoft Client Datasets. The service includes the following methods:

GetSMSGatewayStatus, GetSMSGatewayStatusDS

GetAccessPoints, GetAccessPointsDS

GetAccessPointWithTenant, GetAccessPointDSWithTenant

GetDevices, GetDevicesDS

GetDevicesWithTenant, GetDevicesDSWithTenant

GetPlatforms, GetPlatformsDS

GetPlatformsWithTenant, GetPlatformsDSWithTenant

GetPackages, GetPackagesDS

GetPackagesWithTenant, GetPackagesDSWithTenant

SendOTANotification

SendClientProvisioning

SendText

SendBinary

GetSendStatus

CancelSend

Method: GetSMSGatewayStatus

Retrieve status of the Afaria SMS gateway and any defined SMSC.

Interface:

```
SMSGtwyStatus GetSMSGatewayStatus(string strServerAddr)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.

Return

SMSGtwyStatus

```
public SMSGtwyErrCode m_eRetVal;
public bool m_bOnline;
public long m_lSent;
public long m_lRcvd;
public long m_lQued;
public long m_lFailed;
public SMSGtwySMSCStatus[] m_SMSC

public enum SMSGtwyErrCode
{
    SMSGW_OKAY = 0,
    SMSGW_GENERAL_ERR = 0x0001000,
    SMSGW_INPUT_ERR,
    SMSGW_DEVICE_ERR,
    SMSGW_PLATFORM_ERR,
    SMSGW_DEPLOY_ERR,
    SMSGW_ACCESSPOINT_ERR,
    SMSGW_SUPPORT_ERR,
    SMSGW_SERVICE_ERR,
    SMSGW_DBACCESS_ERR,
    SMSGW_PACKAGE_ERR,
    SMSGW_SEND_ERR
}

public struct SMSGtwySMSCStatus
{
        public SMSGtwySMSCType m_eSMSCType;
        public string m_strName;
        public bool m_bOnline;
        public long m_lSent;
        public long m_lRcvd;
        public long m_lQued;
        public long m_lFailed;
}

public enum SMSGtwySMSCType
{
        SMSGW_SMSC_UNKNOWN = 0,
        SMSGW_SMSC_AT,
        SMSGW_SMSC_SMPP,
        SMSGW_SMSC_MAX
}
```

Method: GetSMSGatewayStatusDS

Retrieve status of the Afaria SMS gateway and any defined SMSC via a standard, self-describing, Microsoft DataSet.

Interface:

```
DataSet GetSMSGatewayStatusDS(string strServerAddr)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.

Return

`DataSet`	DataSet describing server and defined SMSC status.

Method: GetAccessPoints

Retrieve an array of access points, as defined on the Afaria Administrator application's client deployment page. Using this method in a multitenant environment operates on the system tenant.

Interface:

```
SMSGtwyAccessPoints GetAccessPoints(string strServerAddr)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.

Return

`SMSGtwyAccessPoint`	Structure containing return code as well as available access points.

```
public struct SMSGtwyAccessPoints
{
    public SMSGtwyErrCode m_eRetVal;
    public SMSGtwyAccessPoint[] m_AccessPoints;
}

public enum SMSGtwyErrCode
{
    SMSGW_OKAY = 0,
    SMSGW_GENERAL_ERR = 0x0001000,
    SMSGW_INPUT_ERR,
    SMSGW_DEVICE_ERR,
    SMSGW_PLATFORM_ERR,
    SMSGW_DEPLOY_ERR,
    SMSGW_ACCESSPOINT_ERR,
    SMSGW_SUPPORT_ERR,
    SMSGW_SERVICE_ERR,
    SMSGW_DBACCESS_ERR,
    SMSGW_PACKAGE_ERR,
    SMSGW_SEND_ERR
}

public struct SMSGtwyAccessPoint
    {
        public string m_strName;
        public string m_strDescription;
    };
```

Method: GetAccessPointDS

Retrieve a standard, self-describing, Microsoft DataSet of access points, as defined on the Afaria Administrator application's client deployment page. Using this method in a multitenant environment operates on the system tenant.

Interface:

```
DataSet GetAccessPointsDS(string strServerAddr)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.

Return

`DataSet`	`DataSet describing available access points.`

Method: GetAccessPointsWithTenant

In a multitenant environment, retrieve an array of access points for a tenant, as defined on the Afaria Administrator application's client deployment page.

Interface:

```
SMSGtwyAccessPoints GetAccessPointsWithTenant(string strServerAddr, long
lTenantID)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.
`long lTenantID`	The tenant's ID.

Return

`SMSGtwyAccessPoints`	Structure containing return code as well as available access points.

```
public struct SMSGtwyAccessPoints
{
    public SMSGtwyErrCode m_eRetVal;
    public SMSGtwyAccessPoint[] m_AccessPoints;
}

public enum SMSGtwyErrCode
{
    SMSGW_OKAY = 0,
    SMSGW_GENERAL_ERR = 0x0001000,
    SMSGW_INPUT_ERR,
    SMSGW_DEVICE_ERR,
    SMSGW_PLATFORM_ERR,
    SMSGW_DEPLOY_ERR,
    SMSGW_ACCESSPOINT_ERR,
    SMSGW_SUPPORT_ERR,
    SMSGW_SERVICE_ERR,
    SMSGW_DBACCESS_ERR,
    SMSGW_PACKAGE_ERR,
    SMSGW_SEND_ERR
}

public struct SMSGtwyAccessPoint
    {
        public string m_strName;
        public string m_strDescription;
    };
```

Method: GetAccessPointsDSWithTenant

In a multitenant environment, retrieve a dataset of access points for a tenant, as defined on the Afaria Administrator application's client deployment page.

Interface:

```
DataSet GetAccessPointsDSWithTenant(string strServerAddr, long
lTenantID)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.
`long lTenantID`	The tenant's ID.

Return

`DataSet`	DataSet describing available access points.

Method: GetDevices

Retrieve an array of supported device types, as defined on the Afaria Administrator application's client deployment page. Using this method in a multitenant environment operates on the system tenant.

Interface:

```
SMSGtwyDevices GetDevices(string strServerAddr)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.

Return

`SMSGtwyDevices`

```
public struct SMSGtwyDevices
{
    public SMSGtwyErrCode m_eRetVal;
    public SMSGtwyDevice[] m_Devices;
}
public enum SMSGtwyErrCode
    {
        SMSGW_OKAY = 0,
        SMSGW_GENERAL_ERR = 0x0001000,
        SMSGW_INPUT_ERR,
        SMSGW_DEVICE_ERR,
        SMSGW_PLATFORM_ERR,
        SMSGW_DEPLOY_ERR,
        SMSGW_ACCESSPOINT_ERR,
        SMSGW_SUPPORT_ERR,
        SMSGW_SERVICE_ERR,
        SMSGW_DBACCESS_ERR,
        SMSGW_PACKAGE_ERR,
        SMSGW_SEND_ERR
    }
public struct SMSGtwyDevice
{
    public string m_strName;
    public string m_strDescription;
    public string m_strPlatform;
}
```

Method: GetDevicesDS

Retrieve a dataset of supported device types, as defined on the Afaria Administrator application's client deployment page. Using this method in a multitenant environment operates on the system tenant.

Interface:

```
DataSet GetDevicesDS (string strServerAddr)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.

Return

`DataSet`	DataSet containing devices.

Method: GetDevicesWithTenant

In a multitenant environment, retrieve an array of supported device types, as defined on the Afaria Administrator application's client deployment page.

