[MS-COMT]:
Component Object Model Plus (COM+) Tracker Service Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
07/20/2007	0.1	Major	MCPP Milestone 5 Initial Availability
09/28/2007	0.1.1	Editorial	Revised and edited the technical content.
10/23/2007	0.2	Minor	Updated the technical content.
11/30/2007	0.2.1	Editorial	Revised and edited the technical content.
01/25/2008	0.2.2	Editorial	Revised and edited the technical content.
03/14/2008	0.2.3	Editorial	Revised and edited the technical content.
05/16/2008	0.2.4	Editorial	Revised and edited the technical content.
06/20/2008	1.0	Major	Updated and revised the technical content.
07/25/2008	1.1	Minor	Updated the technical content.
08/29/2008	1.2	Minor	Updated the technical content.
10/24/2008	1.3	Minor	Updated the technical content.
12/05/2008	1.4	Minor	Updated the technical content.
01/16/2009	1.4.1	Editorial	Revised and edited the technical content.
02/27/2009	2.0	Major	Updated and revised the technical content.
04/10/2009	2.1	Minor	Updated the technical content.
05/22/2009	2.1.1	Editorial	Revised and edited the technical content.
07/02/2009	2.1.2	Editorial	Revised and edited the technical content.
08/14/2009	2.1.3	Editorial	Revised and edited the technical content.
09/25/2009	2.2	Minor	Updated the technical content.
11/06/2009	2.2.1	Editorial	Revised and edited the technical content.
12/18/2009	2.2.2	Editorial	Revised and edited the technical content.
01/29/2010	2.2.3	Editorial	Revised and edited the technical content.
03/12/2010	2.2.4	Editorial	Revised and edited the technical content.
04/23/2010	2.2.5	Editorial	Revised and edited the technical content.
06/04/2010	2.2.6	Editorial	Revised and edited the technical content.
07/16/2010	2.2.6	No change	No changes to the meaning, language, or formatting of the technical content.

Date	Revision History	Revision Class	Comments
08/27/2010	2.2.6	No change	No changes to the meaning, language, or formatting of the technical content.
10/08/2010	2.2.6	No change	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	2.2.6	No change	No changes to the meaning, language, or formatting of the technical content.
01/07/2011	2.2.6	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	2.2.6	No change	No changes to the meaning, language, or formatting of the technical content.
03/25/2011	2.2.6	No change	No changes to the meaning, language, or formatting of the technical content.
05/06/2011	2.2.6	No change	No changes to the meaning, language, or formatting of the technical content.
06/17/2011	2.3	Minor	Clarified the meaning of the technical content.

Contents

1 Introduction

This document specifies the Component Object Model Plus (COM+) Tracker Service Protocol (COMT), which allows clients to monitor running instances of **components**.

1.1 Glossary

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

activation
class identifier (CLSID)
dynamic endpoint
globally unique identifier (GUID)
Interface Definition Language (IDL)
little-endian
opnum
Unicode
Universal Naming Convention (UNC)
Universally Unique Identifier (UUID)

The following terms are specific to this document:

component: An indivisible unit of software functionality that a **class identifier (CLSID)** identifies.

component configuration: A particular **component** configuration.

component instance: An instantiation of a **component**.

conglomeration: A set of related **component configurations** that are identified by a **conglomeration identifier**.

conglomeration identifier: A **GUID** that identifies a **conglomeration**.

container identifier: A **GUID** that identifies an **instance container**.

container legacy identifier: A nonzero integer that identifies an **instance container**.

container pooling: Enabling a **conglomeration** to support multiple concurrent **instance containers**.

distinguished container: The first **instance container** that is created in a given **process**.

instance container: A container for the instantiation of **components** that are configured in a single **conglomeration**.

instance pooling: Enabling a **component instance** that is no longer active to return to a pool for reuse.

method: A **component**-defined operation that **component instances** are able to execute at the request of external entities.

method call: The act of a **component instance** executing a **method** as a result of a specific request from an external entity.

pausing: Temporarily disabling the creation of a new **component instance** in an **instance container**.

process: A conceptual context in which an **instance container** can be created. A **process** is identified by a **process identifier**.

process dump: A mechanism for automatically gathering debugging data for a **process** into a file.

process identifier: A nonzero integer that identifies a **process**.

recycling: Permanently disabling the creation of a new **component instance** in an **instance container**.

tracker event: A notification that a COM+ Tracker Service Protocol server sends to a client that contains relevant information about the status of **component instances** and **instance containers** on the server.

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

References to Microsoft Open Specification documents do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

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".

