[MS-COMEV]:
Component Object Model Plus (COM+) Event System Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
07/20/2007	0.1	Major	MCPP Milestone 5 Initial Availability
09/28/2007	0.2	Minor	Updated the technical content.
10/23/2007	0.3	Minor	Updated the technical content.
11/30/2007	0.3.1	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
01/25/2008	0.3.2	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

This document specifies the behavior of the Component Object Model Plus (COM+) Event System Protocol.

The COM+ Event System Protocol is a Microsoft-proprietary protocol that exposes DCOM interfaces for storing and managing configuration data for **publishers** of **events** and their respective **subscribers** on remote computers. This protocol also specifies how to get specific information about a publisher and its subscribers.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Activation
Client (1)
CLSID
Computer Name
Globally Unique Identifier (GUID)
Interface Pointer
Object
Object Class
Opnum
Remote Procedure Call (RPC)
Security Identifier (SID)
Security Principal
Server (1)
Universally Unique Identifier (UUID)

The following terms are specific to this document:

Conglomeration: A collection of **event classes** and **subscriptions** together with independent configuration data that is conceptually shared by the both the **event classes** and **subscriptions**. A conglomeration is identified by a **conglomeration identifier**.

Conglomeration Identifier: A **GUID** that identifies a **conglomeration**.

Event Class: A collection of historical data grouped together using criteria specified by the publishing application.

Event Interface: A collection of **event methods**. An **event class** contains one or more **event interfaces**.

Event Method: A method called by the **publisher-subscriber framework** when the **publisher** application generates an **event**.

Events: Historical data that an application exposes that may be of interest to other applications.

Filtering Criteria: A set of rules specified by a **subscriber** as part of a **subscription** to define the type of historical data it wants to receive.

Partition: A container for **conglomerations**. A partition is identified by a **partition identifier**.

Partition Identifier: A **GUID** that identifies a partition

Publisher: An application that needs to publish historical data that may be of interest to other applications.

Publisher-Subscriber Framework: An application framework that allows applications to expose historical data to other applications interested in receiving the historical data.

Subscriber: An application that needs to receive historical data published by another application.

Subscription: A registration performed by a **subscriber** to specify a requirement to receive historical data.

Type Library: A type collection which defines an **event class** in terms of its **event interfaces**. A type library is specified by using a **type library identifier**.

Type Library Identifier: A **GUID** that identifies a **type library**.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[C706] The Open Group, "DCE 1.1: Remote Procedure Call", C706, August 1997, http://www.opengroup.org/public/pubs/catalog/c706.htm

[MS-DCOM] Microsoft Corporation, "Distributed Component Object Model (DCOM) Remote Protocol Specification", March 2007.

[MS-DTYP] Microsoft Corporation, "Windows Data Types", January 2007.

[MS-ERREF] Microsoft Corporation, "Windows Error Codes", January 2007.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-OAUT] Microsoft Corporation, "OLE Automation Protocol Specification", March 2007.

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions", January 2007.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[RFC4122] Leach, P., Mealling, M., and Salz, R., "A Universally Unique Identifier (UUID) URN Namespace", RFC 4122, July 2005, http://www.ietf.org/rfc/rfc4122.txt

[RFC4234] Crocker, D., Ed. and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005, http://www.ietf.org/rfc/rfc4234.txt

1.2.2 Informative References

[MS-COMA] Microsoft Corporation, "Component Object Model Plus (COM+) Remote Administration Protocol Specification", September 2007.

[MSDN-COM] Microsoft Corporation, "Component Object Model", http://msdn2.microsoft.com/en-us/library/aa286559.aspx

[MSDN-COM+] Microsoft Corporation, "COM+ (Component Services)", http://msdn2.microsoft.com/en-us/library/ms685978.aspx

[MSDN-COM+ Events] Microsoft Corporation, "COM+ Events", http://msdn2.microsoft.com/en-us/library/ms679237.aspx

[MSDN-COMEventFiltering] Microsoft Corporation, "Filtering Events in COM+", http://msdn2.microsoft.com/en-us/library/ms687082.aspx

[MSDN-COMMonikers] Microsoft Corporation, "COM Monikers", http://msdn2.microsoft.com/en-us/library/ms691261.aspx

[MSDN-ITypeLib] Microsoft Corporation, "ITypeLib", http://msdn2.microsoft.com/en-us/library/ms890643.aspx

[MSDN-MIDL] Microsoft Corporation, "Microsoft Interface Definition Language (MIDL)", http://msdn2.microsoft.com/en-us/library/ms950375.aspx

1.3 Protocol Overview (Synopsis)

1.3.1 Background

A **publisher-subscriber framework** allows applications to publish historical information that may be of interest to other applications. The applications publishing the information are called publishers, while the applications subscribing to the information are called subscribers. A publisher may specify this information in discrete sets called events. Similarly, a subscriber may subscribe to an event by creating a **subscription** for it.

1.3.2 Component Object Model Plus (COM+) Event System Protocol

The COM+ Event System Protocol provides a way to manage events and their respective subscriptions on a remote machine. The protocol is exposed as a set of DCOM [MS-DCOM] interfaces.

Using the protocol a publisher can publish, update or delete an event on a remote machine. Similarly a subscriber can use the protocol to create a subscription for an event on a remote machine. It can also modify, query, or delete subscriptions for an event on the remote machine.

A subscriber can specify that it wishes to receive a specific type of event or a collection of events. This is defined by specifying **filtering criteria**.

1.4 Relationship to Other Protocols

The COM+ Event System Protocol uses DCOM [MS-DCOM] to communicate over the wire and authenticate all requests issued against the infrastructure. Along with DCOM, this protocol also uses the OLE Automation Protocol [MS-OAUT] by using the **IDispatch** interface BSTR and VARIANT types.

The Component Object Model Plus (COM+) Remote Administration Protocol [MS-COMA] can be used to perform the registration of **type libraries** for **event classes** and subscriber DCOM components used by the COM+ Event System Protocol. It can also be used to discover subscriber DCOM components registered on the **server** to create subscriptions.

1.5 Prerequisites/Preconditions

This protocol assumes that the **client** is in a possession of valid credentials recognized by the server accepting the client requests.

1.6 Applicability Statement

The COM+ Event System Protocol is applicable to managing a store for publisher/subscriber events and subscriptions for scenarios where scalability requirements are minimal. It is not intended for scenarios where the type of events and their subscribers are more than 100. Also, the protocol is intended for scenarios where access to the event store resulting from adding, reading, updating, and deleting subscriptions and events are on the order of once every few minutes.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- **Protocol Versions**: This protocol has multiple interfaces, supported by different versions of the server. The client of this protocol can determine the version of the server by calling **GetVersion** method (section 3.1.4.10.1).

1.8 Vendor-Extensible Fields

This protocol uses HRESULTs as defined in [MS-ERREF]. Vendors are free to choose their own values for this field, as long as the C bit (0x20000000) is set, indicating it is a customer code.

1.9 Standards Assignments

The following is a table of well-known GUIDs (generated using the mechanism specified in [C706] section A.2.5) in this protocol.

Parameter	Value
CLSID for EventSystem	{4E14FBA2-2E22-11D1-9964-00C04FBBB345}
CLSID for EventClass	{cdbec9c0-7a68-11d1-88f9-0080c7d771bf}
CLSID for Subscription	{7542e960-79c7-11d1-88f9-0080c7d771bf}

2 Messages

The following sections specify how COM+ Event System Protocol messages are transported and common data types.

2.1 Transport

All the protocol messages MUST be transported using DCOM [MS-DCOM]. The protocol uses the dynamic endpoints allocated and managed by the DCOM infrastructure.

2.2 Common Data Types

In addition to **RPC** base types and definitions specified in [C706] and [MS-DTYP], additional data types are defined in the following sections.

2.2.1 Query Strings

The query string in the protocol is used for querying for subscriptions, querying for events, and specifying filtering criteria. The following is the ABNF [RFC4234] syntax for the protocol query string:

```
QUERY  = "ALL" / (OREXPRESSION)
OREXPRESSION =  (ANDEXPRESSION OREXPRTAIL) / (ANDEXPRESSION)
OREXPRTAIL =  (OROPERATOR ANDEXPRESSION OREXPRTAIL) / (OROPERATOR ANDEXPRESSION)
ANDEXPRESSION =  (UNARYEXPRESSION ANDEXPRTAIL) / (UNARYEXPRESSION)
ANDEXPRTAIL =  (ANDOPERATOR UNARYEXPRESSION ANDEXPRTAIL) / (ANDOPERATOR UNARYEXPRESSION)
ANDOPERATOR = "&" / "AND"
OROPERATOR =  "|" / "OR"
UNARYEXPRESSION =(NOTOPERATOR UNARYEXPRESSION) / (COMPARISONEXPRESSION)
NOTOPERATOR =  "!" / "~" / "NOT"
COMPARISONEXPRESSION = (COLUMNID COMPARISONOPERATOR COMPERAND) / ("("OREXPRESSION")")
COMPARISONOPERATOR =  "=" / "==" / "!=" / "~=" / "<>"
COMPERAND =  (CONSTANT) / (OPENPAREN CHOICE CLOSEPAREN)
CHOICE =  (CONSTANT MORECHOICES) / (CONSTANT)
MORECHOICES =  (ANDOROPERATOR CONSTANT MORECHOICES)  / (ANDOROPERATOR CONSTANT)
ANDOROPERATOR =  (ANDOPERATOR) / (OROPERATOR)
CONSTANT =  (SINGLEQUOTE STRINGVALUE SINGLEQUOTE)/ (DQUOTE STRINGVALUE DQUOTE) /
  (OPENCURLY UUID CLOSECURLY) / (INTEGERVALUE) / "TRUE" / "FALSE" / "NULL"
STRINGVALUE= 1*ALPHA
INTEGERVALUE = ["-" / "+"] 1*DIGIT*
COLUMNID = KNOWNCOLUMNID \ 1*ALPHA
KNOWNCOLUMNID = "EVENTCLASSID" / "EVENTCLASSNAME"  / "OWNERSID"  /
  "FIRINGINTERFACEIID"  /  "CUSTOMCONFIGCLASSID"  / "DESCRIPTION"  / "TYPELIB"  /
  "MULTIINTERFACEPUBLISHERFILTERCLSID"  / "ALLOWINPROCACTIVATION"  / "FIREINPARALLEL"  /
  "EVENTCLASSPARTITIONID"  / "EVENTCLASSAPPLICATIONID"  / "SUBSCRIPTIONID"  /
  "SUBSCRIPTIONNAME"  / "PUBLISHERID"  / "PERUSER"  / "OWNERSID"  / "ENABLED"  /
  "MACHINENAME"  / "INTERFACEID"  / "FILTERCRITERIA"  / "SUBSCRIBERMONIKER"  /
  "SUBSCRIBERPARTITIONID"  / "SUBSCRIBERAPPLICATIONID"
OPENPAREN = "("
CLOSEPAREN = ")"
SINGLEQUOTE = "'"
OPENCURLY = "{"
CLOSECURLY = "}"
```

DIGIT, HEXDIGIT, and ALPHA are as specified in [RFC4234] Appendix B.

UUID represents the string form of a UUID as specified in [RFC4122] section 3.

Each KNOWNCOLUMNID maps to a property of an event class or a subscription property. These are specified in section 3.1.1.1 and section 3.1.1.2.

2.2.2 Application-specific Properties

The protocol allows applications to associate custom properties with its **objects**. Each property is identified by means of a unique name and a value.

2.2.2.1 Property Names

The following is the ABNF [RFC4234] syntax for these names.

```
APPPROPERTYNAME = 1*255ALPHA
```

2.2.2.2 Property Value Types

These are of VARIANT type as specified in [MS-OAUT] section 2.2.13. The VARIANT type MUST be one of the following: VT_BSTR, VT_I4, VT_I8, or VT_I2, as specified in [MS-OAUT] section 2.2.4.

2.2.3 Curly-Braced GUID Strings

This type is a string representation of the **GUID** type, as specified in [MS-DTYP] section 2.3.4. The following is the ABNF [RFC4234] syntax for this representation:

```
CurlyBraceGuidString  =  "{" UUID "}"
```

UUID represents the string form of a UUID as specified in [RFC4122] section 3.

2.2.4 Entity Name String

The following is the ABNF [RFC4234] syntax for these names.

```
APPPROPERTYNAME = 1*255ALPHA
```

3 Protocol Details

The following sections specify details of COM+ Event System Protocol, including abstract data models, message processing events and sequencing rules.

The client application initiates the conversation with the server by performing DCOM **activation** (as specified in [MS-DCOM] section 3.1.4.1.1) of one of the CLSIDs specified in section 1.9. After getting the **interface pointer** to the DCOM object as a result of the activation, the client application works with the object by making calls on the DCOM interface it supports. Once done, the client application releases the interface pointer.

3.1 Server Details

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

3.1.1.1 Event Classes

A collection of **event interfaces** is grouped into an event class. An event class manifests as a DCOM object which supports each of the event interfaces which are part of the event class. This is known as the event class object. The publisher application publishes events by activating the event class object and calling **event methods** on it to publish events.

The server maintains a table of event classes. Each event class has the following properties:

EventClassName: An application-specific name for the event class. The KNOWNCOLUMNID for this property is "EVENTCLASSNAME".

EventClassID: The DCOM CLSID for the event class object. The KNOWNCOLUMNID for this property is "EVENTCLASSID".

OwnerSID: An application-specific identity of the **security principal** owning the event class. The KNOWNCOLUMNID for this property is "OWNERSID".

FiringInterfaceID: An application-specific UUID identifying the event interface. The KNOWNCOLUMNID for this property is "FIRINGINTERFACEID".

Description: An application-specific description for the event class. The KNOWNCOLUMNID for this property is "DESCRIPTION".

Typelib: A **type library identifier** for the type library containing the event class. The KNOWNCOLUMNID for this property is "TYPELIB".<1>

PublisherID: An application-specific UUID that uniquely identifies the publisher application. The KNOWNCOLUMNID for this property is "PUBLISHERID".

MultiInterfacePublisherFilterCLSID: The publisher application can choose to filter subscribers for the event class. The publisher application uses a DCOM component for this purpose. It uses this property to specify the CLSID of this component. This DCOM component is given the opportunity to filter on subscribers when an event gets fired.<2>

AllowInprocActivation: An application-specific Boolean value for controlling the type of activation for the subscriber application DCOM component. A value of TRUE indicates that the subscriber application DCOM component wants to be activated in the publisher application. A value of FALSE indicates that it does not.

FireInParallel: An application-specific Boolean value for controlling the way to fire the events for delivery to the subscribers. A value of FALSE indicates that the publisher will deliver the events to each subscriber one at a time in any given order. A value of TRUE indicates that each subscriber will be notified in parallel.

EventClassPartitionID: The UUID of the **partition** of the event class object. This is used in addition to EventClassID and EventClassApplicationID. The KNOWNCOLUMNID for this property is "EVENTCLASSPARTITIONID".<3>

EventClassApplicationID: The UUID of the **conglomeration** of the event class object. This is used in addition to EventClassID and EventClassPartitionID. The KNOWNCOLUMNID for this property is "EVENTCLASSAPPLICATIONID".<4>

3.1.1.2 Subscriptions

The server also maintains a table of subscriptions. Like event classes, subscriptions have a set of properties. Some of the properties can also be specified as part of the query (as specified in section 2.2.1). The following properties are specific to a subscription.

SubscriptionID: A UUID that uniquely identifies the subscription. The KNOWNCOLUMNID for this property is "SUBSCRIPTIONID".

SubscriptionName: An application-specific name for the subscriber. The COLUMND for this property is "SUBSCRIPTIONNAME".

PublisherID: The UUID of the publisher for which the subscriber application is to receive events. The publisher identity is defined on the event class by specifying its publisher identity property. The KNOWNCOLUMNID for this property is "PUBLISHERID".

EventClassID: The CLSID of the event class for which the subscription is created. The KNOWNCOLUMNID for this property is "EVENTCLASSID".

MethodName: The name of the event method for the specific event interface defined for the specific event class for which the application is creating a subscription.

SubscriberCLSID: The CLSID for the subscriber application's DCOM object which can be activated and then called by the publisher application once the event occurs. This MUST be mutually exclusive with the SubscriberInterface property. The KNOWNCOLUMNID for this property is "SUBSCRIBERCLSID".

SubscriberInterface: The DCOM object interface pointer for the subscriber application that will receive the notification as a method call when the event occurs. This MUST be mutually exclusive with the SubscriberCLSID property.