Interface:

```
SMSGtwyDevices GetDevicesWithTenant(string strServerAddr, long
lTenantID)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.
`long lTenantID`	The tenant's ID.

Return

`SMSGtwyDevices`

```
public struct SMSGtwyDevices
{
    public SMSGtwyErrCode m_eRetVal;
    public SMSGtwyDevice[] m_Devices;
}

public enum SMSGtwyErrCode
    {
        SMSGW_OKAY = 0,
        SMSGW_GENERAL_ERR = 0x0001000,
        SMSGW_INPUT_ERR,
        SMSGW_DEVICE_ERR,
        SMSGW_PLATFORM_ERR,
        SMSGW_DEPLOY_ERR,
        SMSGW_ACCESSPOINT_ERR,
        SMSGW_SUPPORT_ERR,
        SMSGW_SERVICE_ERR,
        SMSGW_DBACCESS_ERR,
        SMSGW_PACKAGE_ERR,
        SMSGW_SEND_ERR
    }
public struct SMSGtwyDevice
{
    public string m_strName;
    public string m_strDescription;
    public string m_strPlatform;
}
```

Method: GetDevicesDSWithTenant

In a multitenant environment, retrieve a dataset of supported device types, as defined on the Afaria Administrator application's client deployment page.

Interface:

```
DataSet GetDevicesDSWithTenant (string strServerAddr, long lTenantID)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.
`long lTenantID`	The tenant's ID.

Return

`DataSet`	DataSet containing devices.

Method: GetPlatforms

Retrieve an array of supported platforms, as defined on the Afaria Administrator application's client deployment page. Using this method in a multitenant environment operates on the system tenant.

Interface:

```
SMSGtwyPlatforms GetPlatforms (string strServerAddr)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.

Return

`SMSGtwyPlatforms`

```
public struct SMSGtwyPlatforms
{
    public SMSGtwyErrCode m_eRetVal;
    public SMSGtwyPlatform[] m_Platforms;
}

public enum SMSGtwyErrCode
    {
        SMSGW_OKAY = 0,
        SMSGW_GENERAL_ERR = 0x0001000,
        SMSGW_INPUT_ERR,
        SMSGW_DEVICE_ERR,
        SMSGW_PLATFORM_ERR,
        SMSGW_DEPLOY_ERR,
        SMSGW_ACCESSPOINT_ERR,
        SMSGW_SUPPORT_ERR,
        SMSGW_SERVICE_ERR,
        SMSGW_DBACCESS_ERR,
        SMSGW_PACKAGE_ERR,
        SMSGW_SEND_ERR
    }

public struct SMSGtwyPlatform
{
    public string m_strName;
    public string m_strDescription;
    public string m_strDeploymentPackage;
    public bool m_bOmaCpSupport;
    public bool m_bServiceSupport;
}
```

Method: GetPlatformsDS

Retrieve a dataset of supported platforms, as defined on the Afaria Administrator application's client deployment page. Using this method in a multitenant environment operates on the system tenant.

Interface:

```
DataSet GetPlatformsDS (string strServerAddr)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.

Return

`DataSet`	Dataset containing platforms.

Method: GetPlatformsWithTenant

In a multitenant environment, retrieve an array of supported platforms, as defined on the Afaria Administrator application's client deployment page.

Interface:

```
SMSGtwyPlatforms GetPlatformsWithTenant (string strServerAddr, long
lTenantID)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.
`long lTenantID`	The tenant's ID.

Return

`SMSGtwyPlatforms`

```
public struct SMSGtwyPlatforms
{
    public SMSGtwyErrCode m_eRetVal;
    public SMSGtwyPlatform[] m_Platforms;
}

public enum SMSGtwyErrCode
    {
        SMSGW_OKAY = 0,
        SMSGW_GENERAL_ERR = 0x0001000,
        SMSGW_INPUT_ERR,
        SMSGW_DEVICE_ERR,
        SMSGW_PLATFORM_ERR,
        SMSGW_DEPLOY_ERR,
        SMSGW_ACCESSPOINT_ERR,
        SMSGW_SUPPORT_ERR,
        SMSGW_SERVICE_ERR,
        SMSGW_DBACCESS_ERR,
        SMSGW_PACKAGE_ERR,
        SMSGW_SEND_ERR
    }

public struct SMSGtwyPlatform
{
    public string m_strName;
    public string m_strDescription;
    public string m_strDeploymentPackage;
    public bool m_bOmaCpSupport;
    public bool m_bServiceSupport;
}
```

Method: GetPlatformsDSWithTenant

In a multitenant environment, retrieve a dataset of supported platforms, as defined on the Afaria Administrator application's client deployment page.

Interface:

```
DataSet GetPlatformsDSWithTenant (string strServerAddr, long lTenantID)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.
`long lTenantID`	The tenant's ID.

Return

`DataSet`	Dataset containing platforms.

Method: GetPackages

Retrieve an array of defined Afaria OTA deployment packages, as published by the Afaria OTA Publisher application. Using this method in a multitenant environment operates on the system tenant.

Interface:

```
SMSGtwyPackages GetPackages (string strServerAddr)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.

Return

`SMSGtwyPackages`

```
public struct SMSGtwyPackages
{
    public SMSGtwyErrCode m_eRetVal;
    public SMSGtwyPackage[] m_Packages;
}


public enum SMSGtwyErrCode
    {
        SMSGW_OKAY = 0,
        SMSGW_GENERAL_ERR = 0x0001000,
        SMSGW_INPUT_ERR,
        SMSGW_DEVICE_ERR,
        SMSGW_PLATFORM_ERR,
        SMSGW_DEPLOY_ERR,
        SMSGW_ACCESSPOINT_ERR,
        SMSGW_SUPPORT_ERR,
        SMSGW_SERVICE_ERR,
        SMSGW_DBACCESS_ERR,
        SMSGW_PACKAGE_ERR,
        SMSGW_SEND_ERR
    }

public struct SMSGtwyPackage
{
    public string m_strName;
    public string m_strDescription;
    public string m_strPackageID;
}
```

Method: GetPackagesDS

Retrieve a dataset of defined Afaria OTA deployment packages, as published by the Afaria OTA Publisher application. Using this method in a multitenant environment operates on the system tenant.

Interface:

```
Dataset GetPackagesDS (string strServerAddr)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.

Return

`DataSet`	Dataset containing packages.

Method: GetPackagesWithTenant

In a multitenant environment, retrieve an array of defined Afaria OTA deployment packages, as published by the Afaria OTA Publisher application.

Interface:

```
SMSGtwyPackages GetPackagesWithTenant (string strServerAddr, long lTenantID)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.
`long lTenantID`	The tenant's ID.

Return

`SMSGtwyPackages`

```
public struct SMSGtwyPackages
{
    public SMSGtwyErrCode m_eRetVal;
    public SMSGtwyPackage[] m_Packages;
}

public enum SMSGtwyErrCode
    {
        SMSGW_OKAY = 0,
        SMSGW_GENERAL_ERR = 0x0001000,
        SMSGW_INPUT_ERR,
        SMSGW_DEVICE_ERR,
        SMSGW_PLATFORM_ERR,
        SMSGW_DEPLOY_ERR,
        SMSGW_ACCESSPOINT_ERR,
        SMSGW_SUPPORT_ERR,
        SMSGW_SERVICE_ERR,
        SMSGW_DBACCESS_ERR,
        SMSGW_PACKAGE_ERR,
        SMSGW_SEND_ERR
    }

public struct SMSGtwyPackage
    {
        public string m_strName;
        public string m_strDescription;
        public string m_strPackageID;
    }
```

Method: GetPackagesDSWithTenant

In a multitenant environment, retrieve a dataset of defined Afaria OTA deployment packages, as published by the Afaria OTA Publisher application. Using this method in a multitenant environment operates on the system tenant.

Interface:

```
Dataset GetPackagesDSWithTenant (string strServerAddr, long lTenantID)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.
`long lTenantID`	The tenant's ID.

Return

`DataSet`	`Dataset containing packages.`

Method: SendOTANotification

Use the Afaria SMS gateway to send an Afaria OTA deployment notification message to one or more mobile phone numbers.