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

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

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

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

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

[RFC2781] Hoffman, P., and Yergeau, F., "UTF-16, an encoding of ISO 10646", RFC 2781, February 2000, http://www.ietf.org/rfc/rfc2781.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".

[MS-COMEV] Microsoft Corporation, "Component Object Model Plus (COM+) Event System Protocol Specification".

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

[MSDN-Applications] Microsoft Corporation, "Applications (COM+)", http://msdn.microsoft.com/en-us/library/ms686107.aspx

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

[MSDN-FILE] Microsoft Corporation, "Naming a File", http://msdn.microsoft.com/en-us/library/aa365247.aspx

[MSDN-MDWD] Microsoft Corporation, "MiniDumpWriteDump Function (Windows)", http://msdn.microsoft.com/en-us/library/ms680360(VS.85).aspx

[MSDN-Partitions] Microsoft Corporation, "Partitions", http://msdn.microsoft.com/en-us/library/ms679480.aspx

[UML] Object Management Group, "Unified Modeling Language", http://www.omg.org/technology/documents/formal/uml.htm

1.3 Overview

The COM+ Tracker Service Protocol enables remote clients to monitor instances of components running on a server. The server end of the protocol tracks the status of **component instances** and **instance containers** on the server and implements an interface that clients can use to poll for this status. It also optionally includes an event-driven notification system in which the client can supply (via another protocol) a callback interface for receiving **tracker events**. The server then calls the client's callback interface whenever new tracking data is available, for example, as a result of local events on the server.

1.3.1 Background

A component is an indivisible unit of software functionality. Examples of components include Distributed Component Object Model (DCOM) Remote Protocol object classes, as specified in [MS-DCOM], and COM+ Event System Protocol event classes, as specified in [MS-COMEV]. Each component known to the server is identified by a **GUID**, known as the **class identifier (CLSID)**.

A **component configuration** is a particular configuration of a component. Each component configuration tracked by a COMT server is associated with a **conglomeration**, a set of related component configurations that is identified by a GUID known as the **conglomeration identifier**. In general, it is possible for a component to have more than one component configuration on a server. However, a component can have only one component configuration in any given conglomeration. A

component configuration can be identified by the conglomeration identifier and the component CLSID.

A conglomeration is a set of related component configurations and is identified by a GUID, known as the conglomeration identifier. A component that has a component configuration in a conglomeration is said to be configured in that conglomeration.

1.3.2 Instantiation Concepts

A server typically provides local or remote mechanisms by which components can be instantiated. An example of a remote instantiation mechanism is DCOM **activation** (as specified in [MS-DCOM] section 1.3.1). An instantiation of a component is known as a component instance. Although the instantiation details may vary, the following conceptual steps are part of any instantiation that is tracked in the COM+ Tracker Service Protocol.

Through an implementation-specific mechanism, the COMT Protocol server associates the instantiation with a component configuration, which is associated with a conglomeration as described in section 1.3.1.

The COMT Protocol server finds an existing instance container for the conglomeration or creates a new instance container and then associates it with the conglomeration. An instance container is a conceptual container in which components that are configured in a single conglomeration can be instantiated.

The COMT Protocol server creates the component instances in the selected instance container.

An instance container is identified by a GUID, known as the **container identifier**. For historical reasons, an instance container can also be identified by a nonzero integer, known as the **container legacy identifier**.

A **process** is a conceptual context for the creation of instance containers. Instance containers for multiple conglomerations can be created within a process. However, a conglomeration can have only one instance container in any given process. The first instance container created in any given process is known as the **distinguished container** for that process. A process is identified by a nonzero integer, known as the **process identifier**.

The following Unified Modeling Language (UML) static structure diagram summarizes the relationships between components, component configurations, conglomerations, component instances, instance containers, and processes. For more information about UML, see [UML].