PerUser: A Boolean indicating whether or not the subscription is specific to a user logon session. The KNOWNCOLUMNID for this property is "PERUSER".

OwnerSID: The owner security identity for the subscription. The KNOWNCOLUMNID for this property is "OWNERSID".

Enabled: A Boolean value specifying whether the subscription is enabled or disabled. If a subscription is enabled, the value of this property will be TRUE and the subscribing application

will receive a notification when the publisher fires an event. If the subscription is disabled, the value of this property will be FALSE and the subscribing application will not receive any notification when the publisher application fires the event. The KNOWNCOLUMNID for this property is "ENABLED".

Description: An application-specific description for the subscription. The KNOWNCOLUMNID for this property is "DESCRIPTION".

MachineName: The **computer name** of the server machine where the subscriber application component resides. The KNOWNCOLUMNID for this property is "MACHINENAME".

InterfaceID: The UUID identifying the event interface for the event class for which the subscriber is going to receive an event. The KNOWNCOLUMNID for this property is "INTERFACEID".

FilterCriteria: The filtering criteria for the subscription. The KNOWNCOLUMNID for this property is "FILTERCRITERIA".<5>

SubscriberMoniker: A string identifying the subscriber component. This is specified in place of either the SubscriberCLSID or the SubscriberInterface property of the subscription. The KNOWNCOLUMNID for this property is "SUBSCRIBERMONIKER".

EventClassPartitionID: The UUID of the partition of the event class. The KNOWNCOLUMNID for this property is "EVENTCLASSPARTITIONID".

EventClassApplicationID: The UUID of the conglomeration of the event class. The KNOWNCOLUMNID for this property is "EVENTCLASSAPPLICATIONID".

SubscriberPartitionID: The UUID of the partition of the subscriber. It is used in addition to the SubscriberCLSID and SubscriberApplicationID properties to uniquely identify the subscriber component. The KNOWNCOLUMNID for this property is "SUBSCRIBERPARTITIONID".

SubscriberApplicationID: The UUID of the conglomeration of the subscriber. It is used in addition to the SubscriberCLSID and SubscriberPartitionID properties to uniquely identify the subscriber component. The KNOWNCOLUMNID for this property is "SUBSCRIBERAPPLICATIONID".

PublisherProperties: A set of application-specific properties associated with the subscription pertaining to the publisher, as specified in section 2.2.2.

SubscriberProperties: A set of application-specific properties associated with the subscription pertaining to the subscriber, as specified in section 2.2.2.

3.1.2 Timers

None.

3.1.3 Initialization

When the Component Object Model Plus (COM+) Event System Protocol server starts up, the server MUST begin listening for DCOM activation (as specified in [MS-DCOM] section 3.1.4.1.1) for the CLSIDs specified in section 1.9.

3.1.4 Message Processing Events and Sequencing Rules

3.1.4.1 IEventSystem

The **IEventSystem** interface provides methods to create, query, delete, and update event classes and subscriptions. The interface inherits Opnums 0 to 6 from **IDispatch** as specified in [MS-OAUT] section 3.1.4. The version for this interface is 0.0.

To receive incoming remote calls for this interface, the server MUST implement a DCOM **object class** with the CLSID CLSID_EventSystem (see section 1.9) using the UUID {4E14FB9F-2E22-11D1-9964-00C04FBBB345} for this interface.

The interface includes the following methods beyond those in **IDispatch**.

Methods in RPC Opnum Order

Method	Description
Query (section 3.1.4.1.1)	Queries for a collection of event classes or subscriptions based on a query string. Opnum: 7
Store (section 3.1.4.1.2)	Stores an event class or subscription. Opnum: 8
Remove (section 3.1.4.1.3)	Removes a collection of event classes or subscriptions based on a query. Opnum: 9
get_EventObjectChangeEventClassID (section 3.1.4.1.4)	Returns the CLSID for the event class that notifies when a subscription or an event class has changed. Opnum: 10
QueryS (section 3.1.4.1.5)	Queries for a single event class or subscription based on a query string. Opnum: 11
RemoveS (section 3.1.4.1.6)	Removes a single event class or subscription based on a query. Opnum: 12

3.1.4.1.1 Query (Opnum 7)

The **Query** method is called by a client to query a collection for a collection of event classes or subscriptions.

```
[id(1), helpstring("method Query")]
  HRESULT Query(
  [in] BSTR progID,
  [in] BSTR queryCriteria,
  [out] int* errorIndex,
  [out, retval] IUnknown** ppInterface
);
```

progID: A string identifying the type of collection. The value MUST be one of the following:

Value	Meaning
"EventSystem.EventClassCollection"	The store for event classes (as specified in section 3.1.1.1).
"EventSystem.EventSubscriptionCollection"	The store for subscriptions (as specified in section 3.1.1.2).

queryCriteria: The actual query string. The syntax for this string MUST conform to section 2.2.1

errorIndex: The zero-based character index in the *queryCriteria* parameter where an error has occurred. This can occur if the syntax of the query string is incorrect, in which case *errorIndex* specifies the index at which the problematic syntax is present in the *queryCriteria* parameter.

ppInterface: If the method returns a success **HRESULT**, this MUST contain an interface pointer representing the collection of the event classes or subscriptions based on the criteria specified in the *queryCriteria* parameter.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST use the *progID* parameter value to determine the store against which the query needs to be executed and validate the query. If the specified collection is not valid or the specified query is not valid, the server MUST fail the call. Otherwise, the server MUST attempt to use the *queryCriteria* parameter to retrieve a collection of event classes or subscriptions based on the value of the *progID* parameter and fail the call if it cannot. Each of the objects in the collection MUST be wrapped by a DCOM object that MUST support the **IEventClass** and **IEventClass2** interfaces and MAY<6> support the **IEventClass3** interface if the object is an event class. It MUST support **IEventSubscription** and **IEventSubscription2**, and MAY<7> support **IEventSubscription3** if it is a subscription object. These DCOM objects MUST be encapsulated into a collection-based DCOM object supporting the **IEventObjectCollection** interface. This object MUST be returned through the *ppInterface* parameter.

3.1.4.1.2 Store (Opnum 8)

The **Store** method is called by a client to store either an event class or a subscription.

```
[id(2), helpstring("method Store")]
  HRESULT Store(
  [in] BSTR ProgID,
  [in] IUnknown* pInterface
);
```

ProgID: A string that uniquely identifies what sort of an object the client is trying to store. This MUST be one of the following values:

Value	Meaning
"EventSystem.EventClass"	The store for event classes (as specified in section 3.1.1.1).
"EventSystem.EventSubscription"	The store for subscriptions (as specified in section 3.1.1.2).

pInterface: An interface pointer to a DCOM object that was created by performing DCOM activation on the server by the client by using either the CLSID CLSID_EventClass as specified in section 1.9 which represents the Event Class, or CLSID_Subscription as specified in section 1.9 which represents the subscriber.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that all the required properties of the event class or the subscription are specified and are correct. If not, it MUST fail the call. Otherwise, it MUST go through the state of the DCOM object passed in as part of the *pInterface* parameter. If the DCOM object is an event class it MUST verify that the EventClassID, EventClassName and Typelib properties are set, and fail the call if not. Otherwise, if the DCOM object is a subscription, it MUST verify that the SubscriptionID and EventClassID properties are set, and either SubscriberCLSID or the SubscriberInterface properties are set and fail the call if not. Otherwise, it MUST take the individual properties of the event class or the subscription based on the type of store requested and MUST attempt to store these into its internal store, and fail the call if it cannot. If there is already an entry in the store for the particular object being represented by the DCOM object instance, the server MUST update its internal store entry with the new values of the subscription or the event class as specified in the DCOM object instance.

3.1.4.1.3 Remove (Opnum 9)

The **Remove** method is called by a client to remove a collection of event classes or subscriptions by a criteria represented by a query string in the *queryCriteria* parameter.

```
[id(3), helpstring("method Remove")]
  HRESULT Remove(
  [in] BSTR progID,
  [in] BSTR queryCriteria,
  [out] int* errorIndex
);
```

progID: A string that uniquely identifies the type of collection. The value MUST be one of the following

Value	Meaning
"EventSystem.EventClassCollection"	The store for event classes (as specified in section 3.1.1.1).
"EventSystem.EventSubscriptionCollection"	The store for subscriptions (as specified in section 3.1.1.2).

queryCriteria: The actual query string. The syntax for this string MUST conform to section 2.2.1.

errorIndex: The zero-based character index in the *queryCriteria* parameter where an error has occurred. This can occur if the syntax of the query string is incorrect, in which case the *errorIndex* specifies the index at which the problematic syntax is present in the *queryCriteria* parameter.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST use the *progID* parameter value to determine the store against which the query needs to be executed and validate the query. If the specified collection is not valid or the specified query is not valid, the server MUST fail the call. If they are valid, the server MUST use the *queryCriteria* parameter to determine the event classes or subscriptions that need to be removed. If none of the entries in the internal store matched the query criteria, the server MUST fail the call. Otherwise, it MUST attempt to remove them from its internal collection, and fail the call if it cannot.

3.1.4.1.4 get_EventObjectChangeEventClassID (Opnum 10)

The **get_EventObjectChangeEventClassID** method extracts the server-specific EventClassID for server-specific event class or subscription change notifications.

```
[id(4), propget, helpstring("method get EventObjectChangeEventClassID")]
  HRESULT EventObjectChangeEventClassID(
  [out, retval] BSTR* pbstrEventClassID
);
```

pbstrEventClassID: If the method call returns a success HRESULT, this MUST contain the returned unique identifier representing the EventClassID for the server specific EventClass/Subscription change notifications. This MUST be a GUID specified as a string as specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, a server SHOULD return the EventClassID of an event class supporting notifications of changes to the server's event class store and subscriber store. The server MAY instead fail the method call if it does not support such an event class.

3.1.4.1.5 QueryS (Opnum 11)

The **QueryS** method is called by the client to query a specific event class or subscription.

```
[id(5), helpstring("method QueryS")]
  HRESULT QueryS(
  [in] BSTR progID,
  [in] BSTR queryCriteria,
  [out, retval] IUnknown** ppInterface
);
```

progID: A string that uniquely identifies the type of collection. The value MUST be one of the following:

Value	Meaning
"EventSystem.EventClassCollection"	The store for event classes (as specified in section 3.1.1.1).
"EventSystem.EventSubscriptionCollection"	The store for subscriptions (as specified in section 3.1.1.2).

queryCriteria: The actual query string. The syntax for this string MUST conform to section 2.2.1.

ppInterface: If the method returns success, this MUST contain an interface pointer representing the collection of the event classes or subscriptions based on the criteria specified in the *queryCriteria* parameter.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST use the *progID* parameter value to determine the store against which the query needs to be executed and validate the query. If the specified collection is not valid or the specified query is not valid, the server MUST fail the call. Otherwise, the server MUST use the query criteria to attempt to return the first object that matches the criteria, and fail the call if it cannot. The object MUST be a DCOM object that MUST support the **IEventClass** and **IEventClass2** interfaces and MAY<8> support **IEventClass3** interface if the object is an event class. It MUST support **IEventSubscription**, **IEventSubscription2** and MAY<9> support **IEventSubscription3** if it is a subscription object. This object MUST be stored in a collection-based DCOM object supporting **IEventObjectCollection** which MUST be returned through the *ppInterface* parameter.

3.1.4.1.6 RemoveS (Opnum 12)

The **RemoveS** method is called by the client to remove an event class or subscription.

```
[id(6), helpstring("method RemoveS")]
  HRESULT RemoveS(
  [in] BSTR progID,
  [in] BSTR queryCriteria
);
```

progID: A string that uniquely identifies the type of collection. The value MUST be one of the following:

Value	Meaning
"EventSystem.EventClassCollection"	The store for event classes (as specified in section 3.1.1.1).
"EventSystem.EventSubscriptionCollection"	The store for subscriptions (as specified in section 3.1.1.2).

queryCriteria: The actual query string. The syntax for this string MUST conform to section 2.2.1.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST use the *progID* parameter value to determine the store against which the query needs to be executed, and validate the query. If the specified collection is not valid or the specified query is not valid, the server MUST fail the call. If they are valid, the server MUST use the *queryCriteria* value to determine the event classes or subscriptions that need to be removed and it MUST remove any one object that meets the criteria.

3.1.4.2 IEventClass

The **IEventClass** interface provides methods that are used by the client to manipulate an event class on the server. The interface inherits Opnums 0 to 6 from **IDispatch** as specified in [MS-OAUT] section 3.1.4. The version for this interface is 0.0.

To receive incoming remote calls for this interface, the server MUST implement an DCOM object class with the CLSID CLSID_EventClass (see section 1.9) using the UUID {fb2b72a0-7a68-11d1-88f9-0080c7d771bf} for this interface.

The interface includes the following methods beyond those in **IDispatch**.

Methods in RPC Opnum Order

Method	Description
get_EventClassID (section 3.1.4.2.1)	Gets the EventClassID property for the event class. Opnum: 7
put_EventClassID (section 3.1.4.2.2)	Sets the EventClassID property for the event class. Opnum: 8
get_EventClassName (section 3.1.4.2.3)	Gets the EventClassName property of the event class. Opnum: 9
put_EventClassName (section 3.1.4.2.4)	Sets the EventClassName property of the event class. Opnum: 10
get_OwnerSID (section 3.1.4.2.5)	Gets the OwnerSID property of the event class. Opnum: 11
put_OwnerSID (section 3.1.4.2.6)	Sets the OwnerSID property of the event class. Opnum: 12
get_FiringInterfaceID (section 3.1.4.2.7)	Gets the FiringInterfaceID property for the event class. Opnum: 13
put_FiringInterfaceID (section 3.1.4.2.8)	Sets the FiringInterfaceID property for the event class. Opnum: 14
get_Description (section 3.1.4.2.9)	Gets the Description property for the event class. Opnum: 15
put_Description (section 3.1.4.2.10)	Sets the Description property for the event class. Opnum: 16
Opnum17NotUsedOnWire	Reserved for local use. Opnum: 17
Opnum18NotUsedOnWire	Reserved for local use. Opnum: 18
get_TypeLib (section 3.1.4.2.11)	Gets the Typelib property of the event class. Opnum: 19

Method	Description
put_TypeLib (section 3.1.4.2.12)	Sets the Typelib property of the event class. Opnum: 20

In the table above, the term "Reserved for local use" means that the client MUST NOT send the **opnum**, and the server behavior is undefined<10> since it does not affect interoperability.

3.1.4.2.1 get_EventClassID (Opnum 7)

The **get_EventClassID** method is used to get the EventClassID property of the event class.

```
[propget, id(1), helpstring("property EventClassID")]
  HRESULT get_EventClassID(
  [out, retval] BSTR* pbstrEventClassID
);
```

pbstrEventClassID: If the method returns a success HRESULT, it MUST contain the value of the EventClassID property of the event class, as specified in section 3.1.1.1. The value MUST conform to the format specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the EventClassID property.

3.1.4.2.2 put_EventClassID (Opnum 8)

The **put_EventClassID** method sets the EventClassID property of the event class.

```
[propput, id(1), helpstring("property EventClassID")]
  HRESULT put_EventClassID(
  [in] BSTR bstrEventClassID
);
```

bstrEventClassID: The EventClassID property of the event class, as specified in section 3.1.1.1. The value MUST conform to the format specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the EventClassID property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.2.3 get_EventClassName (Opnum 9)

The **get_EventClassName** method gets the EventClassName property of the event class.

```
[propget, id(2), helpstring("property EventClassName")]
  HRESULT get_EventClassName(
  [out, retval] BSTR* pbstrEventClassName
);
```

pbstrEventClassName: If the method returns a success HRESULT, this MUST contain the value of the EventClassName property of the event class, as specified in section 3.1.1.1. The value MUST conform to the format specified in section 2.2.4.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the EventClassName property.

3.1.4.2.4 put_EventClassName (Opnum 10)

The **put_EventClassName** method sets the EventClassName property of the event class.

```
[propput, id(2), helpstring("property EventClassName")]
  HRESULT put_EventClassName(
  [in] BSTR bstrEventClassName
);
```

bstrEventClassName: The EventClassName property of the event class, as specified in section 3.1.1.1. The value MUST conform to the format specified in section 2.2.4.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the EventClassName property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.2.5 get_OwnerSID (Opnum 11)

The **get_OwnerSID** method gets the OwnerSID property of the event class.

```
[propget, id(3), helpstring("property OwnerSID"]
  HRESULT OwnerSID(
  [out, retval] BSTR* pbstrOwnerSID
);
```

pbstrOwnerSID: If the method returns a success HRESULT, this MUST contain the OwnerSID property of the event class, as specified in section 3.1.1.1. The value MUST be specified in the Security Descriptor Description Language specified in [MS-DTYP] section 2.5.1.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the OwnerSID property.