Interface:

```
SMSGtwySendContext SendOTANotification(bool bTrackStatus, string strServerAddr,
string strUserName, string strCommaDelimPhoneNumbers, string strSubject, string
strDevice, string strPlatform, string strPackageID)
```

Parameters

`bool bTrackStatus`	Set to true to track status, false otherwise.
`string strServerAddr`	Afaria Server address or hostname.
`string strUserName`	Name of user making request.
`string strCommaDelimPhoneNumbers`	List of comma-delimited mobile phone numbers.
`string strSubject`	Subject field for message.
`string strDevice`	Optional. Specifies the device type for the message. If defined, this should imply platform type.
`string strPlatform`	Optional. Specifies the platform type to receive the message. If defined, this should imply the specific deployment package to send when strPackageID is null. Priority: takes priority over strDevice.
`string strPackageID`	Optional. Specifies deployment package for provisioning. If this parameter is not null, then the client provisioning message will contain a “home page” setting. If this parameter is null, the client provisioning message will not contain a “home page” setting. Priority: takes priority over strPlatform.
Note	At least one of the input fields, strDevice, strPlatform, or strPackageID must be provided. If more than one is provided, priority rules apply.

Return

`SMSGtwySendContext`	Context structure to track notification status.

Method: SendClientProvisioning

Use the Afaria SMS gateway to send a client provisioning message using OMA CP protocol to one or more mobile phone numbers.

Interface:

```
SMSGtwySendContext SendClientProvisioning(bool bTrackStatus, string
strServerAddr, string strUserName, string strCommaDelimPhoneNumbers, string
strDevice, string strPlatform, string strPackageID, string strAccessPoint,
string strUserPIN, string strNetwPIN)
```

Parameters

`bool bTrackStatus`	Set to true to track status, false otherwise.
`string strServerAddr`	Afaria Server address or hostname.
`string strUserName`	Name of user making request.
`string strCommaDelimPhoneNumbers`	List of comma-delimited mobile phone numbers.
`string strDevice`	Optional. Specifies the device type for the notification. If defined, this should imply platform type.
`string strPlatform`	Optional. Specifies the platform type to receive the message. If defined, this should imply the specific deployment package to send when strPackageID is not null. Priority: takes priority over strDevice.
`string strPackageID`	Optional. Specifies deployment package for provisioning. • IF null — Client provisioning message will not contain a “home page” setting. • IF NOT null AND package ID string matches an existing client deployment package — Client provisioning message will contain a “home page” setting. • IF NOT null AND package ID string does not match an existing client deployment package AND strPlatform is defined with a package — Client provisioning message will contain a “home page” setting. • IF NOT null AND package ID string does not match an existing client deployment package AND strPlatform is not defined with a package — Client provisioning message will not contain a “home page” setting. Priority: takes priority over strPlatform.
`string strAccessPoint`	Name of a valid access point definition to send in the provisioning message.

`string strUserPIN`	Optional. Specifies the User PIN to use as part of the provisioning message.
`string strNetwPIN`	Optional. Specifies the Network PIN to use as part of the provisioning message; if used, this field must specify the International Mobile Subscriber Identity (IMSI) of a particular phone (SIM card). IMSI is typically a 15-character field. Additionally, only the phone number associated with the specific IMSI should be specified in strPhoneNumber.
Note	Either strDevice or strPlatform must be provided; if more than one is provided, priority rules apply.

Return

`SMSGtwySendContext`	Context structure to track provisioning status.

Method: SendText

Use the Afaria SMS gateway to send a text message to one or more mobile phone numbers.

Interface:

```
SMSGtwySendContext SendText(bool bTrackStatus, string strServerAddr, string
strUserName, string strCommaDelimPhoneNumbers, string strPayload, string
strUserPIN string strNetwPIN)
```

Parameters

`bool bTrackStatus`	Set to true to track status, false otherwise.
`string strServerAddr`	Afaria Server address or hostname.
`string strUserName`	Name of user making request.
`string strCommaDelimPhoneNumbers`	List of comma-delimited mobile phone numbers.
`string strPayload`	Message to send to device. Note: If strPayload refers to a properly formatted XML document (e.g., Service Indication), it may be compiled and sent out as WBXML by the SMS gateway.
`string strUserPIN`	Optional. Specifies the User PIN to use as part of the provisioning message. This parameter is used only if strPayload refers to a correctly formed OMA CP message.
`string strNetwPIN`	Optional. Specifies the Network PIN to use as part of the provisioning message; if used, this field must specify the IMSI of a particular phone (SIM card). IMSI is typically a 15-character field. Additionally, only the phone number associated with the specific IMSI should be specified in strPhoneNumber. This parameter is only used if strPayload refers to a correctly formed OMA CP message.

Return

`SMSGtwySendContext`	Context structure to track message status.

Method: SendBinary

Use the Afaria SMS gateway to send a binary message to one or more mobile phone numbers.

Interface:

```
SendBinary (bool bTrackStatus, string strServerAddr, string strUserName, string
strCommaDelimPhoneNumbers, string strPayload)
```

Parameters

`bool bTrackStatus`	Set to true to track status, false otherwise.
`string strServerAddr`	Afaria Server address or hostname.
`string strUserName`	Name of user making request.
`string strCommaDelimPhoneNumbers`	List of comma-delimited mobile phone numbers.
`string strPayload`	Message to send to device. Note: It is assumed strPayload refers to a properly formatted binary SMS message, beginning with a UDH description. This message will be sent straight to the SMPP service, as defined on the Afaria Administrator's service configuration page for the SMS gateway, without any processing.

Return

`SMSGtwySendContext`	Context structure to track message status.

Method: GetSendStatus

Retrieve the status of an active send.

Interface:

```
SMSGtwySendStatus GetSendStatus(string strServerAddr, SMSGtwySendContext
context)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.
`context (SMSGtwySendContext)`	Context associated with a previous send.

Return

`SMSGtwySendStatus`	Status of requested send.

```
public struct SMSGtwySendStatus
{
    public SMSGtwyErrCode m_eRetVal;
    public System.UInt32 m_uiTotal;
    public System.UInt32 m_uiCurrent;
    public System.UInt32 m_uiSuccess;
    public System.UInt32 m_uiFailure;
    public string m_strCurrentName;
    public string m_strCurrentAction;
}
```

Method: CancelSend

Cancels an active send.

Interface:

```
SMSGtwyErrCode CancelSend(string strServerAddr, SMSGtwySendContext context)
```

Parameters

`string strServerAddr`	Afaria Server address or hostname.
`context (SMSGtwySendContext)`	Context associated with a previous send.

Return

`SMSGtwyErrCode`	Returns cancel status.