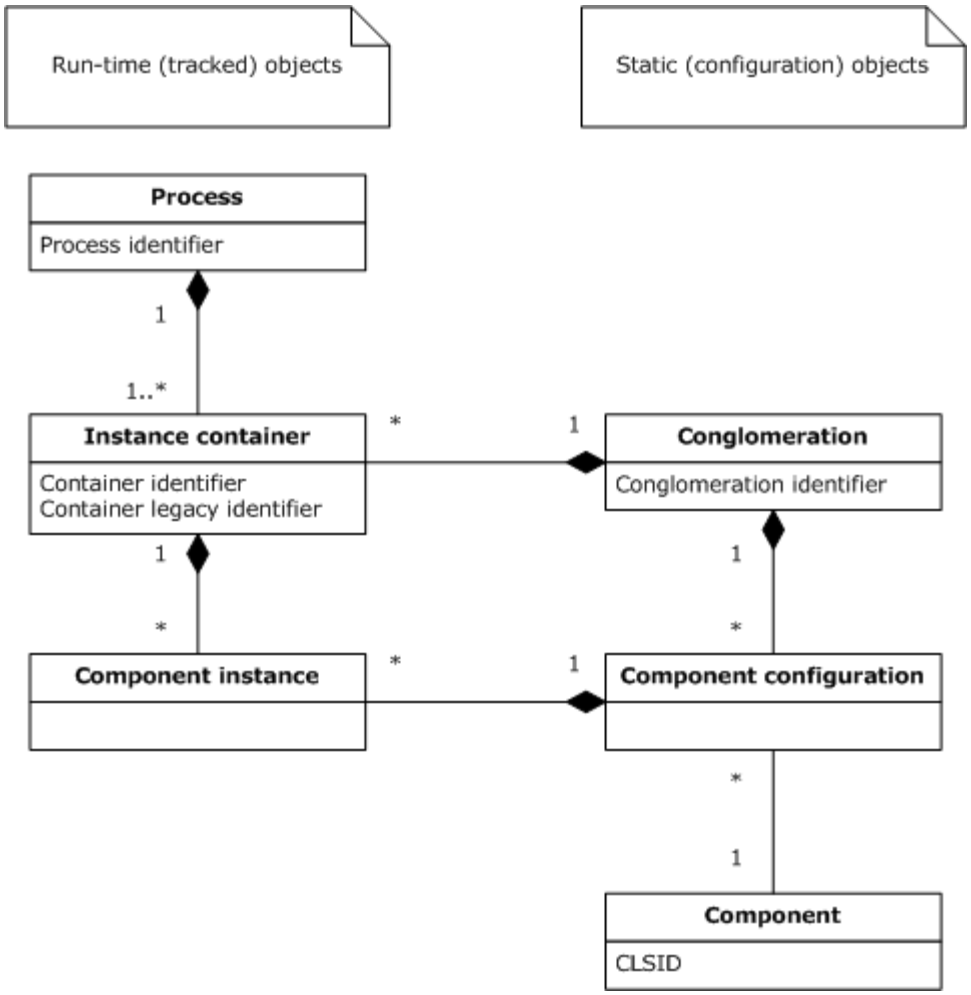


Figure 1: Relationships between static and run-time objects

1.3.3 Pooling

A server might provide, a single instance container, at most, for a conglomeration at any given time, or it might have the capability to provide multiple instance containers. Enabling a conglomeration to support multiple concurrent instance containers is known as **container pooling**. A typical use of container pooling is to increase scalability when contention for system resources within a single instance container is a limiting factor.

For historical reasons, parts of COMT are designed around the assumption that there is a one-to-one correspondence between conglomerations and instance containers and, therefore, the tracker cannot enable information about individual instance containers to be exchanged in cases where container pooling is in use.

Instance pooling refers to enabling component instances that are no longer active to return to a pool for reuse. A typical use of instance pooling is to reduce the performance penalty for the creation and destruction of short-lived component instances. A COMT server that enables instance pooling might track pooling behavior and expose separate statistics for pooled and active component instances.

1.3.4 Recycling and Pausing

Recycling refers to permanently disabling the creation of new component instances in an instance container. An instance container that is recycled shuts down as soon as the existing component instances in the container are destroyed. Recycling enables a problematic instance container to gradually drain its component instances, rather than being immediately and forcibly shut down. COMT enables clients to determine whether an instance container is being recycled.

Pausing refers to temporarily disabling the creation of new component instances in an instance container. COMT enables clients to determine whether an instance container is paused.

1.3.5 Activity Statistics

A COMT server optionally collects run-time activity statistics about component instances and instance containers.

Individual components define operations that component instances are able to execute at the request of external entities. A **method call** is the act of a component instance executing such an operation as a result of a specific request from an external entity. An example of a method call is a DCOM method call, as specified in [MS-DCOM]. COMT enables a client to obtain method call statistics for the components instantiated in an instance container, such as the number of successful method calls or the average time to complete a method call.

Servers that use a reference counting mechanism for component instances optionally collect statistics on the number of references to a component instance. The meaning of a reference is implementation-specific, but this information could be useful to administrators. The COMT enables a client to obtain reference statistics for the components that are instantiated in an instance container.

1.3.6 Polling and Tracker Events

The COMT enables two mechanisms by which clients can obtain tracking data: a push model and a pull model.

In the pull model, the COMT client invokes methods on the server interface to poll for tracking data. The pull model is most appropriate when tracking data is expected to change frequently, or when the client needs control over the frequency of communication.