3.1.4.2.6 put_OwnerSID (Opnum 12)

The **put_OwnerSID** method sets the OwnerSID property of the event class.

```
[propput, id(3), helpstring("property OwnerSID"]
  HRESULT put_OwnerSID(
  [in] BSTR bstrOwnerSID
);
```

bstrOwnerSID: The OwnerSID property of the event class, as specified in section 3.1.1.1. The value MUST be specified in the Security Descriptor Description Language specified in [MS-DTYP] section 2.5.1.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the OwnerSID property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call

3.1.4.2.7 get_FiringInterfaceID (Opnum 13)

The **get_FiringInterfaceID** method gets the FiringInterfaceID property of the event class.

```
[propget, id(4), helpstring("property FiringInterfaceID")]
  HRESULT get_FiringInterfaceID(
  [out, retval] BSTR* pbstrFiringInterfaceID
);
```

pbstrFiringInterfaceID: If the method returns a success HRESULT, it MUST contain the FiringInterfaceID property of the event class, as specified in section 3.1.1.1. The value MUST conform to the format specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the FiringInterfaceID property.

3.1.4.2.8 put_FiringInterfaceID (Opnum 14)

The **put_FiringInterfaceID** method sets the FiringInterfaceID property of the event class.

```
[propput, id(4), helpstring("property FiringInterfaceID")]
  HRESULT put_FiringInterfaceID(
  [in] BSTR bstrFiringInterfaceID
```

```
);
```

bstrFiringInterfaceID: The value of the FiringInterfaceID property of the event class, as specified in section 3.1.1.1. The value MUST conform to the format specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the FiringInterfaceID property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.2.9 get_Description (Opnum 15)

The **get_Description** method gets the Description property of the event class.

```
[propget, id(5), helpstring("property Description")]
  HRESULT get_Description(
  [out, retval] BSTR* pbstrDescription
);
```

pbstrDescription: If the method returns a success HRESULT, this MUST contain the Description property of the event class, as specified in section 3.1.1.1. The string value MUST be of length less than or equal to 255.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the Description property.

3.1.4.2.10 put_Description (Opnum 16)

The **put_Description** method sets the Description property of the event class.

```
[propput, id(5), helpstring("property Description")]
  HRESULT put_Description(
  [in] BSTR bstrDescription
);
```

bstrDescription: The Description property of the event class, as specified in section 3.1.1.1. The string value MUST be of length less than or equal to 255.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the Description property,

and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call

3.1.4.2.11 get_TypeLib (Opnum 19)

The **get_TypeLib** method gets the Typelib property of the event class.

```
[propget, id(7), helpstring("property TypeLib")]
  HRESULT get_TypeLib(
  [out, retval] BSTR* pbstrTypeLib
);
```

pbstrTypeLib: If the method returns a success HRESULT, this MUST contain the Typelib property of the event class, as specified in section 3.1.1.1. The value MUST conform to the format specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the Typelib property.

3.1.4.2.12 put_TypeLib (Opnum 20)

The **put_TypeLib** method sets the Typelib property of the event class.

```
[propput, id(7), helpstring("property TypeLib")]
  HRESULT put_TypeLib(
  [in] BSTR bstrTypeLib
);
```

bstrTypeLib: The Typelib property of the event class, as specified in section 3.1.1.1. The value MUST conform to the format specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the Typelib property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.3 IEventClass2

The **IEventClass2** interface provides additional methods that are used by the client to manipulate event class properties on the server. This interface inherits Opnums 0 to Opnum 20 from **IEventClass** as specified in section 3.1.4.2. The version for this interface is 0.0.

To receive incoming remote calls for this interface, the server MUST implement an [MS-DCOM] Object Class with the CLSID CLSID_EventClass (see section 1.9) using the UUID {fb2b72a1-7a68-11d1-88f9-0080c7d771bf} for this interface.

This interface includes the following methods beyond those in **IEventClass**.

Methods in RPC Opnum Order

Method	Description
get_PublisherID (section 3.1.4.3.1)	Gets the PublisherID property of the event class. Opnum: 21
put_PublisherID (section 3.1.4.3.2)	Sets the PublisherID property of the event class. Opnum: 22
get_MultiInterfacePublisherFilterCLSID (section 3.1.4.3.3)	Gets the MultiInterfacePublisherFilterCLSID property of the event class. Opnum: 23
put_MultiInterfacePublisherFilterCLSID (section 3.1.4.3.4)	Sets the MultiInterfacePublisherFilterCLSID property of the event class. Opnum: 24
get_AllowInprocActivation (section 3.1.4.3.5)	Gets the AllowInprocActivation property of the event class. Opnum: 25
put_AllowInprocActivation (section 3.1.4.3.6)	Sets the AllowInprocActivation property of the event class. Opnum: 26
get_FireInParallel (section 3.1.4.3.7)	Gets the FireInParallel property of the event class. Opnum: 27
put_FireInParallel (section 3.1.4.3.8)	Sets the FireInParallel property of the event class. Opnum: 28

3.1.4.3.1 get_PublisherID (Opnum 21)

The **get_PublisherID** method gets the PublisherID property of the event class.

```
[id(8), propget, helpstring("property PublisherID")]
  HRESULT get_PublisherID(
  [out, retval] BSTR* pbstrPublisherID
);
```

pbstrPublisherID: If the method returns a success HRESULT, this MUST contain the PublisherID property of the event class, as specified in section 3.1.1.1. The value MUST conform to the format specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the PublisherID property.

3.1.4.3.2 put_PublisherID (Opnum 22)

The **put_PublisherID** method sets the PublisherID property of the event class.

```
[id(8), propput, helpstring("property PublisherID")]
  HRESULT put_PublisherID(
  [in] BSTR bstrPublisherID
);
```

bstrPublisherID: The PublisherID property of the event class, as specified in section 3.1.1.1. The value MUST conform to the format specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the PublisherID property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.3.3 get_MultiInterfacePublisherFilterCLSID (Opnum 23)

The **get_MultiInterfacePublisherFilterCLSID** method gets the MultiInterfacePublisherFilterCLSID property of the event class.

```
[id(9), propget, helpstring("property MultiInterfacePublisherFilterCLSID")]
  HRESULT get MultiInterfacePublisherFilterCLSID(
  [out, retval] BSTR* pbstrPubFilCLSID
);
```

pbstrPubFilCLSID: If the method returns a success HRESULT, this MUST contain the MultiInterfacePublisherFilterCLSID property of the event class, as specified in section 3.1.1.1. The value MUST conform to the format specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the MultiInterfacePublisherFilterCLSID property.

3.1.4.3.4 put_MultiInterfacePublisherFilterCLSID (Opnum 24)

The **put_MultiInterfacePublisherFilterCLSID** method sets the MultiInterfacePublisherFilterCLSID property of the event class.

```
[id(9), propput, helpstring("property MultiInterfacePublisherFilterCLSID")]
  HRESULT put MultiInterfacePublisherFilterCLSID(
  [in] BSTR bstrPubFilCLSID
);
```

bstrPubFilCLSID: The MultiInterfacePublisherFilterCLSID property of the event class, as specified in section 3.1.1.1. The value MUST conform to the format specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the MultiInterfacePublisherFilterCLSID property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.3.5 get_AllowInprocActivation (Opnum 25)

The **get_AllowInprocActivation** method gets the AllowInprocActivation property of the event class.

```
[id(10), propget, helpstring("property AllowInprocActivation")]
  HRESULT get_AllowInprocActivation(
  [out, retval] BOOL* pfAllowInprocActivation
);
```

pfAllowInprocActivation: If the method returns a success HRESULT, this MUST contain the AllowInprocActivation property of the event class, as specified in section 3.1.1.1.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the AllowInprocActivation property.

3.1.4.3.6 put_AllowInprocActivation (Opnum 26)

The **put_AllowInprocActivation** method sets the AllowInprocActivation property of the event class.

```
[id(10), propput, helpstring("property AllowInprocActivation")]
  HRESULT put_AllowInprocActivation(
  [in] BOOL fAllowInprocActivation
);
```

fAllowInprocActivation: The value of the AllowInprocActivation property of the event class, as specified in section 3.1.1.1.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the AllowInprocActivation property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.3.7 get_FireInParallel (Opnum 27)

The **get_FireInParallel** method gets the FireInParallel property of the event class.

```
[id(11), propget, helpstring("property FireInParallel")]
  HRESULT get_FireInParallel(
  [out, retval] BOOL* pfFireInParallel
);
```

pfFireInParallel: If the method returns a success HRESULT, this MUST contain the value of the FireInParallel property of the event class, as specified in section 3.1.1.1.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the FireInParallel property.

3.1.4.3.8 put_FireInParallel (Opnum 28)

The **put_FireInParallel** method sets the value of the FireInParallel property of the event class.

```
[id(11), propput, helpstring("property FireInParallel")]
  HRESULT put_FireInParallel(
  [in] BOOL fFireInParallel
);
```

fFireInParallel: The value of the FireInParallel property of the event class, as specified in section 3.1.1.1.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the FireInParallel property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.4 IEventSubscription

The **IEventSubscription** interface provides methods to get/set the properties of a subscription. This interface inherits Opnums 0 to 6 from [MS-OAUT] IDispatch as specified in [MS-OAUT] section 3.1.4. The version for this interface is 0.0.

To receive incoming remote calls for this interface, the server MUST implement an [MS-DCOM] Object Class with the CLSID CLSID_Subscription (see section 1.9) using the UUID {4A6B0E15-2E38-11D1-9965-00C04FBBB345} for this interface.

The interface includes the following methods beyond those in **IDispatch**.

Methods in RPC Opnum Order

Method	Description
get_SubscriptionID (section 3.1.4.4.1)	Gets the SubscriptionID property of the subscription. Opnum: 7
put_SubscriptionID (section 3.1.4.4.2)	Sets the SubscriptionID property of the subscription. Opnum: 8
get_SubscriptionName (section 3.1.4.4.3)	Gets the SubscriptionName property of the subscription. Opnum: 9
put_SubscriptionName (section 3.1.4.4.4)	Sets the SubscriptionName property of the subscription. Opnum: 10
get_PublisherID (section 3.1.4.4.5)	Gets the PublisherID property of the subscription. Opnum: 11
put_PublisherID (section 3.1.4.4.6)	Sets the PublisherID property of the subscription. Opnum: 12
get_EventClassID (section 3.1.4.4.7)	Gets the EventClassID property of the subscription. Opnum: 13
put_EventClassID (section 3.1.4.4.8)	Sets the EventClassID property of the subscription. Opnum: 14
get_MethodName (section 3.1.4.4.9)	Gets the MethodName property of the subscription. Opnum: 15
put_MethodName (section 3.1.4.4.10)	Sets the MethodName property of the subscription. Opnum: 16
get_SubscriberCLSID (section 3.1.4.4.11)	Gets the SubscriberCLSID property of the subscription. Opnum: 17
put_SubscriberCLSID (section 3.1.4.4.12)	Sets the SubscriberCLSID property of the subscription. Opnum: 18
get_SubscriberInterface (section 3.1.4.4.13)	Gets the SubscriberInterface property of the subscription. Opnum: 19
put_SubscriberInterface (section 3.1.4.4.14)	Sets the SubscriberInterface property of the subscription. Opnum: 20
get_PerUser (section 3.1.4.4.15)	Gets the PerUser property of the subscription. Opnum: 21
put_PerUser (section 3.1.4.4.16)	Sets the PerUser property of the subscription. Opnum: 22
get_OwnerSID (section 3.1.4.4.17)	Gets the OwnerSID property of the subscription.

Method	Description
	Opnum: 23
put_OwnerSID (section 3.1.4.4.18)	Sets the OwnerSID property of the subscription. Opnum: 24
get_Enabled (section 3.1.4.4.19)	Gets the Enabled property of the subscription. Opnum: 25
put_Enabled (section 3.1.4.4.20)	Sets the Enabled property of the subscription. Opnum: 26
get_Description (section 3.1.4.4.21)	Gets the Description property of the subscription. Opnum: 27
put_Description (section 3.1.4.4.22)	Sets the Description property of the subscription. Opnum: 28
get_MachineName (section 3.1.4.4.23)	Gets the MachineName property of the subscription. Opnum: 29
put_MachineName (section 3.1.4.4.24)	Sets the MachineName property of the subscription. Opnum: 30
GetPublisherProperty (section 3.1.4.4.25)	Gets the application-specific publisher property for the subscription. Opnum: 31
PutPublisherProperty (section 3.1.4.4.26)	Sets the application-specific publisher property for the subscription. Opnum: 32
RemovePublisherProperty (section 3.1.4.4.27)	Removes an application-specific publisher property for the subscription. Opnum: 33
GetPublisherPropertyCollection (section 3.1.4.4.28)	Gets application-specific publisher properties collection for the subscription. Opnum: 34
GetSubscriberProperty (section 3.1.4.4.29)	Gets an application-specific subscription property for the subscription. Opnum: 35
PutSubscriberProperty (section 3.1.4.4.30)	Sets an application-specific subscription property for the subscription. Opnum: 36
RemoveSubscriberProperty (section 3.1.4.4.31)	Removes an application-specific subscription property for the subscription. Opnum: 37
GetSubscriberPropertyCollection (section 3.1.4.4.32)	Gets the application-specific subscription properties for the subscription as a collection.

Method	Description
	Opnum: 38
get_InterfaceID (section 3.1.4.4.33)	Gets the InterfaceID property for the subscription. Opnum: 39
put_InterfaceID (section 3.1.4.4.34)	Sets the InterfaceID property for the subscription. Opnum: 40

3.1.4.4.1 get_SubscriptionID (Opnum 7)

The **get_SubscriptionID** method gets the SubscriptionID property for the subscription.

```
[propget, id(1), helpstring("property SubscriptionID")]
  HRESULT get_SubscriptionID(
  [out, retval] BSTR* pbstrSubscriptionID
);
```

pbstrSubscriptionID: If the method returns a success HRESULT, this MUST contain the SubscriptionID property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the SubscriptionID property.

3.1.4.4.2 put_SubscriptionID (Opnum 8)

The **put_SubscriptionID** method sets the SubscriptionID property of the subscription.

```
[propput, id(1), helpstring("property SubscriptionID")]
  HRESULT put_SubscriptionID(
  [in] BSTR bstrSubscriptionID
);
```

bstrSubscriptionID: A UUID uniquely identifying the subscription, in the string format specified in section 2.2.3. This MUST be a UUID generated by the client, as specified in [C706] section A.2.5.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the SubscriptionID property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.4.3 get_SubscriptionName (Opnum 9)

The **get_SubscriptionName** method gets the SubscriptionName property of the subscription.

```
[propget, id(2), helpstring("property SubscriptionName")]
  HRESULT get_SubscriptionName(
  [out, retval] BSTR* pbstrSubscriptionName
);
```

pbstrSubscriptionName: If the method returns a success HRESULT, this MUST contain the SubscriptionName property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.4.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the SubscriptionName property.

3.1.4.4.4 put_SubscriptionName (Opnum 10)

The **put_SubscriptionName** method sets the SubscriptionName property of the subscription.

```
[propput, id(2), helpstring("property SubscriptionName")]
  HRESULT put_SubscriptionName(
  [in] BSTR bstrSubscriptionName
);
```

bstrSubscriptionName: The SubscriptionName property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.4.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the SubscriptionName property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.4.5 get_PublisherID (Opnum 11)

The **get_PublisherID** method gets the PublisherID property of the subscription.

```
[propget, id(3), helpstring("property PublisherID")]
  HRESULT get_PublisherID(
  [out, retval] BSTR* pbstrPublisherID
);
```

pbstrPublisherID: If the method returns a success HRESULT, this MUST contain the PublisherID property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.4.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the PublisherID property.