```
public enum SMSGtwyErrCode
{
    SMSGW_OKAY = 0,
    SMSGW_GENERAL_ERR = 0x0001000,
    SMSGW_INPUT_ERR,
    SMSGW_DEVICE_ERR,
    SMSGW_PLATFORM_ERR,
    SMSGW_DEPLOY_ERR,
    SMSGW_ACCESSPOINT_ERR,
    SMSGW_SUPPORT_ERR,
    SMSGW_SERVICE_ERR,
    SMSGW_DBACCESS_ERR,
    SMSGW_PACKAGE_ERR,
    SMSGW_SEND_ERR
}
```

Component service: Security Manager

Afaria includes the following Data Security Manager component service:

secmgrtemppasswordretriever

Service — secmgrtemppasswordretriever

The implementation for the service is located in the following application extension file:

\<virtual_directory>\
PublicWebServices\SecurityManager\SecMgrTempPasswordRetriever.asmx

The service includes the following methods:

GetTemporaryRecoveryPassword

GetTemporaryRecoveryPasswordWin32

Method: GetTemporaryRecoveryPassword

Uses an initial back door value from an Afaria Data Security Manager handheld lient to generate a temporary password for the client to accept.

Parameters

`string ClientGUID`	Afaria Client identifier.
`string AfariaServerAddress`	Afaria Server address or host name.
`string InitialBackdoorValue`	The initial back door value from an Afaria Data Security Manager Client. The initial back door value is the key value located on the Client's Forgot Password screen.

Return

`string <XMLSOAP>`	XML SOAP response for the user's default Web browser that contains the 16-digit temporary password string in hexadecimal format. You may need to adjust your browser's security settings for scripts to ensure that your browser does not block the response.

Method: GetTemporaryRecoveryPasswordWin32

Uses an initial back door value from an Afaria Data Security Manager Windows client to generate a temporary password for the client to accept.

Parameters

`string ClientGUID`	Afaria Client identifier.
`string AfariaServerAddress`	Afaria Server address or host name.

`string Byte1`
`string Byte2`
`string Byte3`
	The initial back door value from an Afaria Data Security Manager client. The initial back door value is the last three pairs of characters from the challenge code. The client exposes the challenge code when the client is in a state of denied access.

Return

`string <XMLSOAP>`	XML SOAP response for the user's default Web browser that contains the 16-digit temporary password string in hexadecimal format. You may need to adjust your browser's security settings for scripts to ensure that your browser does not block the response.

11

Client API

The Client API allows you to write programs that interact with the Server by initiating, monitoring, and controlling channel connections.

The APIs automation interface exposes properties and methods for initiating and controlling channel connections. The control also starts events to allow a Client application to monitor the status of the channel connection.

Windows and Windows CE Client API

The Windows Client API for Windows 32-bit devices and Pocket PC devices allows you to write programs that interact with the Server by initiating, monitoring, and controlling channel connections.

The APIs automation interface exposes properties and methods for initiating and controlling channel connections. The control also starts events to allow a Client application to monitor the status of the channel connection.

You can invoke the ActiveX Control API for use in a variety of programming or scripting languages, including container applications.

The Client Control is located in either the XeClient.DLL (Windows 32-bit) or the XeClientCE.DLL (Windows CE), which is automatically installed and registered along with the Client.

Client status window

You can use the APIs to control the Client's status window:

- To *enable* the window on connection, type AClientControl.ShowWindow = 1
- To *disable* the window on connection, type AclientControl.ShowWindow = 0

Properties

When you use the Client API, you must ensure the Channel Property contains the full path to the channel in one of the following ways:

<TransmitterName>\[<FolderName>]\<ChannelName>

where the Server name is case-sensitive [ ] denotes optional

OR

$root$\<ChannelName>

where $root$ refers to the Server property.

Note: The fully-qualified path name for a channel must be less than 80 characters.

This allows channels with the same name on different Servers.

Properties	*Description*
Authenticate	This property is optional and applies only for Windows client types. This property indicates whether to require user authentication. *Type*: BOOL *Mode*: Read/Write
Channel	This property contains the name of the channel. You must set this property before trying to initiate a connection. *Type*: BSTR *Mode*: Read/Write
Password	This property is optional and applies only for Windows client types. This property contains the user name for channel authentication. You must set this property before trying to initiate a connection. *Type*: BSTR *Mode*: Read/Write
Transmitter	This property contains the name of the Server. You must set this property before trying to initiate a connection. *Type*: BSTR *Mode*: Read/Write
Address	This property contains the IP address or DNS Name for the Server. You must set this property before trying to initiate a connection. *Type*: BSTR *Mode*: Read/Write
RSFarmID	This property contains ASCII text strings that your relay server uses to direct incoming client communication to your Afaria server, as defined in the relay server's configuration file and as configured on the Afaria server configuration. *Type*: string *Mode*: Read/Write
RSInfo	This property contains ASCII text strings that your relay server uses to direct incoming client communication to your Afaria server, as configured on the Afaria server configuration. Syntax: url_prefix=<ClientPrefix>, where ClientPrefix is the Client URL prefix you configured on your Afaria server. *Type*: string *Mode*: Read/Write
Status	This property contains the status of the connection. *Type*: BSTR *Mode*: Read only
StatusText	This property contains a text representation of the status of the connection. You can query the property once a connection has ended. *Type*: BSTR *Mode*: Read/Write

Properties	Description
UserName	This property is optional and applies only for Windows CE client types. This property contains the user name for channel authentication. The Client connects as the device name if you do not set the property. *Type*: BSTR *Mode*: Read only
UserPassword	This property is optional and applies only for Windows CE client types. This property contains the password for channel authentication. *Type*: BSTR *Mode*: Read only
Visible	This property is optional and applies only for Windows CE client types. This property prevents displaying any user interface elements, including the authentication dialog. *Type*: BOOL *Mode*: Read/Write

Methods

Method	*Description*
Connect()	Initiates a channel connection on the server and receives interim status messages until returning at the end of the session. Returns 0 if successful, otherwise returns an error code.
ConnectAndWait()	This method is only for Windows CE clients. Initiates a channel connection on the specified server and waits for the session to complete without receiving interim status messages. Returns 0 if successful, otherwise returns an error code.
StopWaitingEvent()	This method is only for Windows CE clients. Passes an event to the ConnectAndWait method that can cause ConnectAndWait to return before the session ends. Returns 0 if successful, otherwise returns an error code.
Cancel()	Cancels the currently running channel. *Return Value*: none

OnMessageText event

OnMessageText(long lType, BSTR Text)

This event advises a Client that the message text has changed. There are two types of messages: Normal and Control. Normal messages are informative and are often displayed in a message list box. Control messages indicate the state of the connection.

Message Type	*Description*
Normal Messages	XEC_MSG_FATAL = 0 Provides general information about the status of the current connection. XEC_MSG_ERROR = 1 Indicates an error occurred while processing a session event. The text provides more information about the error. XEC_MSG_INFO = 2 Provides information about current event processing. XEC_MSG_DEBUG = 3 Provides developer-defined message text. XEC_MSG_USER = 4 Provides user-defined message text. User-defined messages are specified by the "Message" event.
Control Messages	XEC_MSG_SESSION_STATUS = -2 Provides general information about the status of the current connection. XEC_MSG_FILE_STATUS = -3 Provides information about the current event being executed. For example, while transferring a file, you might receive the message "Sending File: c:\file\data.dat" XEC_MSG_FILE_INFO = -4 Provides detailed information about the current event being executed. For example, while transferring a file, you might receive the message "85k of 115k." XEC_MSG_TRANSMITTER_ID = -5 Provides a string indicating a unique Server identifier. XEC_MSG_SESSION_ENCRYPTED = -6 Indicates the channel is encrypted. XEC_MSG_LISTING_RAN = -7 Indicates the Server ran the "channel listings." XEC_MSG_USER_AUTH_COOKIE = -8 Indicates the Server sent an updated cookie with new authentication or assignment information.

OnProgress event

OnProgress(long lProgress)

This event marks the progress of a specific event. You can use this event to increment a progress bar. For example, while transferring a file, you might view a graphical progress bar of the transfer's percentage complete based on the information from this event. The lProgress parameter ranges from 0 – 100.

OnEndSession event

OnEndSession(long lStatus, BSTR StatusText)

This event signals the end of a connection. The lStatus and StatusText parameters convey the final connection status. All valid status codes are listed below:

Status code	*Description*
XEC_STATUS_SUCCESSFUL = 0	Indicates a successful connection.
XEC_STATUS_CANCELLED = 1	Indicates that the user cancelled the connection. Also indicates the Server refused the connection.
XEC_STATUS_INTERRUPTED = 2	Indicates that the connection timed-out while waiting for a response from the Server or that the connection has been lost.
XEC_STATUS_TRANSMITTER_NOT_FOUND = 3	Indicates an invalid Server address.
XEC_STATUS_TRANSMITTER_UNAVAILABLE = 4	Indicates the Server is presently unavailable.
XEC_STATUS_CHANNEL_UNDEFINED = 5	Indicates the channel was not defined on the Server.
XEC_STATUS_CHANNEL_UNPUBLISHED = 6	Indicates the channel is not published on the Server.
XEC_STATUS_TRANSMITTER_PASSWORD = 7	Indicates the Server requires a password.
XEC_STATUS_TRANSMITTER_VERSION = 8	Indicates that the Server cannot communicate with the currently installed version of the Afaria software.
XEC_STATUS_CHANNEL_ENCRYPTED = 9	Indicates a discrepancy between the Client and the Server about whether the channel should be encrypted. For example, the Server generates the appropriate encryption keys for a secure connection, but the Client does not generate its encryption keys.
XEC_STATUS_TRANSMITTER_HIDDEN = 10	Indicates the Server is hidden.
XEC_STATUS_AUTHENTICATION_REQUIRED = 13	Usually occurs when the Client thinks a channel doesn't need authentication and the Server thinks a channel does need authentication.
XEC_STATUS_USER_AUTH_FAILED = 14	Indicates a Client has entered an incorrect user name and/or password.
XEC_STATUS_USER_AUTH_ERROR = 15	Indicates a user authentication subsystem error.

Status code	*Description*
XEC_STATUS_USER_NOT_ASSIGNED = 16	Indicates Client user is not a member of any group assigned to this channel.
XEC_STATUS_USER_ASN_ERROR = 17	Indicates a user assignments subsystem error.
XEC_STATUS_NO_UPDATES = 18	Indicates that even thought the Client has run the Software Updates channel, the Server has determined there are no updates.
XEC_STATUS_TCPIP_NOT_INSTALLED = 19	Indicates TCP/IP has not been installed at the Client.
XEC_STATUS_NO_VALID_WORKOBJS = 20	Indicates when a channel has no work objects assigned and therefore has nothing to do.