In the push model, the client application implements a callback interface to receive tracker events. The push model is most appropriate when tracking data is expected to change infrequently, or when the server needs control over the frequency of communication. The COM+ Tracker Service Protocol does not provide a mechanism to register the callback interface with the server; it only defines the interface that the client registers. Hence, COMT requires that client applications use another protocol, such as the COM+ Event System Protocol [MS-COMEV], to register the COMT callback interface. For example, if a COMT server exposes tracker events as a COM+ Event System Protocol event class (as specified in [MS-COMEV] section 3.1.1.1), a client application could create a subscription (as specified in [MS-COMEV] section 3.1.1.2) to the event class and set the SubscriberInterface property to its callback interface (for more information, see **IEventSubscription::put_SubscriberInterface**, [MS-COMEV] section 3.1.4.4.14).

1.3.7 Process Dump

If tracking data received by an administrator or administration client application indicates that there may be a problem with the instance containers in a particular process, the administrator or application may wish to collect additional debugging data to investigate this problem. A mechanism

for automatically gathering additional debugging data for a process into a file is known as a **process dump**.

COMT enables clients to request a process dump for the process containing an instance container on the COMT server. The file format to be used for a process dump, as well as the nature and extent of the debugging data collected, are implementation-specific. However, typical data included in a process dump might be full or partial contents of the process's address space, information on the process's usage of system resources, and history of exceptional events that have occurred.

1.4 Relationship to Other Protocols

COMT is built on top of DCOM [MS-DCOM].

The COM+ Remote Administration Protocol [MS-COMA] also provides functionality for obtaining run-time information about instance containers. COMT makes this functionality obsolete by enabling clients to obtain a richer set of information, and by providing a push model.

Client applications that want to receive notifications via the push model also need to use another protocol, such as the Component Object Model Plus (COM+) Event System Protocol [MS-COMEV], to first register the COMT callback interface.

1.5 Prerequisites/Preconditions

COMT assumes that a client application that wants to receive tracker events by using the push model has previously registered a callback interface to the server by using some other mechanism; for example, the Component Object Model Plus (COM+) Event System Protocol, as specified in [MS-COMEV].

COMT assumes that a client application or administrator that wants to request a process dump to be written in a location other than the COMT server's default location recognizes the convention for paths in the COMT server's file system.

COMT assumes that a client application or administrator that wants to interpret debugging data from a process dump recognizes the file format in which this data will be written.

1.6 Applicability Statement

The COM+ Tracker Service Protocol is most appropriate for monitoring running instances of components when the tracking information is used for informational purposes. It is not appropriate when this information is required for correct behavior of a client application.

1.7 Versioning and Capability Negotiation

The COM+ Tracker Service Protocol has no versioning and capability negotiation functionality.

1.8 Vendor-Extensible Fields

The COM+ Tracker Service Protocol uses **HRESULT** values, as specified in [MS-ERREF] section 2.1. Vendors can define their own **HRESULT** values, provided that they set the C bit (0x20000000) for each vendor-defined value to indicate that the value is a customer code.

1.9 Standards Assignments

The following table lists well-known GUIDs in the COMT Protocol. These GUIDs were generated using the mechanism specified in [C706] section A.2.5.

Parameter	Value
DCOM CLSID for tracker service (CLSID_TrackerService)	{ECABAFB9-7F19-11D2-978E-0000F8757E2A}
DCOM CLSID for process dump service (CLSID_ProcessDump)	{ECABB0C4-7F19-11D2-978E-0000F8757E2A}
RPC Interface Identifier (IID) for IGetTrackingData interface	{B60040E0-BCF3-11D1-861D-0080C729264D}
RPC Interface Identifier (IID) for IComTrackingInfoEvents interface	{4E6CDCC9-FB25-4FD5-9CC5-C9F4B6559CEC}
RPC Interface Identifier (IID) for IProcessDump interface	{23C9DD26-2355-4FE2-84DE-F779A238ADBD}
OBJREF_CUSTOM unmarshaler CLSID for TrackingInfoCollection (CLSID_TrkInfoCollUnmarshal)	{ECABAFCD-7f19-11D2-978E-0000F8757E2A}
OBJREF_CUSTOM unmarshaler CLSID for TrackingInfoObject (CLSID_TrkInfoObjUnmarshal)	{ECABAFCE-7f19-11D2-978E-0000F8757E2A}

2 Messages

The following sections specify how COM+ Tracker Service Protocol messages are transported as well as COMT Protocol message syntax.

2.1 Transport

All COM+ Tracker Service Protocol messages are transported via DCOM, as specified in [MS-DCOM]. COMT uses the **dynamic endpoints** allocated and managed by the DCOM infrastructure.