3.1.4.4.6 put_PublisherID (Opnum 12)

The **put_PublisherID** method sets the PublisherID property of the subscription.

```
[propput, id(3), helpstring("property PublisherID")]
  HRESULT put_PublisherID(
  [in] BSTR bstrPublisherID
);
```

bstrPublisherID: The PublisherID property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.4.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the PublisherID property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.4.7 get_EventClassID (Opnum 13)

The **get_EventClassID** method gets the EventClassID property of the subscription.

```
[propget, id(4), helpstring("property EventClassID")]
  HRESULT get_EventClassID(
  [out, retval] BSTR* pbstrEventClassID
);
```

pbstrEventClassID: If the method returns a success HRESULT, this MUST contain the EventClassID property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.4.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the EventClassID property.

3.1.4.4.8 put_EventClassID (Opnum 14)

The **put_EventClassID** method sets the EventClassID property of the subscription.

```
[propput, id(4), helpstring("property EventClassID")]
  HRESULT put_EventClassID(
  [in] BSTR bstrEventClassID
);
```

bstrEventClassID: The EventClassID property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.4.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the EventClassID property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.4.9 get_MethodName (Opnum 15)

The **get_MethodName** method gets the MethodName property of the subscription.

```
[propget, id(5), helpstring("property MethodName")]
  HRESULT get_MethodName(
  [out, retval] BSTR* pbstrMethodName
);
```

pbstrMethodName: If the method returns a success HRESULT, this MUST contain the MethodName property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.4.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the MethodName property.

3.1.4.4.10 put_MethodName (Opnum 16)

The **put_MethodName** method sets the MethodName property of the subscription.

```
[propput, id(5), helpstring("property MethodName")]
  HRESULT put_MethodName(
  [in] BSTR bstrMethodName
);
```

bstrMethodName: The MethodName property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.4.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the MethodName property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.4.11 get_SubscriberCLSID (Opnum 17)

The **get_SubscriberCLSID** method gets the SubscriberCLSID property of the subscription.

```
[propget, id(6), helpstring("property SubscriberCLSID")]
  HRESULT get_SubscriberCLSID(
  [out, retval] BSTR* pbstrSubscriberCLSID
);
```

pbstrSubscriberCLSID: If the method returns a success HRESULT, this MUST contain the SubscriberCLSID property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the SubscriberCLSID property.

3.1.4.4.12 put_SubscriberCLSID (Opnum 18)

The **put_SubscriberCLSID** method sets the SubscriberCLSID property of the subscription.

```
[propput, id(6), helpstring("property SubscriberCLSID")]
  HRESULT put_SubscriberCLSID(
  [in] BSTR bstrSubscriberCLSID
);
```

bstrSubscriberCLSID: The SubscriberCLSID property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the SubscriberCLSID property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.4.13 get_SubscriberInterface (Opnum 19)

The **get_SubscriberInterface** method gets the SubscriberInterface property of the subscription.

```
[propget, id(7), helpstring("property SubscriberInterface")]
  HRESULT get_SubscriberInterface(
  [out, retval] IUnknown** ppSubscriberInterface
);
```

ppSubscriberInterface: If the method returns a success HRESULT, this MUST contain the SubscriberInterface property of the subscription, as specified in section 3.1.1.2.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the SubscriberInterface property.

3.1.4.4.14 put_SubscriberInterface (Opnum 20)

The **put_SubscriberInterface** method sets the SubscriberInterface property of the subscription.

```
[propput, id(7), helpstring("property SubscriberInterface")]
  HRESULT put_SubscriberInterface(
  [in] IUnknown* pSubscriberInterface
);
```

pSubscriberInterface: The SubscriberInterface property of the subscription, as specified in section 3.1.1.2.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the SubscriberInterface property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call

3.1.4.4.15 get_PerUser (Opnum 21)

The **get_PerUser** method gets the PerUser property of the subscription.

```
[propget, id(8), helpstring("property PerUser")]
  HRESULT get_PerUser(
  [out, retval] BOOL* pfPerUser
);
```

pfPerUser: If the method returns a success HRESULT, this MUST contain the value of the PerUser property of the subscription, as specified in section 3.1.1.2.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the PerUser property.

3.1.4.4.16 put_PerUser (Opnum 22)

The **put_PerUser** method sets the PerUser property of the subscription.

```
[propput, id(8), helpstring("property PerUser")]
  HRESULT put_PerUser(
  [in] BOOL fPerUser
);
```

fPerUser: This is the PerUser property of the subscription, as specified in section 3.1.1.2.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the PerUser property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.4.17 get_OwnerSID (Opnum 23)

The **get_OwnerSID** method gets the OwnerSID property of the subscription.

```
[propget, id(9), helpstring("property OwnerSID")]
  HRESULT get_OwnerSID(
  [out, retval] BSTR* pbstrOwnerSID
);
```

pbstrOwnerSID: If the method returns a success HRESULT, this MUST contain the value of the OwnerSID property of the subscription, as specified in section 3.1.1.2. The value MUST be specified in the Security Descriptor Description Language specified in [MS-DTYP] section 2.5.1.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the OwnerSID property.

3.1.4.4.18 put_OwnerSID (Opnum 24)

The **put_OwnerSID** method sets the OwnerSID property of the subscription

```
[propput, id(9), helpstring("property OwnerSID")]
  HRESULT put_OwnerSID(
  [in] BSTR bstrOwnerSID
);
```

bstrOwnerSID: The OwnerSID property of the subscription, as specified in section 3.1.1.2. The value MUST be specified in the Security Descriptor Description Language specified in [MS-DTYP] section 2.5.1.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the OwnerSID property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.4.19 get_Enabled (Opnum 25)

The **get_Enabled** method gets the Enabled property of the subscription.

```
[propget, id(10), helpstring("property Enabled")]
  HRESULT get_Enabled(
  [out, retval] BOOL* pfEnabled
);
```

pfEnabled: If the method returns a success HRESULT, this MUST contain the value of the Enabled property of the subscription, as specified in section 3.1.1.2.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the Enabled property.

3.1.4.4.20 put_Enabled (Opnum 26)

The **put_Enabled** method sets the Enabled property of the subscription.

```
[propput, id(10), helpstring("property Enabled")]
  HRESULT put_Enabled(
  [in] BOOL fEnabled
);
```

fEnabled: The new value of the Enabled property of the subscription, as specified in section 3.1.1.2.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the Enabled property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.4.21 get_Description (Opnum 27)

The **get_Description** method gets the Description property of the subscription.

```
[propget, id(11), helpstring("property Description")]
  HRESULT get_Description(
  [out, retval] BSTR* pbstrDescription
);
```

pbstrDescription: If the method returns a success HRESULT, this MUST contain the value of the Description property of the subscription, as specified in section 3.1.1.2. This MUST be a string of character length less than or equal to 255.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the Description property.

3.1.4.4.22 put_Description (Opnum 28)

The **put_Description** method sets the Description property of the subscription.

```
[propput, id(11), helpstring("property Description")]
  HRESULT put_Description(
  [in] BSTR bstrDescription
);
```

bstrDescription: The Description property of the subscription, as specified in section 3.1.1.2. This MUST be a string of character length less than or equal to 255.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the Description property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.4.23 get_MachineName (Opnum 29)

The **get_MachineName** method gets the MachineName property of the subscription.

```
[propget, id(12), helpstring("property MachineName")]
  HRESULT get_MachineName(
  [out, retval] BSTR* pbstrMachineName
);
```

pbstrMachineName: If the method returns a success HRESULT, this MUST contain the value of the MachineName property of the subscription, as specified in section 3.1.1.2. This MUST be a string of character length less than or equal to 255.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the MachineName property.

3.1.4.4.24 put_MachineName (Opnum 30)

The **put_MachineName** method sets the MachineName property of the subscription.

```
[propput, id(12), helpstring("property MachineName")]
  HRESULT put_MachineName(
  [in] BSTR bstrMachineName
);
```

bstrMachineName: The MachineName property of the subscription, as specified in section 3.1.1.2. This MUST be a string of character length less than or equal to 255.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the MachineName property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.4.25 GetPublisherProperty (Opnum 31)

The **GetPublisherProperty** method gets the application-specific publisher property of the subscription. See publisher properties in section 3.1.1.2.

```
[id(13), helpstring("method GetPublisherProperty")]
  HRESULT GetPublisherProperty(
  [in] BSTR bstrPropertyName,
  [out, retval] VARIANT* propertyValue
);
```

bstrPropertyName: The application-specific name for publisher property. The format for the publisher property name MUST adhere to the format specified in section 2.2.2.1.

propertyValue: If the function returns a success HRESULT, this MUST contain the application-specific publisher property value which MUST be of the type specified in 2.2.2.2.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the *bstrPropertyName* parameter. If validation fails, the server MUST fail the call. The server MUST then check to see if the value for this

property is associated with state of the instance of the DCOM object servicing this call specific to publisher properties. The server MUST verify that the value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the property.

3.1.4.4.26 PutPublisherProperty (Opnum 32)

The **PutPublisherProperty** method sets the application-specific publisher property of the subscription. If the subscription does not already have a publisher property, this method will add it to the publisher property collection. If the same name property exists, it would be overwritten by the new value provided as part of this method. See publisher properties in section 3.1.1.2.

```
[id(14), helpstring("method PutPublisherProperty")]
  HRESULT PutPublisherProperty(
  [in] BSTR bstrPropertyName,
  [in] VARIANT* propertyValue
);
```

bstrPropertyName: The application-specific name for publisher property. The format for the publisher property name MUST adhere to the format specified in section 2.2.2.1.

propertyValue: The application-specific publisher property value which MUST be of the type specified in 2.2.2.2.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate both the *bstrPropertyName* and *propertyValue* parameter. If the validation fails, the server MUST fail the call. Otherwise, the server MUST attempt to store the value into the state of the DCOM object instance servicing this call specific to publisher properties, and fail the call if it cannot. Otherwise, the server MUST override any previously associated value with this property name.

3.1.4.4.27 RemovePublisherProperty (Opnum 33)

The **RemovePublisherProperty** method removes the specified application-specific publisher property for the subscription. See publisher properties in section 3.1.1.2.

```
[id(15), helpstring("method RemovePublisherProperty")]
  HRESULT RemovePublisherProperty(
  [in] BSTR bstrPropertyName
);
```

bstrPropertyName: The application-specific name for publisher property. The format for the publisher property name MUST adhere to the format specified in section 2.2.2.1.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the syntax for the *bstrPropertyName* parameter. If validation fails, the server MUST fail the call. Otherwise, the server MUST verify that the value for the property name is associated with the state of the DCOM object servicing this call specific to publisher properties. If not, the server MUST fail the call. Otherwise, the server MUST

remove any state specific to this property name associated with the state of the DCOM object servicing this call specific to publisher properties.

3.1.4.4.28 GetPublisherPropertyCollection (Opnum 34)

The **GetPublisherPropertyCollection** method gets all the application-specific publisher properties as a collection of the subscription. See publisher properties in section 3.1.1.2.

```
[id(16), helpstring("method GetPublisherPropertyCollection")]
  HRESULT GetPublisherPropertyCollection(
  [out, retval] IEventObjectCollection** collection
);
```

collection: If the function returns a success HRESULT, this MUST return an instance of DCOM object supporting the **IEventObjectCollection** which MUST contain a collection of application-specific publisher properties. These properties MUST conform to the specification given in section 2.2.2.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST enumerate all publisher properties associated with the instance of the DCOM object servicing this call. It MUST attempt to store these in a collection DCOM object supporting **IEventObjectCollection** interface and fail the call if it cannot. It MUST then return this DCOM object instance through the *collection* parameter.

3.1.4.4.29 GetSubscriberProperty (Opnum 35)

The **GetSubscriberProperty** method gets the value of an application-specific subscriber property of the subscription, as specified in section 3.1.1.2.

```
[id(17), helpstring("method GetSubscriberProperty")]
  HRESULT GetSubscriberProperty(
  [in] BSTR bstrPropertyName,
  [out, retval] VARIANT* propertyValue
);
```

bstrPropertyName: The application-specific name for subscriber property. The format for the subscriber property name MUST adhere to the format specified in section 2.2.2.1.

propertyValue: If the function returns a success HRESULT, this MUST contain the application-specific subscriber property value which MUST be of the type specified in 2.2.2.2.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate both bstrPropertyName. If validation fails, the server MUST fail the call. The server MUST then check to see if the value for this property is associated with state of the instance of the DCOM object servicing this call specific to subscriber properties. The server MUST verify that the value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the property.

3.1.4.4.30 PutSubscriberProperty (Opnum 36)

The **PutSubscriberProperty** method sets the value of an application-specific subscriber property of the subscription, as specified in section 3.1.1.2.

```
[id(18), helpstring("method PutSubscriberProperty")]
  HRESULT PutSubscriberProperty(
  [in] BSTR bstrPropertyName,
  [in] VARIANT* propertyValue
);
```

bstrPropertyName: The application-specific name for subscriber property. The format for the subscriber property name MUST adhere to the format specified in section 2.2.2.1.

propertyValue: The application-specific subscriber property value which MUST be of the type specified in 2.2.2.2.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate both the *bstrPropertyName* and *propertyValue* parameter. If the validation fails, the server MUST fail the call. Otherwise, the server MUST attempt to store the value into the state of the DCOM object instance servicing this call specific to subscriber properties, and fail the call if it cannot. The server MUST override any previously associated value with this property name.

3.1.4.4.31 RemoveSubscriberProperty (Opnum 37)

The **RemoveSubscriberProperty** method removes the specified application-specific subscriber property for the subscription, as specified in section 3.1.1.2.

```
[id(19), helpstring("method RemoveSubscriberProperty")]
  HRESULT RemoveSubscriberProperty(
  [in] BSTR bstrPropertyName
);
```

bstrPropertyName: The application-specific name for subscriber property. The format for the subscriber property name MUST adhere to the format specified in section 2.2.2.1.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the syntax for the *bstrPropertyName* parameter. If validation fails, the server MUST fail the call. Otherwise, the server MUST verify that the value for the property name is associated with the state of the DCOM object servicing this call specific to subscriber properties. If not, the server MUST fail the call. Otherwise, the server MUST attempt to remove any state specific to this property name associated with the state of the DCOM object servicing this call specific to subscriber properties, and fail the call if it cannot.

3.1.4.4.32 GetSubscriberPropertyCollection (Opnum 38)

The **GetSubscriberPropertyCollection** method gets the collection of all the application-specific subscriber properties for the subscription, as specified in section 3.1.1.2.

```
[id(20), helpstring("method GetSubscriberPropertyCollection")]
  HRESULT GetSubscriberPropertyCollection(
  [out, retval] IEventObjectCollection** collection
);
```

collection: If the function returns a success HRESULT, this MUST return an instance of a DCOM object supporting the **IEventObjectCollection** which MUST contain a collection of application-specific subscriber properties. These properties MUST conform to the specification given in section 2.2.2.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST enumerate all subscriber properties associated with the instance of the DCOM object servicing this call. It MUST attempt to store these in a collection DCOM object supporting **IEventObjectCollection** interface, and fail the call if it cannot. It MUST then return this DCOM object instance through the *collection* parameter and fail the call if it cannot.

3.1.4.4.33 get_InterfaceID (Opnum 39)

The **get_InterfaceID** method gets the InterfaceID property for the subscription.

```
[id(21), propget, helpstring("property InterfaceID")]
  HRESULT get_InterfaceID(
  [out, retval] BSTR* pbstrInterfaceID
);
```

pbstrInterfaceID: If the method returns a success HRESULT, this MUST contain the value of the InterfaceID property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the InterfaceID property.

3.1.4.4.34 put_InterfaceID (Opnum 40)

The **put_InterfaceID** method sets the InterfaceID property for the subscription.

```
[id(21), propput, helpstring("property InterfaceID")]
  HRESULT put_InterfaceID(
  [in] BSTR bstrInterfaceID
);
```

bstrInterfaceID: This is the InterfaceID property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the InterfaceID property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.5 IEnumEventObject

The **IEnumEventObject** interface provides methods that are used to enumerate a collection. The version for this interface is 0.0.

A client gets this interface by means of the **get_NewEnum (Opnum 9) (section 3.1.4.6.3)** method of the **IEventObjectCollection**. As this is a DCOM interface, Opnums 0 to Opnum 2 are **IUnknown** methods, which MUST be implemented by means of **IRemUnknown**, as specified in [MS-DCOM] section 3.2.1.5.6. The DCOM object implementing this interface MUST use the UUID {F4A07D63-2E25-11D1-9964-00C04FBBB345}.

This interface includes the following methods beyond those of **IUnknown**.