XEC_STATUS_SOFTWARE_UPDATE_IN_PROGRESS = 21	Session aborted because an Electronic Software Delivery (ESD) update is in progress.
XEC_STATUS_USER_AUTH_CRYPT_MODE = 22	Client cryptographic capability is incompatible with server.
XEC_STATUS_NOT_APPROVED = 23	Client not approved to run sessions.
XEC_STATUS_SERVICE_NOT_RUNNING = 24	(Windows Vista clients only) Service not running.
XEC_STATUS_ALREADY_QUEUED = 25	A request for the current channel is already in the server's queue.
XEC_STATUS_NOT_APPROVED_PKG_CODE = 26	The client passed an invalid client package authentication code.
XEC_STATUS_GENERAL_FAILURE = 100	General failures may include the absence of Microsoft Cryptography API (found in advapi32.dll); a TCP/IP error; or a channel subsystem error, such as an error because of an unlisted channel.
XEC_STATUS_UNKNOWN = 101	An error that does not belong in any other category listed in this section.

Windows CE Client .NET API

The Windows client API for Windows CE devices allows you to write programs that interact with the server by initiating, monitoring, and controlling channel connections. The APIs .NET class interface exposes properties and methods for initiating and controlling channel connections. The control also starts events to allow a client application to monitor the status of the channel connection.

The Windows CE client .NET API consists of two binaries: the .NET assembly to be referenced by the client application (XeManagedClient.dll) and the helper library to be installed with the client application (ClientAPIShim.dll).

Deprecation notice – Sybase anticipates deprecating the Windows CE client .NET API in an upcoming release. Compact Framework 2 or later users should stop using the .NET API and transfer functionality to the COM API as soon as possible in anticipation of this event. Compact Framework pre-2 users may want to target the COM API for new development.

The XeManagedClient.dll assembly contains classes that make up the .NET API:

- *XeManagedClient*. Contains the main functionality of the API.
- *XeClientException*. The exception class for errors.
- *XeMessage*. An enum class for all message types.
- *XeStatus*. An enum class for all status types.
- *MessageCallback*. A delegate class used to capture Message Events.
- *ProgressCallback*. A delegate class used to capture Progress Events.
- *StatusCallback*. A delegate class used to capture Status Events.

All classes are contained in the namespace “Xcellenet.ClientAPI”.

The Windows CE client .NET API binaries are not installed by default along with the client.

Client status window

You can use the .NET API to control the client’s status window:

- To enable the window on connection, use XeManagedClient.ShowWindow = true.
- To disable the window on connection, use XeManagedClient.ShowWindow = false.

Properties

When you use the client API, you must ensure the Channel Property contains the full path to the channel in one of the following ways:

<TransmitterName>\[<FolderName>]\<ChannelName>

where the server name is case-sensitive and [ ] denotes optional

OR

$root$\<ChannelName>

where $root$ refers to the server property.

Note: The fully-qualified path name for a channel must be less than 80 characters.

This allows channels with the same name on different servers.

Properties	*Description*
Channel	This property contains the name of the channel. You must set this property before trying to initiate a connection. *Type*: string *Mode*: Read/Write
Address	This property contains the IP address or DNS Name for the server. You must set this property before trying to initiate a connection. *Type*: string *Mode*: Read/Write
Transmitter	This property contains the name of the server. You must set this property before trying to initiate a connection. *Type*: string *Mode*: Read/Write
Status	This property contains the latest status returned by the server. *Type*: XeStatus *Mode*: Read only
StatusText	This property contains the text for the latest status message from the server. *Type*: string *Mode*: Read only
UserName	This property contains the user name for channel authentication. *Type*: string *Mode*: Read/Write
Password	This property contains the password for channel authentication. You must set this property before trying to initiate a connection. *Type*: string *Mode*: Read/Write

Properties	*Description*
RSFarmID	This property contains ASCII text strings that your relay server uses to direct incoming client communication to your Afaria server, as defined in the relay server's configuration file and as configured on the Afaria server configuration. *Type*: string *Mode*: Read/Write
RSInfo	This property contains ASCII text strings that your relay server uses to direct incoming client communication to your Afaria server, as configured on the Afaria server configuration. *Type*: string *Mode*: Read/Write

Methods

Method	*Description*
Connect()	Initiates a channel connection on the server. Upon error, this method raises an exception of type XeClientException. Return Value: None
Cancel()	Cancels the currently running channel. Upon error, this method raises an exception of type XeClientException. Return Value: None

Event MessageEvent

MessageEvents can be captured with a delegate of type MessageCallback(XeMessage eMessage ,string strMessage). This event advises a client that the message text has changed. There are two types of messages: Normal and Control. Normal messages are informative and are often displayed in a message list box. Control messages indicate the state of the connection.

eMessage Description

Normal messages

Message	*Description*
XEC_MSG_FATAL = 0	Provides general information about the status of the current connection.
XEC_MSG_ERROR = 1	Indicates an error occurred while processing a session event. The text provides more information about the error.
XEC_MSG_INFO = 2	Provides information about current event processing.
XEC_MSG_DEBUG = 3	Provides developer-defined message text.
XEC_MSG_USER = 4	Provides user-defined message text. User-defined messages are specified by the "Message" event.

Control Messages

Message	*Description*
XEC_MSG_SESSION_STATUS = -2	Provides general information about the status of the current connection.
XEC_MSG_FILE_STATUS = -3	Provides information about the current event being executed. For example, while transferring a file, you might receive the message "Sending File: c:\file\data.dat"
XEC_MSG_FILE_INFO = -4	Provides detailed information about the current event being executed. For example, while transferring a file, you might receive the message "85k of 115k."
XEC_MSG_TRANSMITTER_ID = -5	Provides a string indicating a unique server identifier.
XEC_MSG_SESSION_ENCRYPTED = -6	Indicates the channel is encrypted.
XEC_MSG_LISTING_RAN = -7	Indicates the server ran the "channel listings."
XEC_MSG_USER_AUTH_COOKIE = -8	Indicates the server sent an updated cookie with new authentication or assignment information.

Event ProgressEvent

ProgressEvents can be captured with a delegate of type ProgressCallback(int iProgress). This event marks the progress of a specific event. You can use this event to increment a progress bar. For example, while transferring a file, you might view a graphical progress bar of the transfer's percentage complete based on the information from this event. The lProgress parameter ranges from 0-100.

Event StatusEvent

StatusEvents can be captured with a delegate of type StatusCallback(XeStatus eStatus, string strStatus). This event signals the end of a connection. The eStatus and strStatus parameters convey the final connection status. All valid status codes are listed below:

Status code	*Description*
XEC_STATUS_SUCCESSFUL = 0	Indicates a successful connection.
XEC_STATUS_CANCELLED = 1	Indicates that the user cancelled the connection. Also indicates the server refused the connection.

Status code	Description
XEC_STATUS_INTERRUPTED = 2	Indicates that the connection timed-out while waiting for a response from the server or that the connection has been lost.
XEC_STATUS_TRANSMITTER_NOT_FOUND = 3	Indicates an invalid server address.
XEC_STATUS_TRANSMITTER_UNAVAILABLE = 4	Indicates the server is presently unavailable.
XEC_STATUS_CHANNEL_UNDEFINED = 5	Indicates the channel was not defined on the server.
XEC_STATUS_CHANNEL_UNPUBLISHED = 6	Indicates the channel is not published on the server.
XEC_STATUS_TRANSMITTER_PASSWORD = 7	Indicates the server requires a password.
XEC_STATUS_TRANSMITTER_VERSION = 8	Indicates that the server cannot communicate with the currently installed version of the Afaria software.
XEC_STATUS_CHANNEL_ENCRYPTED = 9	Indicates a discrepancy between the client and the server about whether the channel should be encrypted. For example, the server generates the appropriate encryption keys for a secure connection, but the client does not generate its encryption keys.
XEC_STATUS_TRANSMITTER_HIDDEN = 10	Indicates the server is hidden.
XEC_STATUS_AUTHENTICATION_REQUIRED = 13	Usually occurs when the client thinks a channel doesn't need authentication and the server thinks a channel does need authentication.
XEC_STATUS_USER_AUTH_FAILED = 14	Indicates a client has entered an incorrect user name and/or password.
XEC_STATUS_USER_AUTH_ERROR = 15	Indicates a user authentication subsystem error.
XEC_STATUS_USER_NOT_ASSIGNED = 16	Indicates client user is not a member of any group assigned to this channel.
XEC_STATUS_USER_ASN_ERROR = 17	Indicates a user assignments subsystem error.
XEC_STATUS_NO_UPDATES = 18	Indicates that even thought the client has run the Software Updates channel, the server has determined there are no updates.
XEC_STATUS_TCPIP_NOT_INSTALLED = 19	Indicates TCP/IP has not been installed at the client.
XEC_STATUS_NO_VALID_WORKOBJS = 20	Indicates when a channel has no work objects assigned and therefore has nothing to do.
XEC_STATUS_SOFTWARE_UPDATE_IN_PROGRESS = 21	Session aborted because an Electronic Software Delivery (ESD) update is in progress.
XEC_STATUS_USER_AUTH_CRYPT_MODE = 22	Client cryptographic capability is incompatible with server.
XEC_STATUS_NOT_APPROVED = 23	Client not approved to run sessions.
XEC_STATUS_SERVICE_NOT_RUNNING = 24	(Windows Vista clients only) Service not running.

Status code	Description
XEC_STATUS_ALREADY_QUEUED = 25	A request for the current channel is already in the server's queue.
XEC_STATUS_NOT_APPROVED_PKG_CODE = 26	The client passed an invalid client package authentication code.
XEC_STATUS_GENERAL_FAILURE = 100	General failures may include the absence of Microsoft Cryptography API (found in advapi32.dll); a TCP/IP error; or a channel subsystem error, such as an error because of an unlisted channel.
XEC_STATUS_UNKNOWN = 101	An error that does not belong in any other category listed in this section.

Java Client API

The Java Client API allows you to write programs that interact with the Server by initiating and monitoring channel connections.

The ClientApi class exposes various methods of connecting to the Server. You can extend the ClientApi class, as shown in the sample provided in "Sample Java Client API" on page 389, to monitor the progress of a session by overriding the appropriate send methods.

connect

The connect method initiates a request to connect to the Server, returning an status code. If the connect request fails, a non-zero status code is returned; generally, it will return XEC_STATUS_GENERAL_FAILURE (100). Further information may returned through sendDebugMsg(). Refer to "Status codes" on page 388 for status code definitions.

When requesting a channel by name, you must ensure the channelName parameter contains the full path to the channel in one of the following ways:

<ServerName>\[<FolderName>]\<ChannelName>

where the Server name is case-sensitive and [ ] denotes optional

OR

$root$\<ChannelName>

In this case, $root$ refers to the Server property.

Note: The fully-qualified path name for a channel must be less than 80 characters.