Methods in RPC Opnum Order

Method	Description
Clone (section 3.1.4.5.1)	Clones the collection into another **IEnumEventObject**-based DCOM object. Opnum: 3
Next (section 3.1.4.5.2)	Return the next elements and iterates over them. Opnum: 4
Reset (section 3.1.4.5.3)	Resets the enumerating object back to the first element. Opnum: 5
Skip (section 3.1.4.5.4)	Skips ahead in the collection. Opnum: 6

3.1.4.5.1 Clone (Opnum 3)

The **Clone** method clones the underlying collection into another DCOM object implementing the **IEnumEventObject** interface.

```
[id(1), helpstring("method Clone")]
  HRESULT Clone(
  [out] IEnumEventObject** ppInterface
);
```

ppInterface: If the function returns a success HRESULT, this MUST contain the interface pointer of the clone DCOM object supporting the **IEnumEventObject** interface.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST attempt to clone the underlying collection into another DCOM object implementing the **IEnumEventObject** interface, and return the result.

3.1.4.5.2 Next (Opnum 4)

The **Next** method gets up to a specified number of items from the collection, if they are available, starting at the current enumerator position.

```
[id(3), helpstring("method Next")]
  HRESULT Next(
  [in] ULONG cReqElem,
  [out, size_is(cReqElem), length_is(*cRetElem)]
    IUnknown** ppInterface,
  [out] ULONG* cRetElem
);
```

cReqElem: The number of elements requested by the client to return from the collection.

ppInterface: If the function returns a success HRESULT, this MUST contain an array of interface pointers of size *cRetElem*. Each element in the array MUST be either a DCOM object supporting the **IEventClass2** interface if the underlying collection is of EventClasses or the element MUST be a DCOM object supporting **IEventSubscription** DCOM interface if the underlying collection is of subscriptions.

cRetElem: If the function returns a success HRESULT, this MUST contain a number of items returned in the array contained in *ppInterface*.

Return Values: An **HRESULT** specifying success or failure. All success codes other than S_FALSE MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST attempt to return items from the current position of the enumerator on the collection, and move the enumerator ahead in the collection by the value of *cRetElem*. If the number of elements in the collection is less than *cReqElem*, the function MUST return S_FALSE for failure.

3.1.4.5.3 Reset (Opnum 5)

The **Reset** method resets the enumerator back to the first element in the collection.

```
[id(4), helpstring("method Reset")]
  HRESULT Reset();
```

This method has no parameters.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST attempt to reset the enumerator back to the first element in the collection, and fail the call if it cannot. If there are no elements in this collection, the function MUST return a success.

3.1.4.5.4 Skip (Opnum 6)

The **Skip** method skips ahead in the collection by the number of elements specified.

```
[id(5), helpstring("method Skip")]
```

```
  HRESULT Skip(
  [in] ULONG cSkipElem
);
```

cSkipElem: The number of elements to skip ahead in the collection.

Return Values: An **HRESULT** specifying success or failure. All success codes other than S_FALSE MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST attempt to skip ahead *cSkipElem* elements in the enumerator from its current location, and fail the call if it cannot. If the number of elements in the enumerator from its current location is less than the count specified in *cSkipElem*, the server MUST return S_FALSE as a success HRESULT.

3.1.4.6 IEventObjectCollection

The **IEventObjectCollection** interface provides methods that manage and enumerate over a collection of objects. The interface inherits Opnums 0 to 6 from **IDispatch** as specified in [MS-OAUT] section 3.1.4. The version for this interface is 0.0.

The server returns a DCOM object implementing this interface with UUID {f89ac270-d4eb-11d1-b682-00805fc79216} from the following methods:

1. **Query (Opnum 7) (section 3.1.4.1.1)**
2. **GetPublisherPropertyCollection (Opnum 34) (section 3.1.4.4.28)**
3. **GetSubscriberPropertyCollection (Opnum 38) (section 3.1.4.4.32)**

This interface includes the following methods beyond those of **IDispatch**.

Methods in RPC Opnum Order

Method	Description
get__NewEnum (section 3.1.4.6.1)	Returns an enumerator DCOM object. Opnum: 7
get_Item (section 3.1.4.6.2)	Gets the item in the collection based on an ID. Opnum: 8
get_NewEnum (section 3.1.4.6.3)	Gets the **IEnumEventObject** supporting the DCOM based object instance for the underlying collection. Opnum: 9
get_Count (section 3.1.4.6.4)	Gets the number of items in the collection. Opnum: 10
Add (section 3.1.4.6.5)	Adds an item to the collection. Opnum: 11
Remove (section 3.1.4.6.6)	Removes an item from the collection. Opnum: 12

3.1.4.6.1 get__NewEnum (Opnum 7)

The **get__NewEnum** method gets a DCOM object for enumerating the underlying collection.

```
[id(DISPID NEWENUM), propget, restricted, helpstring("Create new IEnumVARIANT")]
  HRESULT get  NewEnum(
  [out, retval] IUnknown** ppUnkEnum
);
```

ppUnkEnum: If the function returns a success HRESULT, this MUST contain a DCOM object which implements the **IEnumEventObject** interface.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST attempt to create a collection-based DCOM object which supports the **IEnumEventObject** interface, and fail the call if it cannot. The underlying collection MUST be the same collection that is being enumerated by the DCOM object instance servicing this call. The created collection DCOM object MUST be returned through the **ppUnkEnum** interface.

3.1.4.6.2 get_Item (Opnum 8)

The **get_Item** method gets the item in the collection with a specified ID.

```
[id(DISPID_VALUE), propget] HRESULT get_Item(
  BSTR objectID,
  [out, retval] VARIANT* pItem
);
```

objectID: The name of the object to get from the collection. If the underlying collection is of publisher or subscriber application-specific subscription properties, this name MUST conform to the specification of application-specific property names as specified in section 2.2.2.1. If the underlying collection is of event classes, this MUST conform to the specification of EventClassID as specified in section 3.1.1.1. If the underlying collection is of subscriptions, this MUST conform to the specification of SubscriptionID as specified in section 3.1.1.2.

pItem: If the function returns a success HRESULT, this MUST contain an application-specific publisher/subscriber property as specified in section 2.2.2.2 if the underlying collection is of publisher/subscriber application-specific subscriptions properties. If the underlying collection is of event classes, then this MUST contain a VT_IUNKNOWN for the DCOM object supporting the **IEventClass2** DCOM interface. If the underlying collection is of subscriptions, then this MUST contain a VT_IUNKNOWN for the DCOM object supporting the **IEventSubscription** DCOM interface.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the objectID parameter for syntax. If the validation fails the server MUST fail the call. Otherwise, the server MUST enumerate the collection and match the objectID to the individual objects in the collection. If an object with the matching objectID is found, it MUST be returned through the *pItem* parameter. If not the server MUST fail the call.

3.1.4.6.3 get_NewEnum (Opnum 9)

The **get_NewEnum** method gets an **IEnumEventObject**-based object for enumerating the underlying collection.

```
[[id(1), propget, helpstring("Create new IEnumEventObject")]
  HRESULT get_NewEnum(
  [out, retval] IEnumEventObject** ppEnum
);
```

ppEnum: If the function returns a success HRESULT, this MUST contain a DCOM object supporting the **IEnumEventObject** interface. Note this method is only supported if the underlying collection is of event classes or subscriptions.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST attempt to create a collection-based DCOM object which supports the **IEnumEventObject** interface, and fail the call if it cannot. The underlying collection MUST be the same collection that is being enumerated by the DCOM object instance servicing this call. The created collection DCOM object MUST be returned through the **ppEnum** interface.

3.1.4.6.4 get_Count (Opnum 10)

The **get_Count** method gets the count of number of items in the collection contained by the enumerator.

```
[id(2), propget, helpstring("Number of items in the collection")]
  HRESULT get_Count(
  [out, retval] long* pCount
);
```

pCount: If the function returns a success HRESULT, this MUST contain the number specifying the number of elements in the underlying collection.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST attempt to return the count of the number of elements in the collection. If the server is not able to return the count, it MUST fail the call.

3.1.4.6.5 Add (Opnum 11)

The **Add** method adds an item to the underlying collection of the enumerator; if the item is already present in the collection, it is replaced by the one being passed to this method.

```
[id(3), helpstring("Add an item to the collection")]
  HRESULT Add(
  [in] VARIANT* item,
  [in] BSTR objectID
);
```

item: If the underlying collection of application-specific publisher/subscriber subscription properties, this MUST conform to the application-specific property values as specified in section 2.2.2.2. If the underlying collection is of event classes, the type of the VARIANT MUST be VT_UNKNOWN and MUST contain a DCOM object supporting the **IEventClass2** interface. If the underlying collection is of subscriptions, the type of the VARIANT MUST be VT_UNKNOWN and MUST contain a DCOM object supporting the **IEventSubscription** interface.

objectID: The identity of the object. If the underlying collection is of the application-specific publisher/subscriber subscription properties, this MUST conform to the application-specific property names as specified in 2.2.2.1. If the underlying collection is of event classes, this MUST conform to the EventClassID property of the event class as specified in section 3.1.1.1. If the underlying collection is of subscriptions, this MUST conform to the SubscriptionID property of the subscription as specified in section 3.1.1.2.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the *item* and the *objectID* parameters. If the validation fails, the server MUST fail the call. Otherwise, the server MUST attempt to add this item to the collection of the enumerator DCOM object servicing this call, and fail the call if it cannot. If object in the collection already has the *objectID* specified in the call, the server MUST fail the call.

3.1.4.6.6 Remove (Opnum 12)

The **Remove** method removes an item from the underlying collection of the enumerator.

```
[id(4), helpstring("Remove an item from the collection")]
  HRESULT Remove(
  [in] BSTR objectID
);
```

objectID: The identity of the object. If the underlying collection is of the application-specific publisher/subscriber subscription properties this MUST conform to the application-specific property names as specified in 2.2.2.1. If the underlying collection is of event classes this MUST conform to the EventClassID property of the event class as specified in section 3.1.1.1. If the underlying collection is of subscriptions this MUST conform to the SubscriptionID property of the subscription as specified in section 3.1.1.2.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the syntax of the objectID. If the validation fails, the server MUST fail the call. Otherwise, the server MUST enumerate through the collection and remove the object matching the given objectID. If the object matching the objectID is not found in the collection, the server MUST fail the call.

3.1.4.7 IEventClass3

The **IEventClass3** interface provides additional methods that are used by the client to manipulate an event class on the server. This interface inherits Opnums 0 to Opnum 28 from the **IEventClass2** interface, as specified in section 3.1.4.3. The version for this interface is 0.0.

The server SHOULD support this interface.<11> To receive incoming remote calls for this interface, the server MUST implement an DCOM Object class with the CLSID CLSID_EventClass (see section 1.9) using the UUID {7FB7EA43-2D76-4ea8-8CD9-3DECC270295E} for this interface.

The interface contains the following methods beyond the ones defined for **IEventClass2**.

Methods in RPC Opnum Order

Method	Description
get_EventClassPartitionID (section 3.1.4.7.1)	Gets the EventClassPartitionID property of the event class. Opnum: 29
put_EventClassPartitionID (section 3.1.4.7.2)	Sets the EventClassPartitionID property of the event class. Opnum: 30
get_EventClassApplicationID (section 3.1.4.7.3)	Gets the EventClassApplicationID property of the event class. Opnum: 31
put_EventClassApplicationID (section 3.1.4.7.4)	Sets the EventClassApplicationID property of the event class. Opnum: 32

3.1.4.7.1 get_EventClassPartitionID (Opnum 29)

The **get_EventClassPartitionID** method gets the EventClassPartitionID property of the event class.

```
[id(12), propget, helpstring("property EventClassPartitionID")]
  HRESULT get_EventClassPartitionID(
  [out, retval] BSTR* pbstrEventClassPartitionID
);
```

pbstrEventClassPartitionID: If the function returns a success HRESULT, this MUST contain the EventClassPartitionID property of the event class as specified in section 3.1.1.1. The value MUST conform to the format as specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the EventClassPartitionID property.

3.1.4.7.2 put_EventClassPartitionID (Opnum 30)

The **put_EventClassPartitionID** method sets the EventClassPartitionID property for the event class.

```
[id(12), propput, helpstring("property EventClassPartitionID")]
  HRESULT put_EventClassPartitionID(
  [in] BSTR bstrEventClassPartitionID
);
```

bstrEventClassPartitionID: The value of the EventClassPartitionID property of the event class as specified in section 3.1.1.1. The value MUST conform to the format as specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the EventClassPartitionID property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.7.3 get_EventClassApplicationID (Opnum 31)

The **get_EventClassApplicationID** method gets the EventClassApplicationID property of the event class.

```
[id(13), propget, helpstring("property EventClassApplicationID")]
  HRESULT get_EventClassApplicationID(
  [out, retval] BSTR* pbstrEventClassApplicationID
);
```

pbstrEventClassApplicationID: If the function returns a success HRESULT, this MUST contain the EventClassApplicationID property of the event class as specified in section 3.1.1.1. The value MUST conform to the format as specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the EventClassApplicationID property.

3.1.4.7.4 put_EventClassApplicationID (Opnum 32)

The **put_EventClassApplicationID** method sets the EventClassApplicationID property of the event class.

```
[id(13), propput, helpstring("property EventClassApplicationID")]
  HRESULT put_EventClassApplicationID(
  [in] BSTR bstrEventClassApplicationID
);
```

bstrEventClassApplicationID: The value of the EventClassApplicationID property of the event class as specified in section 3.1.1.1. The value MUST conform to the format as specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the

EventClassApplicationID property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.8 IEventSubscription2

The **IEventSubscription2** interface provides methods to get/set the properties of a subscription. This interface inherits Opnums 0 to 40 from **IEventSubscription** as specified in section 3.1.4.4. The version for this interface is 0.0.

To receive incoming remote calls for this interface, the server MUST implement an DCOM Object Class with the CLSID CLSID_Subscription (see section 1.9) using the UUID {4A6B0E16-2E38-11D1-9965-00C04FBBB345} for this interface.

The interface contains the following methods beyond the ones for **IEventSubscription**.

Methods in RPC Opnum Order

Method	Description
get_FilterCriteria (section 3.1.4.8.1)	Gets the FilterCriteria property of the subscription. Opnum: 41
put_FilterCriteria (section 3.1.4.8.2)	Sets the FilterCriteria property of the subscription. Opnum: 42
get_SubscriberMoniker (section 3.1.4.8.3)	Gets the SubscriberMoniker property of the subscription. Opnum: 43
put_SubscriberMoniker (section 3.1.4.8.4)	Sets the SubscriberMoniker property of the subscription. Opnum: 44

3.1.4.8.1 get_FilterCriteria (Opnum 41)

The **get_FilterCriteria** method gets the FilterCriteria property for the subscription.

```
[propget, id(22), helpstring("property FilterCriteria")]
  HRESULT get_FilterCriteria(
  [out, retval] BSTR* pbstrFilterCriteria
);
```

pbstrFilterCriteria: If the method returns a success HRESULT, this MUST contain the value of the FilterCriteria property of the subscription, as specified in section 3.1.1.2. The syntax for this string is specified in section 2.2.1.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the FilterCriteria property.

3.1.4.8.2 put_FilterCriteria (Opnum 42)

The **put_FilterCriteria** method sets the value of the FilterCriteria property of the subscription.

```
[propput, id(22), helpstring("property FilterCriteria")]
  HRESULT put_FilterCriteria(
  [in] BSTR bstrFilterCriteria
);
```

bstrFilterCriteria: The FilterCriteria property of the subscription, as specified in section 3.1.1.2. The syntax for this string is specified in section 2.2.1.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the FilterCriteria property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.8.3 get_SubscriberMoniker (Opnum 43)

The **get_SubscriberMoniker** method gets the SubscriberMoniker property of the subscription.

```
[propget, id(23), helpstring("property SubscriberMoniker")]
  HRESULT get_SubscriberMoniker(
  [out, retval] BSTR* pbstrMoniker
);
```

pbstrMoniker: If the method returns a success HRESULT, this MUST contain the value of the SubscriberMoniker property of the subscription, as specified in section 3.1.1.2. This MUST be a string of character length less than or equal to 255.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the SubscriberMoniker property.