```
/**
  * Connect to a transmitter and execute the specified channel.
  * <p>
  *
  * @param ipAddress The IP address or registered
  * domain name of the Transmitter/Server.
  * @param channelID The channel unique identifier to be executed.
  *
  * @return an xecStatus if connection is successful, otherwise false.
  */
public final int connect( String ipAddress, int channelID )
{
}
/**
  * Connect to a transmitter and execute the specified channel.
  * <p>
```

```
  *
  * @param username The username of the connecting user (can be null).
  * @param password The password for the user connecting (can be null).
  * @param ipAddress The IP address or registered
  * domain name of the Transmitter/Server.
  * @param ipPort If IP port of the server
  * @param channelName The name of the channel to
  * execute (can be null.  ignored if channelID is non-zero).
  * @param channelID The channel unique identifier to be executed.
  *
  * @return an xecStatus if connection is successful, otherwise false.
  */
public final int connect( String username, String password, String ipAddress, int
ipPort,  String channelName, int channelID )
{
}
/**
  * Connect to a transmitter and execute the session information contained in
  * the XEC file passed.
  * <p>
  *
  * @param xecFile The name of the XEC file to process.
  */
public final int connect( String xecFile )
{
}
```

sendConnInfoChange

Called anytime a change in connection status has occurred. The information contained in the string passed is the connection status change method.

```
/**
* Sends a connection information change message
* <p>
*
* @param info The connection information change message.
* /
public void sendConnInfoChange (String info)
{
}
```

sendWorkInfoChange

Called anytime a work event status has occurred during the execution of the channel session. The information contained in the string passed is the work event status change message.

```
/**
* Sends a work event information change message.
* <p>
*
* @param info The work event change message.
* /
public void sendWorkInfochange (String info)
{
}
```

sendTransInfoChange

Called to provide file transfer status information. The information contained in the string passed is the transfer status change message.

```
/**
* Sends a File or Data Transfer information change message.
* <p>
*
* @param info The transfer information change message.
* /
public void sendTransInfoChange (String info)
{
}
```

sendUserMsg

Called to provide user information feedback from the Server. The information contained in the string passed is the user information feedback message.

```
/**
* Sends a user notification message.
* <p>
*
* @param msg The notification message.
* /
public void sendUserMsg (String msg)
{
}
```

sendInfoMsg

Called to provide user warnings from the Server. The information contained in the string passed is the warning message.

```
/**
* Sends a warning notification message.
* <p>
*
* @param msg The notification message.
* /
public void sendInfoMsg (String msg)
{
}
```

sendFatalMsg

Called anytime a fatal error has occurred during channel execution. The information contained in the string passed is the error message.

```
/**
* Sends a Fatal Error notification message.
* <p>
*
* @param msg The fatal notification message.
* /
public void sendFatalMsg (String msg)
{
}
```

sendDebugMsg

Called to provide additional debugging or status information. The information contained in the string passed is the debug message.

```
/**
 * Sends a debug notification message.
 * <p>
 *
 * @param msg The debug notification message.
 */
public void sendDebugMsg (String msg)
{
}
```

sendErrorMsg

Called anytime a recoverable error has occurred during channel execution. The information contained in the string passed is the error message.

```
/**
* Sends an Error notification message.
* <p>
*
* @param msg The error notification message.
* /
public void sendErrorMsg (String msg)
{
}
```

sendProgressInfo

Called to provide file transfer progress information. The integer value passed relates the percentage that has been transferred for the file transfer process.

```
/**
* Sends a progress update value for file transfers.
* <p>
*
* @param iPercent the progress update value.
* /
public void sendProgressInfo (int iPercent)
{
}
```

sendSessionActive

Called only when a channel session has been established with a Server. The information contained in the string passed is the channel connection activation message.

```
/**
* Sends a session active notification message.
* <p>
*
* @param msg The session active notification message.
* /
public void sendSessionActive (String msg)
```

sendSessionComplete

If the Connect method's connect request is successfully submitted, as indicated when calling the connect method returns XEC_STATUS_SUCCESSFUL (0), sendSessionComplete is invoked at the termination of the session/connection to provide a final status code. Refer to "Status codes" on page 388 for status code definitions.

```
/**
 * Sends a session complete value.
 * <p>
 *
 * @param xecStatus The session XEC status value.
 */
public void sendSessionComplete (int xecStatus)
{
}
```

sendSessionDeactive

Called to provide session status message. The information contained in the string passed is the session status message.

```
/**
 * Sends a session deactive notification message.
 * <p>
 *
 * @param msg The session deactive notification message.
 */
public void sendSessionDeactive (String msg)
{
}
```

Status codes

The following status codes are returned by the Connect method or sent to the API via sendSessionComplete.

Status code	*Description*
XEC_STATUS_SUCCESSFUL = 0	Indicates a successful connection.
XEC_STATUS_CANCELLED = 1	Indicates that the user cancelled the connection. Also indicates the Server refused the connection.
XEC_STATUS_INTERRUPTED = 2	Indicates that the connection timed-out while waiting for a response from the Server or that the connection has been lost.
XEC_STATUS_TRANSMITTER_NOT_FOUND = 3	Indicates an invalid Server address.
XEC_STATUS_TRANSMITTER_UNAVAILABLE = 4	Indicates the Server is presently unavailable.
XEC_STATUS_CHANNEL_UNDEFINED = 5	Indicates the channel was not defined on the Server.
XEC_STATUS_CHANNEL_UNPUBLISHED = 6	Indicates the channel is not published on the Server.
XEC_STATUS_TRANSMITTER_PASSWORD = 7	Indicates the Server requires a password.
XEC_STATUS_TRANSMITTER_VERSION = 8	Indicates that the Server cannot communicate with the currently installed version of the Afaria software.
XEC_STATUS_CHANNEL_ENCRYPTED = 9	Indicates a discrepancy between the Client and the Server about whether the channel should be encrypted. For example, the Server generates the appropriate encryption keys for a secure connection, but the Client does not generate its encryption keys.
XEC_STATUS_TRANSMITTER_HIDDEN = 10	Indicates the Server is hidden.
XEC_STATUS_AUTHENTICATION_REQUIRED = 13	Usually occurs when the Client thinks a channel doesn't need authentication and the Server thinks a channel does need authentication.
XEC_STATUS_USER_AUTH_FAILED = 14	Indicates a Client has entered an incorrect user name and/or password.
XEC_STATUS_USER_AUTH_ERROR = 15	Indicates a user authentication subsystem error.
XEC_STATUS_USER_NOT_ASSIGNED = 16	Indicates Client user is not a member of any group assigned to this channel.
XEC_STATUS_USER_ASN_ERROR = 17	Indicates a user assignments subsystem error.
XEC_STATUS_NO_UPDATES = 18	Indicates that even thought the Client has run the Software Updates channel, the Server has determined there are no updates.
XEC_STATUS_TCPIP_NOT_INSTALLED = 19	Indicates TCP/IP has not been installed at the Client.

Status code	Description
XEC_STATUS_NO_VALID_WORKOBJS = 20	Indicates when a Server listing channel has no work objects assigned and therefore has nothing to do.
XEC_STATUS_GENERAL_FAILURE = 100	General failures may include the absence of Microsoft Cryptography API (found in advapi32.dll); a TCP/IP error; or a channel subsystem error, such as an error because of an unlisted channel.
XEC_STATUS_UNKNOWN = 101	An error that does not belong in any other category listed in this section.

Sample Java Client API

In the sample presented below, the messaging methods in ClientAPI are overwritten to write the output to the console.

```
import com.afaria.client.api.ClientApi;

/**
 * This program illustrates how to call the Afaria Java Client API.
 * <p>
 * In order to make use of the Java Client API,
 * one must override the ClientAPI class to supply
 * your own message handling.
 * <p>
 * In order to compile this class,
 * one must have the CLASSPATH environment variable pointing to Afaria.jar.
 * Typically this resides in $HOME/.Afaria.
 * To compile, type
 * javac apiTest
 * <p>
 * To run this sample, type
 * java apiTest -i <ipaddress> -c <channelID>
 *
 * If the current directory and/or the Afaria.jar file are not in CLASSPATH, type
 * java -cp [path_to_Afaria.jar/]Afaria.jar:. apiTest -i <ipaddress> -c
<channelID>
 */
public class apiTest
{

/*
 *This default implementation of the ClientAPI
 *  provides an override of the ClientApi
 *  class by redirection all messaging output to the java console. In order to do
 *  something more useful with this API, please include processing logic in the
```

```
 *  messaging methods.
 */
class MyClientApi extends ClientApi
{
  /**
  * Sends a connection information change message.
  * <p>
  *
  * @param info The connection information change message.
  */
  public void sendConnInfoChange(String info)
  {
  System.out.println ( "ConnInfoChange: " + info );
  }


  /**
  * Sends a work event information change message.
  * <p>
  *
  * @param info The work event change message.
  */
  public void sendWorkInfoChange(String info)
  {
  System.out.println ( "WorkInfoChange: " + info );
  }


  /**
  * Sends a File or Data Transfer information change message.
  * <p>
  *
  * @param info The transfer information change message.
  */
  public void sendTransInfoChange(String info)
  {
  System.out.println ( "TransInfoChange: " + info );
  }


  /**
  * Sends a user notification message.
  * <p>
  *
  * @param msg The notification message.
  */
  public void sendUserMsg( String msg )
  {
  System.out.println ( "UserMsg: " + msg );
  }

  /**
```

```
* Sends a warning notification message.
* <p>
*
* @param msg The notification message.
*/
public void sendInfoMsg( String msg )
{
System.out.println ( "InfoMsg: " + msg );
}


/**
* Sends a Fatal Error notification message.
* <p>
*
* @param msg The fatal notification message.
*/
public void sendFatalMsg( String msg )
{
System.out.println ( "FatalMsg: " + msg );
}


/**
* Sends a debug notification message.
* <p>
*
* @param msg The debug notification message.
*/
public void sendDebugMsg( String msg )
{
System.out.println ( "DebugMsg: " + msg );
}


/**
* Sends an Error notification message.
* <p>
*
* @param msg The error notification message.
*/
public void sendErrorMsg( String msg )
{
System.out.println ( "ErrorMsg: " + msg );
}


/**
* Sends a progress update value for file transfers.
* <p>
*
* @param iPercent The progress update value.
```

```
  */
  public void sendProgressInfo( int iPercent )
  {
  System.out.println ( "ProgressInfo: " + iPercent );
  }


  /**
  * Sends a session active notification message.
  * <p>
  *
  * @param msg The session active notification message.
  */
  public void sendSessionActive( String msg )
  {
  System.out.println ( "SessionActive: " + msg );
  }


      /**
       * Sends a session deactive notification message.
       * <p>
       *
       * @param msg The session deactive notification message.
       */
       public void sendSessionDeactive( String msg )
       {
           System.out.println ( "SessionDeactive: " + msg );
       }


       /**
        * Sends a session complete value.
        * <p>
        *
        * @param xecStatus The session XEC status value.
        */
        public void sendSessionComplete( int xecStatus )
        {
            System.out.println ( "SessionComplete: " + xecStatus );
        }
    }


/**
 * Default Constructor
 */
public apiTest()
{
}

/**
```

```
 * Entry point into main method
 */
public static void main( String[] args )
{
    try
    {
  apiTest app = new apiTest();
  MyClientApi theclient = app.new MyClientApi();
  if ( !app.parseArgs(args) )
      System.exit(0);

        //Demonstration of the 3 versions of the connect method
        int status = 0;

            if (m_connectmethod.equalsIgnoreCase("XEC"))
            {
                System.out.println("XEC file = " + m_xecfile);
                status = theclient.connect( m_xecfile );
            }
            else if (apiTest.m_connectmethod.equalsIgnoreCase("verbose"))
            {
                System.out.println("address = " + m_ipaddress);
                System.out.println("channel id = " + m_channelid);
                System.out.println("channel name = " + m_username);
                System.out.println("user name = " + m_username);
                System.out.println("password = " + m_password);
                //use default port for protocol by hard-coding 0
                //if you want to test other ports/protocols specify address as
              //[protocol://]address[:port] where protocol is xnet, xnets, http,
or https
                 status = theclient.connect( m_username, m_password, m_ipaddress,
0,  m_channelName, m_channelid  );
            }
            else
            {
                System.out.println("address = " + m_ipaddress);
                System.out.println("channel id = " + m_channelid);
                status = theclient.connect( m_ipaddress, m_channelid );
            }
            if (0 == status)
                System.out.println("Connect request submitted successfully");
            else
              System.out.println ( "Connect request submission failed with error
code " + status);
  }
  catch(Exception e)
  {
e.printStackTrace();
  }
}