3.1.4.8.4 put_SubscriberMoniker (Opnum 44)

The **put_SubscriberMoniker** method sets the SubscriberMoniker property of the subscription.

```
[propput, id(23), helpstring("property SubscriberMoniker")]
  HRESULT put_SubscriberMoniker(
  [in] BSTR bstrMoniker
);
```

bstrMoniker: The SubscriberMoniker property of the subscription, as specified in section 3.1.1.2. This MUST be a string of character length less than or equal to 255.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the SubscriberMoniker property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.9 IEventSubscription3

The **IEventSubscription3** interface provides methods to get or set the properties of a subscription.<12> This interface inherits Opnums 0 to Opnum 44 from **IEventSubscription2**, as specified in section 3.1.4.8. The version for this interface is 0.0.

To receive incoming remote calls for this interface, the server MUST implement an DCOM Object Class with the CLSID CLSID_Subscription (see section 1.9) using the UUID {FBC1D17D-C498-43a0-81AF-423DDD530AF6} for this interface.

The interface contains the following methods beyond those of **IEventSubscription2**.

Methods in RPC Opnum Order

Method	Description
get_EventClassPartitionID (section 3.1.4.9.1)	Gets the EventClassPartitionID property of the subscription. Opnum: 45
put_EventClassPartitionID (section 3.1.4.9.2)	Sets the EventClassPartitionID property of the subscription. Opnum: 46
get_EventClassApplicationID (section 3.1.4.9.3)	Gets the EventClassApplicationID property of the subscription. Opnum: 47
put_EventClassApplicationID (section 3.1.4.9.4)	Sets the EventClassApplicationID property of the subscription. Opnum: 48
get_SubscriberPartitionID (section 3.1.4.9.5)	Gets the SubscriberPartitionID property of the subscription. Opnum: 49
put_SubscriberPartitionID (section 3.1.4.9.6)	Sets the SubscriberPartitionID property of the subscription. Opnum: 50
get_SubscriberApplicationID (section 3.1.4.9.7)	Gets the SubscriberApplicationID property of the subscription. Opnum: 51
put_SubscriberApplicationID (section 3.1.4.9.8)	Sets the SubscriberApplicationID property of the subscription. Opnum: 52

3.1.4.9.1 get_EventClassPartitionID (Opnum 45)

The **get_EventClassPartitionID** gets the EventClassPartitionID property for the subscription.

```
[propget, id(24), helpstring("property EventClassPartitionID")]
  HRESULT get_EventClassPartitionID(
  [out, retval] BSTR* pbstrEventClassPartitionID
);
```

pbstrEventClassPartitionID: If the method returns a success HRESULT, this MUST contain the EventClassPartitionID property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the EventClassPartitionID property.

3.1.4.9.2 put_EventClassPartitionID (Opnum 46)

The **put_EventClassPartitionID** method sets the EventClassPartitionID property for the subscription.

```
[propput, id(24), helpstring("property EventClassPartitionID")]
  HRESULT put_EventClassPartitionID(
  [in] BSTR bstrEventClassPartitionID
);
```

bstrEventClassPartitionID: The EventClassPartitionID property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails the server MUST fail the method call. Otherwise the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the EventClassPartitionID property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.9.3 get_EventClassApplicationID (Opnum 47)

The **get_EventClassApplicationID** method gets the EventClassApplicationID property for the subscription.

```
[propget, id(25), helpstring("property EventClassApplicationID"]
  HRESULT get_EventClassApplicationID(
  [out, retval] BSTR* pbstrEventClassApplicationID
);
```

pbstrEventClassApplicationID: If the method returns a success HRESULT, this MUST contain the EventClassApplicationID property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the EventClassApplicationID property.

3.1.4.9.4 put_EventClassApplicationID (Opnum 48)

The **put_EventClassApplicationID** method sets the EventClassApplicationID property for the subscription.

```
[propput, id(25), helpstring("property EventClassApplicationID")]
  HRESULT put_EventClassApplicationID(
  [in] BSTR bstrEventClassApplicationID
);
```

bstrEventClassApplicationID: The EventClassApplicationID property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails the server MUST fail the method call. Otherwise the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the EventClassApplicationID property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.9.5 get_SubscriberPartitionID (Opnum 49)

The **get_SubscriberPartitionID** method gets the SubscriberPartitionID for the subscription.

```
[propget, id(26), helpstring("property SubscriberPartitionID")]
  HRESULT get_SubscriberPartitionID(
  [out, retval] BSTR* pbstrSubscriberPartitionID
);
```

pbstrSubscriberPartitionID: If the method returns a success HRESULT, this MUST contain the SubscriberPartitionID property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the SubscriberPartitionID property.

3.1.4.9.6 put_SubscriberPartitionID (Opnum 50)

The **put_SubscriberPartitionID** method sets the SubscriberPartitionID property for the subscription.

```
[propput, id(26), helpstring("property SubscriberPartitionID")]
  HRESULT put_SubscriberPartitionID(
  [in] BSTR bstrSubscriberPartitionID
);
```

bstrSubscriberPartitionID: The SubscriberPartitionID property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise, the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the SubscriberPartitionID property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.9.7 get_SubscriberApplicationID (Opnum 51)

The **get_SubscriberApplicationID** method gets the SubscriberApplicationID property for the subscription.

```
[propget, id(27), helpstring("property SubscriberApplicationID")]
  HRESULT get_SubscriberApplicationID(
  [out, retval] BSTR* pbstrSubscriberApplicationID
);
```

pbstrSubscriberApplicationID: If the method returns a success HRESULT, this MUST contain the SubscriberApplicationID property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST verify that this value was previously set on this DCOM object instance servicing this method call. If not, the server MUST fail the call. Otherwise, the server MUST return the value of the SubscriberApplicationID property.

3.1.4.9.8 put_SubscriberApplicationID (Opnum 52)

The **put_SubscriberApplicationID** method sets the SubscriberApplicationID property for the subscription.

```
[propput, id(27), helpstring("property SubscriberApplicationID")]
  HRESULT put_SubscriberApplicationID(
  [in] BSTR bstrSubscriberApplicationID
);
```

bstrSubscriberApplicationID: The SubscriberApplicationID property of the subscription, as specified in section 3.1.1.2. The value MUST conform to the format as specified in section 2.2.3.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST validate the new value of the property for syntax. If validation fails, the server MUST fail the method call. Otherwise the server MUST attempt to update the state of the DCOM object servicing this method with the new value of the SubscriberApplicationID property, and fail the call if it cannot. Otherwise, it MUST override any previous value that was set for this DCOM object instance servicing this method call.

3.1.4.10 IEventSystem2

This interface is used to perform version checking and transient subscription verifications on the server by the client. **IEventSystem2** inherits Opnums 0 to 12 from the **IEventSystem** interface, as specified in section 3.1.4.1. The version for this interface is 0.0.

The server SHOULD support this interface.<13> To receive incoming remote calls for this interface, the server MUST implement a DCOM object class with the CLSID CLSID_EventSystem (see section 1.9) using the UUID {99CC098F-A48A-4e9c-8E58-965C0AFC19D5}for this interface.

The interface contains the following methods beyond those of **IEventSystem**.

Methods in RPC Opnum Order

Method	Description
GetVersion (section 3.1.4.10.1)	Gets the version of the event system implementation. Opnum: 13
VerifyTransientSubscribers (section 3.1.4.10.2)	Verifies the state of the transient subscribers. Opnum: 14

3.1.4.10.1 GetVersion (Opnum 13)

The **GetVersion** method retrieves the version of the server implementation of the protocol.

```
[id(7), helpstring("method GetVersion")]
  HRESULT GetVersion(
  [out] int* pnVersion
);
```

pnVersion: If the function returns a success HRESULT, it MUST contain one of the following:

Value	Meaning
0x00000001	The server does not support the **IEventSubscription3** and **IEventClass3** interfaces, nor does it support the PartitionID and ApplicationID properties on the subscription (section 3.1.1.2) and event class (section 3.1.1.1) objects.
0x00000002	The server supports the **IEventSubscription3** and **IEventClass3** interfaces. It also

Value	Meaning
	supports the PartitionID and ApplicationID properties on the subscription (section 3.1.1.2) and event class (section 3.1.1.1) objects.

Return Values: An **HRESULT** specifying success or failure. All success codes MUST be treated the same, and all failure codes MUST be treated the same.

When this method is invoked, the server MUST attempt to return the pnVersion value corresponding to the interfaces it supports, and fail the call if it cannot.

3.1.4.10.2 VerifyTransientSubscribers (Opnum 14)

The **VerifyTransientSubscribers** method verifies the state of the transient subscribers.

```
[id(8), helpstring("method VerifyTransientSubscribers")]
  HRESULT VerifyTransientSubscribers();
```

This method has no parameters.

Return Values: This function MUST return S_OK.

When this method is invoked, the server MUST verify that, for all transient subscribers, the server is able to make a method call on them successfully. If any transient subscription fails this test, the server MUST remove the subscription from its internal store for subscriptions.

3.1.4.11 IEventSystemInitialize

The **IEventSystemInitialize** interface is used to initialize the server implementing this protocol. As this is a DCOM interface, Opnum 0 to Opnum 2 are **IUnknown** methods, which MUST be implemented by means of **IRemUnknown**, as specified in [MS-DCOM] section 3.2.1.5.6. The version for this interface is 0.0.

To receive incoming remote calls for this interface, the server MUST implement an DCOM object class with the CLSID CLSID_EventSystem (see section 1.9) using the UUID {a0e8f27a-888c-11d1-b763-00c04fb926af} for this interface .

The interface contains the following methods beyond those of IUnknown.

Methods in RPC Opnum Order

Method	Description
SetCOMCatalogBehaviour	Initializes the server. Opnum: 3

3.1.4.11.1 SetCOMCatalogBehaviour (Opnum 3)

The **SetCOMCatalogBehaviour** method does nothing.

```
HRESULT SetCOMCatalogBehaviour(
  BOOL bRetainSubKeys
);
```

bRetainSubKeys: The server SHOULD ignore this value.

Return Values: The server MUST return S_OK.

The server MUST NOT expect clients to call this method. The client application SHOULD NOT call this method.<14>

3.1.5 Timer Events

None.

3.1.6 Other Local Events

None.

4 Protocol Examples

The following examples build on the examples given in [MS-DCOM] section 4.1.

4.1 Creating an Event Class

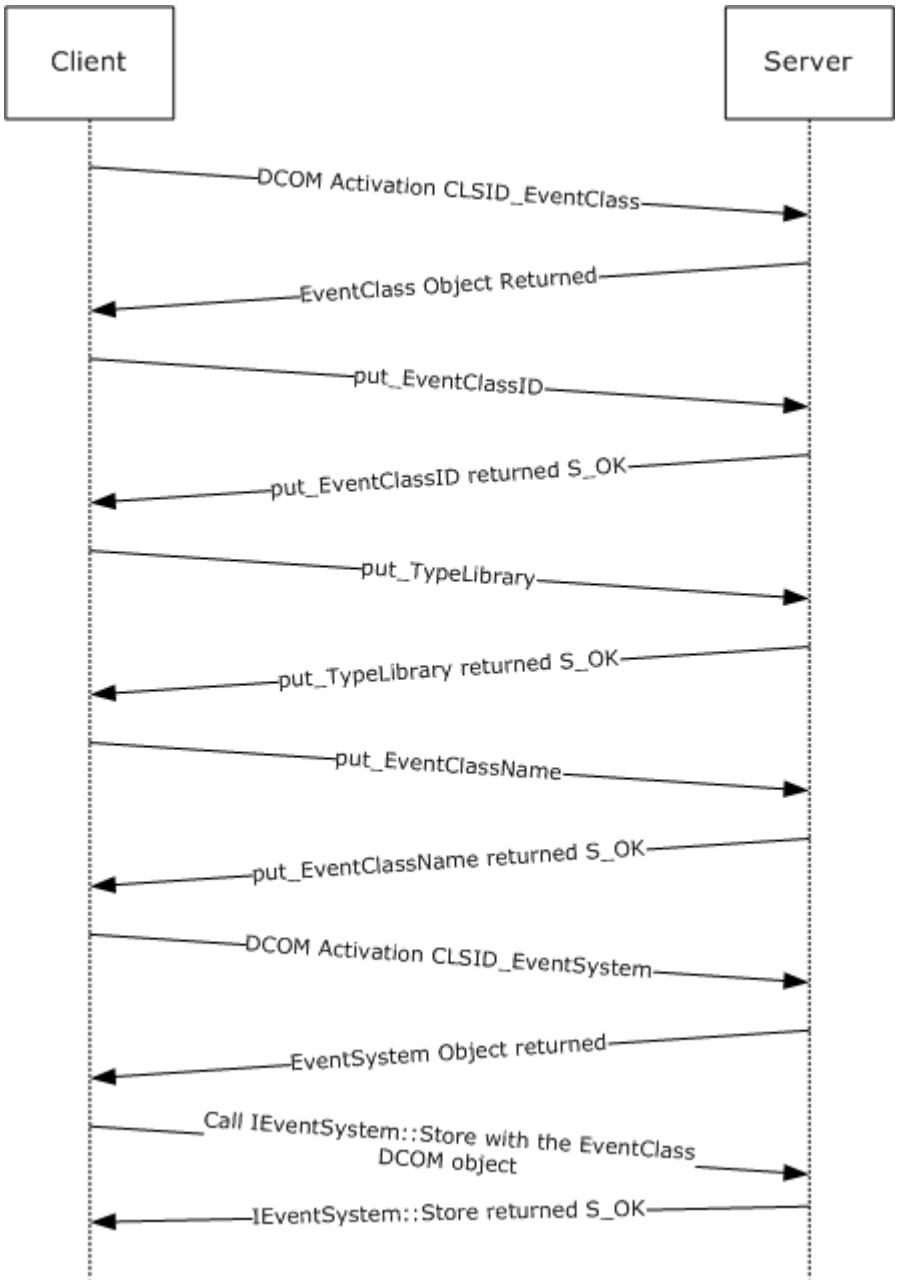

Figure 1: Creating an Event Class

Figure 1 shows the sequence for a client publisher application creating an event on the server. It assumes that the client already knows that the type library ID for the EventClass is 3BFF4039-03C2-410f-B6A6-F4EBC250C2CC.

To set up an event class:

1. The client application starts by performing a DCOM activation for the event class object on the server by using the CLSID CLSID_EventClass.

2. The client application dynamically generates a UUID (in this example, DF01D194-D694-41e5-BA79-8DEDE00ED0EA) according to [C706] section A.2.5. It converts the UUID to a string using the format specified in section 2.2.3. Using the string the client calls **put_EventClassID** on the event class object to set the EventClassID property.

```
HRESULT
put_EventClassID(
  [in] BSTR bstrEventClassID ="{DF01D194-D694-41e5-BA79-8DEDE00ED0EA}"
  );
```

3. The server stores the EventClassID and returns S_OK.

4. It then sets the Typelib property of the event class by calling **put_TypeLib** method.

```
HRESULT
put_TypeLib(
  [in] BSTR bstrTypeLib ="{3BFF4039-03C2-410f-B6A6-F4EBC250C2CC}"
  );
```

5. The server stores the TypeLib property and returns S_OK.

6. The client chooses a human-readable name for the event class (in this example, "TestEventClass"). It then uses this name to set the EventClassName property by calling the **put_EventClassName** method.

```
HRESULT
put_EventClassName(
  [in] BSTR bstrEventClassName = "TestEventClass");
```

7. The server stores the EventClassName and returns S_OK.

To store an event class:

1. The client activates the CLSID CLSID_EventSystem to get the EventSystem DCOM object on the server.

2. The client calls **Store** on the EventSystem DCOM object with the "EventSystem.EventClass".

```
HRESULT
Store([in] BSTR ProgID = "EventSystem.EventClass",
  [in] IUnknown* pInterface = {pointer to CLSID_EventClass
                               interface created above}
  );
```

3. The server verifies the values specified in the subscription object. In this example it verifies that the EventClassID is not already in its event class store. If it were, the server would update the current entry with new value for the properties and return S_OK. However, in this example, it is not already in the store, and so the server creates an entry for the event class in its store and returns S_OK.