/**
```

```
 * Parse Arguments routine
 */
public boolean parseArgs( String [] args )
{
  boolean ret_ = true;
  boolean bWeHaveNoData_ = true;

  if (( null == args ) || (0 == args.length))
  {
    System.err.println( "Usage:");
    System.err.println( "apiTest -f <xecfilepath> | -u <username> -p <pswd> -i
<ipAddress> -n <Channel Name> -c <Channel ID Number>");
    System.err.println( "");
    ret_ = false;
    return ret_;
  }

  for ( int i = 0; i < args.length && bWeHaveNoData_; i++ )
  {
    if ( args[i].length() > 1 && ( args[i].charAt(0) == '-' ) )
    {
      if ( i == args.length - 1 )
      {
        ret_ = false;
        break;
      }

      // The master switch
      switch ( args[i].charAt(1) )
      {
        case 'f':
          m_xecfile = args[++i];
              m_connectmethod = "XEC";
          break;
        case 'u':
          m_username = args[++i];
              m_connectmethod = "verbose";
          break;
        case 'p': // password
          m_password = args[++i];
              m_connectmethod = "verbose";
          break;
        case 'i': // ipaddress
          m_ipaddress = args[++i];
          if ( null == m_ipaddress || 0 >= m_ipaddress.trim().length())
          {
            ret_ = false;
            bWeHaveNoData_ = false;
            break;
          }
          break;
        case 'n': // Channel name
```

```
          m_channelName = args[++i];
              m_connectmethod = "verbose";
          break;
        case 'c': // Channel ID
          m_channel = args[++i];
          if ( null == m_channel || 0 >= m_channel.trim().length())
          {
            ret_ = false;
            bWeHaveNoData_ = false;
            break;
          }
  m_channelid = new Integer( m_channel ).intValue();
          break;
        default:
          ret_ = false;
          break;
      } // switch ( args[i].charAt(1) )
    } // if ( args[i].length() > 1 && ( args[i].charAt(0) == '-' ) )
  } // for ( int i = 0; i < args.length && bWeHaveNoData; i++ )

  return ret_;
} // boolean parseArgs( String [] args )

    public static String m_connectmethod = "simple";
    private static String m_username = null;
    private static String m_password = null;
    private static String m_ipaddress = null;
    private static String m_channelName = null;
    private static String m_channel = null;
    private static String m_xecfile = null;
    private static int m_channelid = 0;
}
```

Palm Client API

The Afaria API method removes the need to load a shared library. Instead, applications simply fill out an `AfariaConnectCmd` structure and launch the Afaria Client with a call to `SysUIAppSwitch ()`. This results in the application being sent an `appStopEvent` and exiting. The Palm OS then launches the Afaria Client with the `AfariaConnectCmd` structure being passed in the parameter block. The Afaria Client runs the session and then uses `SysUIAppSwitch()` to return back to the calling application. The launch code indicates to the application that the Afaria Client has returned. At this point the application can continue however it sees fit.

This method may require changes to existing applications if the calling application has any state information that needs to be saved before exiting and retrieved upon returning.

Running an Afaria session

Finding Afaria

You can find Afaria by calling the Palm OS function

DmGetNextDatabaseByTypeCreator:

Example

```
DmSearchStateType srchState;
Uint              cardNo;
LocalID         dbID;

DmGetNextDatabaseByTypeCreator (true, &srchState, 'appl', 'XNET', true,
&cardNo, &dbID);
```

DmGetNextDatabaseByTypeCreator passes back the card number and the ID of the viewer application.

Launching Afaria to run a session

Applications that wish to run a session must first fill out an AfariaConnectCmd structure defining the channel to run during the session and the application launching Afaria:

Example

```
AfariaConnectCmdPtr pAfariaCmd = 0;

pAfariaCmd = (AfariaConnectCmdPtr)MemPtrNew( sizeof(AfariaConnectCmd) );
if( pAfariaCmd )
{
     MemPtrSetOwner( pAfariaCmd, 0 );
     pAfariaCmd->callerType = 'appl';
     pAfariaCmd->callerCreator = kAppCreator;
     pAfariaCmd->wPort = 3007;
     StrCopy( pAfariaCmd->szTransmitterAddr, szTransmitterAddress );
     StrCopy( pAfariaCmd->szUserName, szUserName );
     StrCopy( pAfariaCmd->szPassword, szPassword );
}
```

With this information, the application can send the `SysLaunchAPIConnectCmd` launch code to Afaria:

```
SysUIAppSwitch( cardNo, dbId, SysAppLaunchAPIConnectCmd, pAfariaCmd );
```

This launch code is sent by the launching application to Afaria and specifies the channel you wish to execute during the session. The parameter block with this launch code is the `AfariaConnectCmd` structure that was filled out earlier.

When this launch code is received, Afaria will save the callerType and callerCreator information, then run the session. When the session has completed, Afaria will launch the calling application with `SysAppLaunchAPIReturn` launch code (see below).

Responding to Afaria return launch code

Afaria sends the `SysAppLaunchAPIReturn` launch code back to the calling application once the session is complete. The parameter block will contain error information. The calling application can proceed in any manner that it finds necessary. A typical use would be to open the form from which Afaria was launched.

Example

```
UInt32 PilotMain(UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags)
{
  Err     error             = 0;
  Err*    pUpdateErr        = errNone;

  switch( cmd )
  {
    case sysAppLaunchCmdNormalLaunch:
    {
      // Normal launch
      break;
    }

    case SysAppLaunchAPIReturn:
    {
      pUpdateErr = reinterpret_cast<Err*>( cmdPBP );
      // Retrieve error information and continue
      // as desired.
      break;
    }
```

This launch code must be handled by the calling application if the caller is to be launched when the user is done viewing the document. In most cases, this launch code can be handled the same as the regular `sysAppLaunchCmdNormalLaunch` is handled.

Afaria Palm launch codes

The following excerpt shows the two launch commands about which the Palm Client needs to know.

Example

```
#define SysAppLaunchAPIConnectCmd          35500
// This launch code tells Afaria to run a session. The
// parameter block that comes with this launch code is a
// 'AfariaConnectCmd' structure containing the channel name,
// transmitter address, transmitter port, user name, and
// password.

#define SysAppLaunchAPIReturn              35501
// This launch code tells the caller that Afaria has
// closed and is returning control to whoever has
// launched it.
```

AfariaConnectCmd structure

Example

```
typedef struct
{
  char    szTransmitterAddr[64];
  UInt16  wPort;
  char    szChannelName[64];
  char    szUserName[MAX_TITLE];
  char    szPassword[MAX_TITLE];
  char    szFormTitle[MAX_TITLE];
  UInt32  callerType;
  UInt32  callerCreator;
} AfariaConnectCmd;
typedef AfariaConnectCmd* AfariaConnectCmdPtr;
```

Table 1.

Code	Description
`szTransmitterAddr`	Null-terminated string containing the address of the Afaria Server
`wPort`	IP port on which the Afaria Server is listening
`szChannelName`	Null-terminated string containing the channel name to run during the session

Table 1.

`szUserName`	Null-terminated string containing the username for channel authentication and assignments
`szPassword`	Null-terminated string containing the password for channel authentication and assignments
`callerType`	Calling the application's database type
`callerCreator`	Calling application's database creator

Afaria API error codes

You may see the following error codes:

```
// Channel not published
#define XE_API_CHANNEL_NOTPUBLISHED (6101)

// Channel is unknown
#define XE_API_CHANNEL_UNKNOWN      (6102)
```

Symbian Client API

The APIs for Symbian Clients are defined through the CAfariaClient class. The product image includes documentation for the class and a sample application.

On your product image:

- Sample application – \Samples\Symbian
- Client API documentation – \Samples\Symbian\Docs\HTML, open index.html to start

The application displays the last session log to the log window, configures the client with enough settings to connect to an Afaria server and run a session, and writes session progress to the log window. The documentation describes classes and data structures, their associated function and variable members, and header files.