4.2 Creating a Subscription

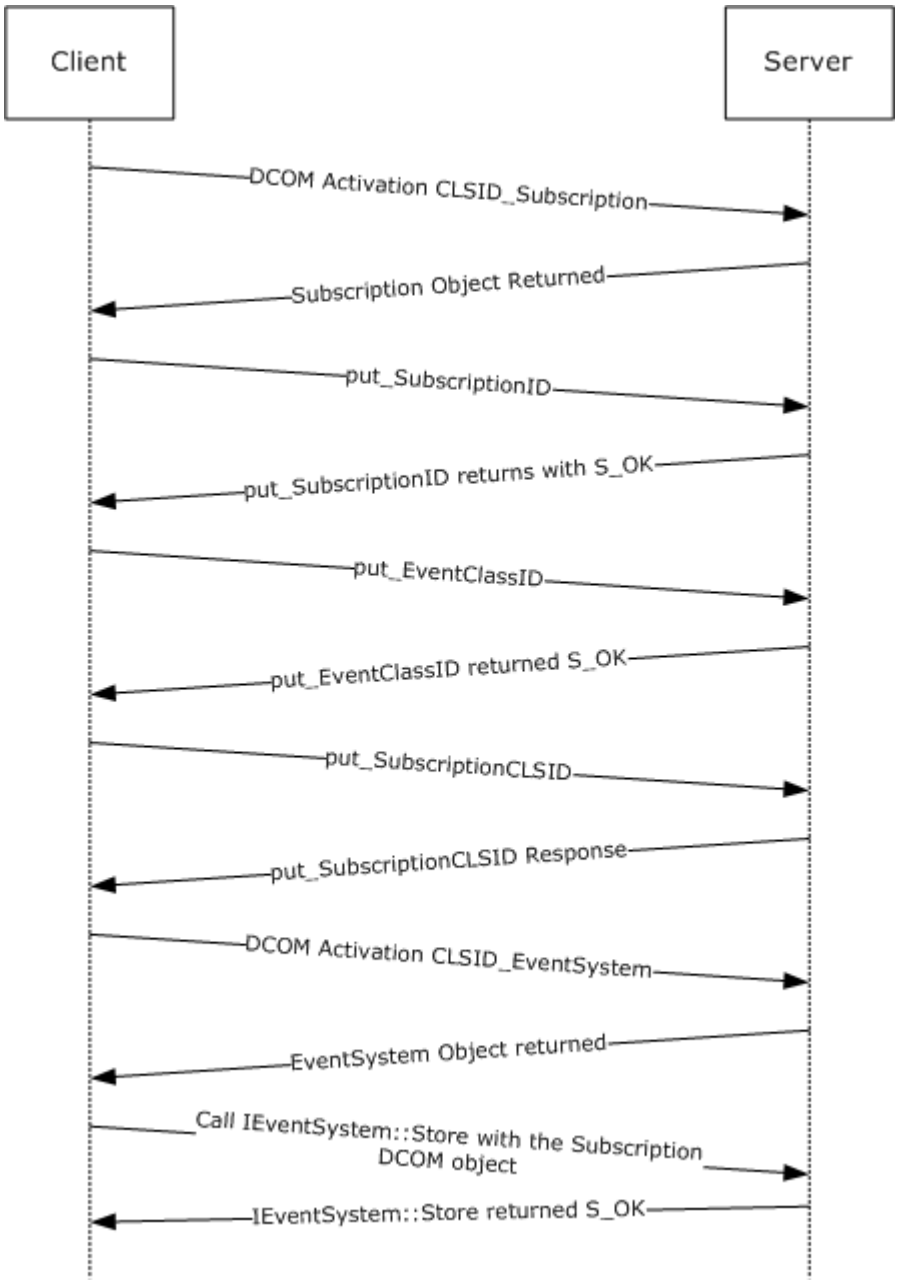

Figure 2: Creating a Subscription

Figure 2 shows the sequence for a client application creating a subscription on the server for the event class that has the UUID DF01D194-D694-41e5-BA79-8DEDE00ED0EA. It assumes that the client application already knows the subscriber application DCOM object CLSID which is 19D10A70-1B07-4b76-87B6-99F58DEE37E7.

To set up the subscription:

1. The client starts by performing a DCOM activation for the subscription DCOM object on the server by using the CLSID CLSID_EventSubscription.

2. The server returns an object reference to the subscription DCOM object.

3. The client generates a UUID (for example, B7E3D561-3BB1-46df-B47F-51DF3B307EC9) according to [C706] section A.2.5. It converts the UUID to a string using the format specified in section 2.2.3. Using the string the client calls **put_SubscriptionID** on the subscription DCOM object to set the SubscriptionID property for the subscription.

```
HRESULT
put_SubscriptionID(
[in] BSTR bstrSubscriptionID ="{B7E3D561-3BB1-46df-B47F-51DF3B307EC9}");
```

4. The server stores the SubscriptionID, and returns S_OK.

5. The client then sets the EventClassID property which identifies the event class for which it is creating the subscription by calling **put_EventClassID**.

```
HRESULT
put_EventClassID(
[in] BSTR bstrEventClassID ="{DF01D194-D694-41e5-BA79-8DEDE00ED0EA}");
```

6. The server stores the EventClassID, and returns S_OK.

7. It then gets the SubscriberCLSID property by calling the **put_SubscriberCLSID** method.

```
HRESULT
put_SubscriberCLSID(
BSTR bstrSubscriberCLSID ="{19D10A70-1B07-4b76-87B6-99F58DEE37E7}");
```

8. The server stores the SubscriberCLSID and returns S_OK.

The CLSID_Subscription object at this point is just a place holder; no verification of the data is done at this point.

To store the subscription:

1. The client performs DCOM activation for CLSID_EventSystem on the server for the EventSystem DCOM object.

2. The server returns an object reference to the EventSystem DCOM object.

3. The client calls **Store** on the EventSystem DCOM object with the string "EventSystem.EventSubscription".

```
HRESULT
Store(
  [in] BSTR ProgID = "EventSystem.EventSubscription",
  [in] IUnknown* pInterface = {pointer to CLSID_Subscription
                               interface created above}
  );
```

4. The server now verifies the values specified in the subscription object. In this example it verifies that the EventClassID is in its event class store. It then checks to see whether the SubscriberID matches any existing subscription. Since it does not, the server goes ahead and creates a subscription, and returns S_OK.

4.3 Updating a Subscription

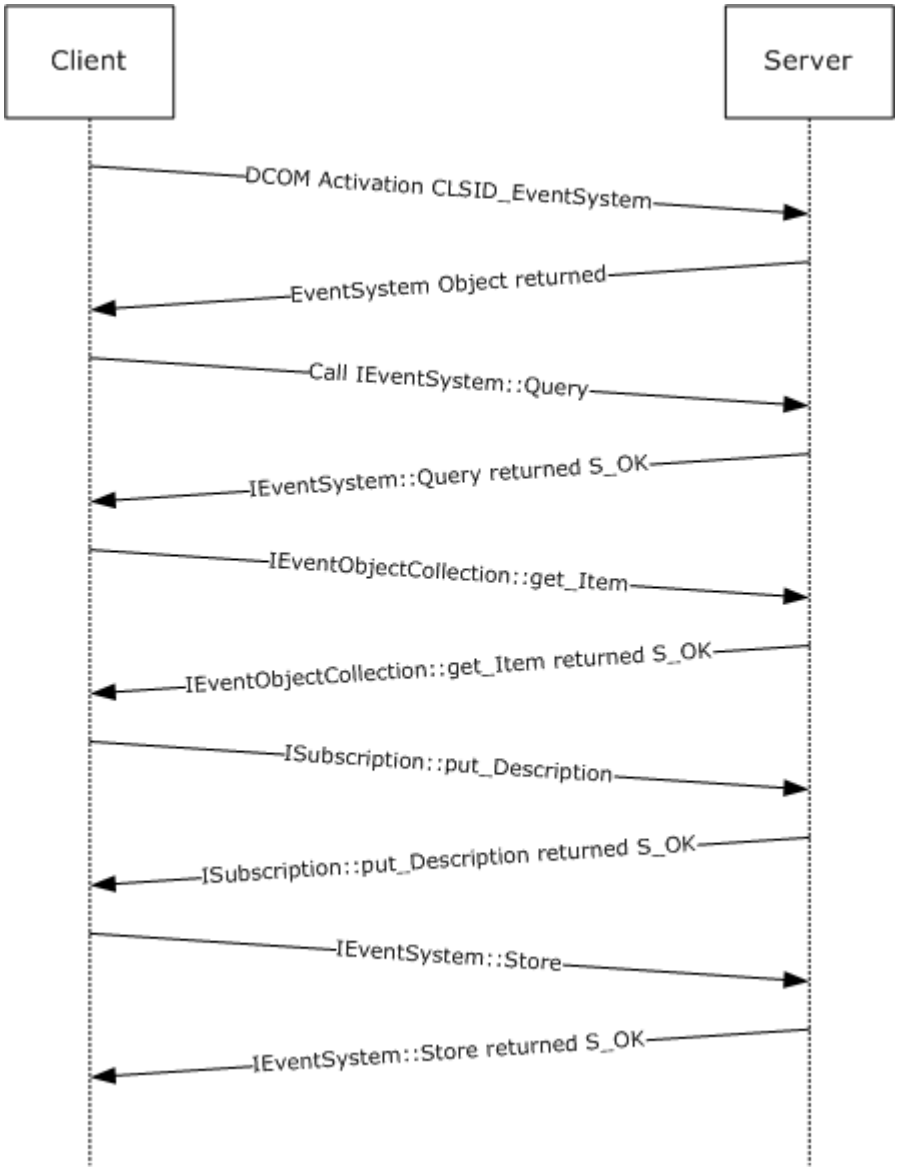

Figure 3: Updating a Subscription

Figure 3 shows a sequence of a client application updating its subscription. From before, the client knows that its SubscriptionID is B7E3D561-3BB1-46df-B47F-51DF3B307EC9, and its SubscriberCLSID is 19D10A70-1B07-4b76-87B6-99F58DEE37E7.

1. The client activates the CLSID CLSID_EventSystem to get the EventSystem DCOM object on the server.

2. The client calls **Query** with the subscriptions collection.

```
HRESULT
HRESULT Query(
  [in] BSTR progID= "EventSystem.EventSubscriptionCollection",
  [in] BSTR queryCriteria = "SubscriberCLSID='{19D10A70-1B07-4b76-87B6-
99F58DEE37E7}'",
  [out] int* errorIndex = {uninitialized},
  [out, retval] IUnknown** ppInterface = {uninitialized}
  );
```

3. The server goes through its internal store for subscriptions and looks for a subscription which has the SubscriberCLSID property set to "{19D10A70-1B07-4b76-87B6-99F58DEE37E7}". After successfully finding the subscription with the matching SubscriberCLSID property, the server creates a DCOM object for the subscription and populates its state with the subscription properties. The DCOM object is then stored in a collection-based DCOM object supporting the **IEventObjectCollection** interface. The server then returns S_OK with *ppInterface* containing the collection DCOM object:

```
HRESULT = S_OK
Query(
  [in] BSTR progID = {unchanged},
  [in] BSTR queryCriteria = {unchanged},
  [out] int* errorIndex = 0,
  [out, retval] IUnknown** ppInterface = {DCOM object supporting
                            IEventObjectCollection interface}
  );
```

4. The client calls the **get_Item** method to get the particular subscription for its SubscriptionID.

```
HRESULT
Item(
  [in] BSTR objectID = "{B7E3D561-3BB1-46df-B47F-51DF3B307EC9}",
  [out,retval] VARIANT* pItem = {uninitialized)
  );
```

5. The server goes through the underlying collection and looks for the subscription DCOM object that has the SubscriptionID property set to "{B7E3D561-3BB1-46df-B47F-51DF3B307EC9}". After successfully finding the object, it embeds it in a VARIANT in the **punk** field and sets the VT_TYPE on the VARIANT to VT_IUNKNOWN and returns S_OK:

```
HRESULT = S_OK
Item(
  [in] BSTR objectID = {unchanged},
```

```
    [out,retval] VARIANT* pItem = {vt = VT IUNKNOWN,punk = {DCOM object that supports
                                    IEventSubscrption interface}
    );
```

6. The client calls **put_Description** to update the description of the item.

```
HRESULT
put_Description(
  [in] BSTR* pbstrDescription = "A custom subscription"
  );
```

7. The server stores the description and returns S_OK.

8. The client calls **Store** on the EventSystem DCOM object with the "EventSystem.EventSubscription" string.

```
HRESULT
Store(
  [in] BSTR ProgID = "EventSystem.EventSubscription",
  [in] IUnknown* pInterface = {interface pointer to subscription
                               object updated above}
  );
```

9. The server verifies that the description is 255 characters or less, updates the description of the subscription in its local state, and returns S_OK.

Note that the same set of operations can be performed by the client to update an event class.

4.4 Removing a Subscription

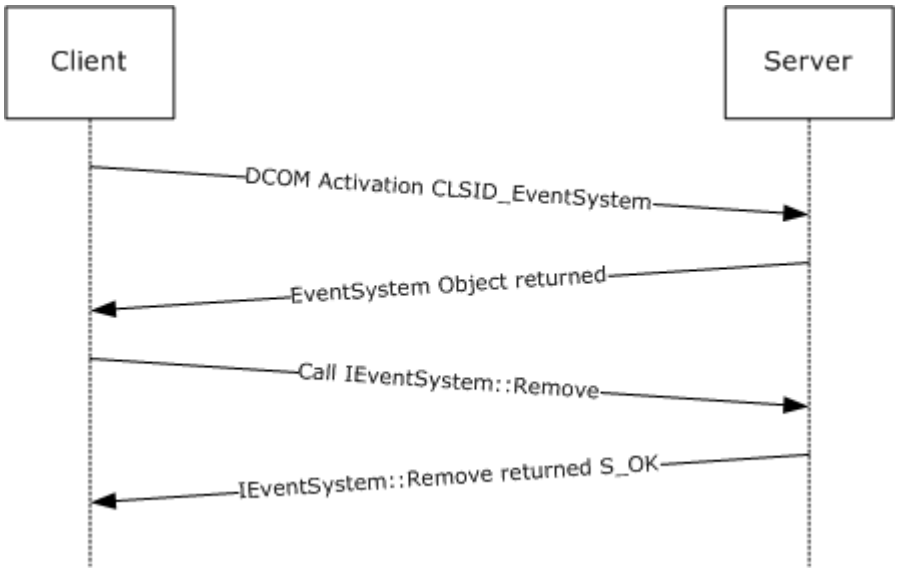

Figure 4: Removing a Subscription

Figure 4 shows the sequence of a client application submitting a request to a server to remove all subscriptions with its Subscriber CLSID. From before, the client knows that its SubscriberCLSID is 19D10A70-1B07-4b76-87B6-99F58DEE37E7.

1. The client activates the CLSID CLSID_EventSystem to get the EventSystem DCOM object on the server.

2. The client calls the **Remove** method with the appropriate collection name for subscriptions (i.e., "EventSystem.EventSubscriptionCollection"), along with the criteria for removing it.

```
HRESULT
Remove(
  [in] BSTR progID = "EventSystem.EventSubscriptionCollection",
  [in] BSTR queryCriteria = "SubscriberCLSID='{19D10A70-1B07-4b76-87B6-
99F58DEE37E7}'",
  [out] int* errorIndex = (uninitialized)
  );
```

As a result of this method call the server finds all subscriptions with that Subscriber CLSID, removes them, and returns:

```
HRESULT = S_OK
Remove(
  [in] BSTR progID = {unchanged},
  [in] BSTR queryCriteria = {unchanged},
  [out] int* errorIndex = 0
  );
```

Note that the same set of operations can be performed by the client application to remove an event class.

5 Security

The following sections specify security considerations for implementers of the COM+ Event System Protocol.

5.1 Security Considerations for Implementers

Implementers should ensure that authorization checks exist on the event class and subscription stores.

5.2 Index of Security Parameters

None.

6 Appendix A: Full IDL

For ease of implementation, the full IDL is provided below, where "ms-oaut.idl" is the IDL found in [MS-OAUT] Appendix A.

```
import "ms-oaut.idl";
interface IEventObjectCollection;

[
 object,
 uuid(4E14FB9F-2E22-11D1-9964-00C04FBBB345),
 dual,
 helpstring("IEventSystem Interface"),
 pointer default(unique)
]
interface IEventSystem : IDispatch
{
[id(1), helpstring("method Query")]
HRESULT Query([in] BSTR progID,
  [in] BSTR queryCriteria,
  [out] int* errorIndex,
  [out,retval] IUnknown** ppInterface);

[id(2), helpstring("method Store")]
HRESULT Store([in] BSTR ProgID,
  [in] IUnknown* pInterface);

[id(3), helpstring("method Remove")]
HRESULT Remove([in] BSTR progID,
   [in] BSTR queryCriteria,
   [out] int* errorIndex);

[id(4), propget, helpstring("method get EventObjectChangeEventClassID")]
HRESULT EventObjectChangeEventClassID([out,retval] BSTR* pbstrEventClassID);

[id(5), helpstring("method QueryS")]
HRESULT QueryS([in] BSTR progID,
   [in] BSTR queryCriteria,
   [out,retval] IUnknown** ppInterface);

[id(6), helpstring("method RemoveS")]
HRESULT RemoveS([in] BSTR progID,
[in] BSTR queryCriteria);
};



[
 object,
 uuid(fb2b72a0-7a68-11d1-88f9-0080c7d771bf),
 dual,
 helpstring("IEventClass Interface"),
 pointer_default(unique)
]
interface IEventClass : IDispatch
{
[propget, id(1), helpstring("property EventClassID")]
HRESULT EventClassID([out,retval] BSTR* pbstrEventClassID);
[propput, id(1), helpstring("property EventClassID")]
HRESULT EventClassID([in] BSTR bstrEventClassID);

[propget, id(2), helpstring("property EventClassName")]
HRESULT EventClassName([out,retval] BSTR* pbstrEventClassName);
```

```
[propput, id(2), helpstring("property EventClassName")]
HRESULT EventClassName([in] BSTR bstrEventClassName);

[propget, id(3), helpstring("property OwnerSID")]
HRESULT OwnerSID([out,retval] BSTR* pbstrOwnerSID);
[propput, id(3), helpstring("property OwnerSID")]
HRESULT OwnerSID([in] BSTR bstrOwnerSID);

[propget, id(4), helpstring("property FiringInterfaceID")]
HRESULT FiringInterfaceID([out,retval] BSTR* pbstrFiringInterfaceID);
[propput, id(4), helpstring("property FiringInterfaceID")]
HRESULT FiringInterfaceID([in] BSTR bstrFiringInterfaceID);

[propget, id(5), helpstring("property Description")]
HRESULT Description([out,retval] BSTR* pbstrDescription);
[propput, id(5), helpstring("property Description")]
HRESULT Description([in] BSTR bstrDescription);

[propget, id(6), helpstring("property CustomConfigCLSID")]
HRESULT CustomConfigCLSID([out,retval] BSTR* pbstrCustomConfigCLSID);
[propput, id(6), helpstring("property CustomConfigCLSID")]
HRESULT CustomConfigCLSID([in] BSTR bstrCustomConfigCLSID);

[propget, id(7), helpstring("property TypeLib")]
HRESULT TypeLib([out,retval] BSTR* pbstrTypeLib);
[propput, id(7), helpstring("property TypeLib")]
HRESULT TypeLib([in] BSTR bstrTypeLib);
};


[
 object,
 uuid(fb2b72a1-7a68-11d1-88f9-0080c7d771bf),
 dual,
 helpstring("IEventClass2 Interface"),
 pointer default(unique)
]
interface IEventClass2 : IEventClass
{
[id(8), propget, helpstring("property PublisherID")]
HRESULT PublisherID([out,retval] BSTR* pbstrPublisherID);
[id(8), propput, helpstring("property PublisherID")]
HRESULT PublisherID([in] BSTR bstrPublisherID);

[id(9), propget, helpstring("property MultiInterfacePublisherFilterCLSID")]
HRESULT MultiInterfacePublisherFilterCLSID([out,retval] BSTR* pbstrPubFilCLSID);
[id(9), propput, helpstring("property MultiInterfacePublisherFilterCLSID")]
HRESULT MultiInterfacePublisherFilterCLSID([in] BSTR bstrPubFilCLSID);

[id(10), propget, helpstring("property AllowInprocActivation")]
HRESULT AllowInprocActivation([out,retval]BOOL* pfAllowInprocActivation);
[id(10), propput, helpstring("property AllowInprocActivation")]
HRESULT AllowInprocActivation([in]BOOL fAllowInprocActivation);

[id(11), propget, helpstring("property FireInParallel")]
HRESULT FireInParallel([out,retval]BOOL* pfFireInParallel);
[id(11), propput, helpstring("property FireInParallel")]
HRESULT FireInParallel([in]BOOL fFireInParallel);
}


[
 object,
 uuid(F4A07D63-2E25-11D1-9964-00C04FBBB345),
 helpstring("IEnumEventObject Interface"),
```

```
 pointer default(unique)
]
interface IEnumEventObject : IUnknown
{
[id(1), helpstring("method Clone")]
HRESULT Clone([out] IEnumEventObject** ppInterface);

[id(3), helpstring("method Next")]
HRESULT Next([in] ULONG cReqElem,
 [out,size is(cReqElem), length is(*cRetElem)] IUnknown** ppInterface,
 [out] ULONG* cRetElem);

[id(4), helpstring("method Reset")]
HRESULT Reset();

[id(5), helpstring("method Skip")]
HRESULT Skip([in] ULONG cSkipElem);
};


[
 object,
 uuid(f89ac270-d4eb-11d1-b682-00805fc79216),
 dual,
 helpstring("IEventObjectCollection Interface"),
 pointer_default(unique)
]
interface IEventObjectCollection : IDispatch
{
[id(DISPID NEWENUM), propget, restricted, helpstring("Create new IEnumVARIANT")]
HRESULT  NewEnum([out,retval] IUnknown** ppUnkEnum);

[id(DISPID_VALUE), propget]
HRESULT Item([in] BSTR objectID, [out,retval] VARIANT* pItem);

[id(1), propget, helpstring("Create new IEnumEventObject")]
HRESULT NewEnum([out,retval] IEnumEventObject** ppEnum);

[id(2), propget, helpstring("Number of items in the collection")]
HRESULT Count([out,retval] long* pCount);

[id(3), helpstring("Add an item to the collection")]
HRESULT Add([in] VARIANT* item, [in] BSTR objectID);

[id(4), helpstring("Remove an item from the collection")]
HRESULT Remove([in] BSTR objectID);
}


[
 object,
 uuid(4A6B0E15-2E38-11D1-9965-00C04FBBB345),
 dual,
 helpstring("IEventSubscription Interface"),
 pointer_default(unique)
]
interface IEventSubscription : IDispatch
{
[propget, id(1), helpstring("property SubscriptionID")]
HRESULT SubscriptionID([out,retval] BSTR* pbstrSubscriptionID);
[propput, id(1), helpstring("property SubscriptionID")]
HRESULT SubscriptionID([in] BSTR bstrSubscriptionID);

[propget, id(2), helpstring("property SubscriptionName")]
HRESULT SubscriptionName([out,retval] BSTR* pbstrSubscriptionName);
```

```
[propput, id(2), helpstring("property SubscriptionName")]
HRESULT SubscriptionName([in] BSTR bstrSubscriptionName);

[propget, id(3), helpstring("property PublisherID")]
HRESULT PublisherID([out,retval] BSTR* pbstrPublisherID);
[propput, id(3), helpstring("property PublisherID")]
HRESULT PublisherID([in] BSTR bstrPublisherID);

[propget, id(4), helpstring("property EventClassID")]
HRESULT EventClassID([out,retval] BSTR* pbstrEventClassID);
[propput, id(4), helpstring("property EventClassID")]
HRESULT EventClassID([in] BSTR bstrEventClassID);

[propget, id(5), helpstring("property MethodName")]
HRESULT MethodName([out,retval] BSTR* pbstrMethodName);
[propput, id(5), helpstring("property MethodName")]
HRESULT MethodName([in] BSTR bstrMethodName);

[propget, id(6), helpstring("property SubscriberCLSID")]
HRESULT SubscriberCLSID([out,retval] BSTR* pbstrSubscriberCLSID);
[propput, id(6), helpstring("property SubscriberCLSID")]
HRESULT SubscriberCLSID([in] BSTR bstrSubscriberCLSID);

[propget, id(7), helpstring("property SubscriberInterface")]
HRESULT SubscriberInterface([out,retval] IUnknown** ppSubscriberInterface);
[propput, id(7), helpstring("property SubscriberInterface")]
HRESULT SubscriberInterface([in] IUnknown* pSubscriberInterface);

[propget, id(8), helpstring("property PerUser")]
HRESULT PerUser([out,retval] BOOL* pfPerUser);
[propput, id(8), helpstring("property PerUser")]
HRESULT PerUser([in] BOOL fPerUser);

[propget, id(9), helpstring("property OwnerSID")]
HRESULT OwnerSID([out,retval] BSTR* pbstrOwnerSID);
[propput, id(9), helpstring("property OwnerSID")]
HRESULT OwnerSID([in] BSTR bstrOwnerSID);

[propget, id(10), helpstring("property Enabled")]
HRESULT Enabled([out,retval] BOOL* pfEnabled);
[propput, id(10), helpstring("property Enabled")]
HRESULT Enabled([in] BOOL fEnabled);

[propget, id(11), helpstring("property Description")]
HRESULT Description([out,retval] BSTR* pbstrDescription);
[propput, id(11), helpstring("property Description")]
HRESULT Description([in] BSTR bstrDescription);

[propget, id(12), helpstring("property MachineName")]
HRESULT MachineName([out,retval] BSTR* pbstrMachineName);
[propput, id(12), helpstring("property MachineName")]
HRESULT MachineName([in] BSTR bstrMachineName);

[id(13), helpstring("method GetPublisherProperty")]
HRESULT GetPublisherProperty([in] BSTR bstrPropertyName,
[out,retval] VARIANT* propertyValue);
[id(14), helpstring("method PutPublisherProperty")]
HRESULT PutPublisherProperty([in] BSTR bstrPropertyName,
[in] VARIANT* propertyValue);
[id(15), helpstring("method RemovePublisherProperty")]
HRESULT RemovePublisherProperty([in] BSTR bstrPropertyName);
[id(16), helpstring("method GetPublisherPropertyCollection")]
HRESULT GetPublisherPropertyCollection([out,retval] IEventObjectCollection** collection);

[id(17), helpstring("method GetSubscriberProperty")]
```

```
HRESULT GetSubscriberProperty([in] BSTR bstrPropertyName,
  [out,retval] VARIANT* propertyValue);
[id(18), helpstring("method PutSubscriberProperty")]
HRESULT PutSubscriberProperty([in] BSTR bstrPropertyName,
  [in] VARIANT* propertyValue);
[id(19), helpstring("method RemoveSubscriberProperty")]
HRESULT RemoveSubscriberProperty([in] BSTR bstrPropertyName);
[id(20), helpstring("method GetSubscriberPropertyCollection")]
HRESULT GetSubscriberPropertyCollection([out,retval] IEventObjectCollection**
collection);

[id(21), propget, helpstring("property InterfaceID")]
HRESULT InterfaceID([out,retval] BSTR* pbstrInterfaceID);
[id(21), propput, helpstring("property InterfaceID")]
HRESULT InterfaceID([in] BSTR bstrInterfaceID);
};


[
 object,
 uuid(4A6B0E16-2E38-11D1-9965-00C04FBBB345),
 dual,
 helpstring("IEventSubscription2 Interface"),
 pointer default(unique)
]
interface IEventSubscription2 : IEventSubscription
{
[propget, id(22), helpstring("property FilterCriteria")]
HRESULT FilterCriteria([out,retval] BSTR* pbstrFilterCriteria);
[propput, id(22), helpstring("property FilterCriteria")]
HRESULT FilterCriteria([in] BSTR bstrFilterCriteria);

[propget, id(23), helpstring("property SubscriberMoniker")]
HRESULT SubscriberMoniker([out,retval] BSTR* pbstrMoniker);
[propput, id(23), helpstring("property SubscriberMoniker")]
HRESULT SubscriberMoniker([in] BSTR bstrMoniker);
}


[
 object,
 uuid(7FB7EA43-2D76-4ea8-8CD9-3DECC270295E),
 dual,
 helpstring("IEventClass3 Interface"),
 pointer default(unique)
]
interface IEventClass3 : IEventClass2
{
[id(12), propget, helpstring("property EventClassPartitionID")]
HRESULT EventClassPartitionID([out,retval] BSTR* pbstrEventClassPartitionID);
[id(12), propput, helpstring("property EventClassPartitionID")]
HRESULT EventClassPartitionID([in] BSTR bstrEventClassPartitionID);

[id(13), propget, helpstring("property EventClassApplicationID")]
HRESULT EventClassApplicationID([out,retval] BSTR* pbstrEventClassApplicationID);
[id(13), propput, helpstring("property EventClassApplicationID")]
HRESULT EventClassApplicationID([in] BSTR bstrEventClassApplicationID);
}

[
 object,
 uuid(FBC1D17D-C498-43a0-81AF-423DDD530AF6),
 dual,
 helpstring("IEventSubscription3 Interface"),
 pointer_default(unique)
```

```
]
interface IEventSubscription3 : IEventSubscription2
{
[propget, id(24), helpstring("property EventClassPartitionID")]
HRESULT EventClassPartitionID([out,retval] BSTR* pbstrEventClassPartitionID);
[propput, id(24), helpstring("property EventClassPartitionID")]
HRESULT EventClassPartitionID([in] BSTR bstrEventClassPartitionID);

[propget, id(25), helpstring("property EventClassApplicationID")]
HRESULT EventClassApplicationID([out,retval] BSTR* pbstrEventClassApplicationID);
[propput, id(25), helpstring("property EventClassApplicationID")]
HRESULT EventClassApplicationID([in] BSTR bstrEventClassApplicationID);

[propget, id(26), helpstring("property SubscriberPartitionID")]
HRESULT SubscriberPartitionID([out,retval] BSTR* pbstrSubscriberPartitionID);
[propput, id(26), helpstring("property SubscriberPartitionID")]
HRESULT SubscriberPartitionID([in] BSTR bstrSubscriberPartitionID);

[propget, id(27), helpstring("property SubscriberApplicationID")]
HRESULT SubscriberApplicationID([out,retval] BSTR* pbstrSubscriberApplicationID);
[propput, id(27), helpstring("property SubscriberApplicationID")]
HRESULT SubscriberApplicationID([in] BSTR bstrSubscriberApplicationID);

};

[
 object,
 uuid(99CC098F-A48A-4e9c-8E58-965C0AFC19D5),
 dual,
 helpstring("IEventSystem2 Interface"),
 pointer default(unique)
]
interface IEventSystem2 : IEventSystem
{
[id(7), helpstring("method GetVersion")]
HRESULT GetVersion([out] int* pnVersion);
[id(8), helpstring("method VerifyTransientSubscribers")]
HRESULT VerifyTransientSubscribers();
}


[
 uuid(a0e8f27a-888c-11d1-b763-00c04fb926af),
 pointer_default(unique)
]
interface IEventSystemInitialize : IUnknown
{
 HRESULT SetCOMCatalogBehaviour(BOOL bRetainSubKeys);
};
```

7 Appendix B: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows Server 2008
- Windows Vista
- Windows Server 2003
- Windows XP
- Windows 2000

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 3.1.1.1: TypeLibrary specifies a registered OLE automation type library on Windows platforms. For more information on type libraries, see [MSDN-ITypeLib].

<2> Section 3.1.1.1: More information on MultiInterfacePublisherFilterCLSID property of event classes on Windows platforms is available under "Event Filtering" in [MSDN-COM+ Events].

<3> Section 3.1.1.1: EventClassPartitionID specifies a COM+ partition. For more information on COM+ partitions, see [MSDN-COM+]. This property is not available on Windows 2000.

<4> Section 3.1.1.1: EventClassApplicationID specifies a COM+ Application. For more information on COM+ applications, see [MSDN-COM+]. This property is not available on Windows 2000.

<5> Section 3.1.1.2: More information on the FilterCriteria property of subscriptions on Windows platforms is available under "Event Filtering" in [MSDN-COM+ Events].

<6> Section 3.1.4.1.1: **IEventClass3** is not supported on Windows 2000.

<7> Section 3.1.4.1.1: **IEventSubscription3** is not supported on Windows 2000.

<8> Section 3.1.4.1.5: **IEventClass3** is not supported on Windows 2000.

<9> Section 3.1.4.1.5: **IEventSubscription3** is not supported on Windows 2000.

<10> Section 3.1.4.2: Opnums reserved for local use apply to Windows as follows:

Opnum	Description
17-18	Just returns ERROR_NOT_IMPLEMENTED. It is never used.

<11> Section 3.1.4.7: **IEventClass3** is not supported on Windows 2000.

<12> Section 3.1.4.9: **IEventSubscription3** is not supported on Windows 2000.

<13> Section 3.1.4.10: **IEventSystem2** is not supported on Windows 2000.

<14> Section 3.1.4.11.1: For historical reasons, Windows client applications call this method anyway.

8 Index