2.2 Common Data Types

In addition to remote procedure call (RPC) base types and definitions specified in [C706] and [MS-RPCE], the following table defines additional data types.

Field types in packet diagrams are defined by the packet diagram and the field descriptions. All fields in packet diagrams use **little-endian** byte ordering, unless otherwise stated.

All extra padding bytes MUST be zero, unless otherwise stated, and MUST be ignored upon receipt.

This protocol uses the following types, as specified in [MS-DTYP] and [MS-OAUT].

Type	Reference
DWORD	As specified in [MS-DTYP] section 2.2.9
GUID	As specified in [MS-DTYP] section 2.3.2
HRESULT	As specified in [MS-DTYP] section 2.2.18
WCHAR	As specified in [MS-DTYP] section 2.2.59
BSTR	As specified in [MS-OAUT] section 2.2.23

2.2.1 CurlyBraceGuidString

The CurlyBraceGuidString type is a string representation of the GUID type, as specified in [MS-DTYP] section 2.3.2.3. The following is the Augmented Backus-Naur Form (ABNF) syntax, as referenced in [RFC4234], for this representation.

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

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

2.2.2 ContainerStatistics

The **ContainerStatistics** type represents activity statistics for an instance container.

```
typedef struct {
  DWORD cCalls;
  DWORD cComponentInstances;
  DWORD cComponents;
  DWORD cCallsPerSecond;
```

```
} ContainerStatistics;
```

cCalls: The number of method calls that the component instances perform in an instance container.

cComponentInstances: The number of component instances in an instance container.

cComponents: The number of distinct components currently instantiated in an instance container.

cCallsPerSecond: This SHOULD be set to a running average, over an implementation-specific time period,<1> of the number of method calls per second received by an instance container. Alternatively, an implementation MAY instead simply set this field to zero.

2.2.3 ContainerData

The **ContainerData** type represents run-time information for a conglomeration that has one or more instance containers on the server. The meanings of the fields in this structure depend on the number of instance containers that exist on the server for the conglomeration represented, as specified in the following section.

```
typedef struct {
  DWORD dwLegacyId;
  WCHAR wszApplicationIdentifier[40];
  DWORD dwProcessId;
  ContainerStatistics statistics;
} ContainerData;
```

dwLegacyId: The container legacy identifier of one of the instance containers, arbitrarily selected by the server, that exist for the conglomeration represented.

wszApplicationIdentifier: A null-terminated **Unicode** string that MUST contain the CurlyBraceGuidString (section 2.2.1) representation of a conglomeration identifier. Note that a null-terminated CurlyBraceGuidString is 39 Unicode characters, including the null character, and this field is 40 characters long. The final element in this array is unused. It SHOULD be set to 0 and MUST be ignored upon receipt.

dwProcessId: The process identifier of the process that contains one of the instance containers, arbitrarily selected by the server, that exist for the conglomeration represented.

statistics: A **ContainerStatistics (section 2.2.2)** structure with fields that contain statistics averaged across all instance containers that exist for the conglomeration represented.

2.2.4 ComponentData

The **ComponentData** type represents activity statistics for a component that has one or more component instances in an instance container.

```
typedef struct {
  GUID clsid;
  DWORD cTotalReferences;
  DWORD cBoundReferences;
```

```
  DWORD cPooledInstances;
  DWORD cInstancesInCall;
  DWORD dwResponseTime;
  DWORD cCallsCompleted;
  DWORD cCallsFailed;
} ComponentData;
```

clsid: The CLSID of the component.

cTotalReferences: An implementation-specific<2> count of the number of references to all component instances of the component. This MUST be set to 0xffffffff if the server does not track this information.<3>

cBoundReferences: The number of references to all active (not pooled) component instances of the component. This MUST be set to 0xffffffff if the server does not track this information.<4>

cPooledInstances: The number of pooled component instances of the component, if the server enables instance pooling. This MUST be set to 0xffffffff if the server does not track this information.<5>

cInstancesInCall: The number of component instances of the component that are currently performing a method call. This MUST be set to 0xffffffff if the server does not track this information.<6>

dwResponseTime: A value that indicates the average time, in milliseconds, it takes to complete method calls to component instances of the component. Calculation of this value is implementation-specific.<7> This MUST be set to 0xffffffff if the server does not track this information.<8>

cCallsCompleted: The number of method calls to component instances of the component that were successfully completed in an implementation-specific<9> time period. Whether a server considers a method call successfully completed is implementation-specific.<10> This MUST be set to 0xffffffff if the server does not track this information.<11>

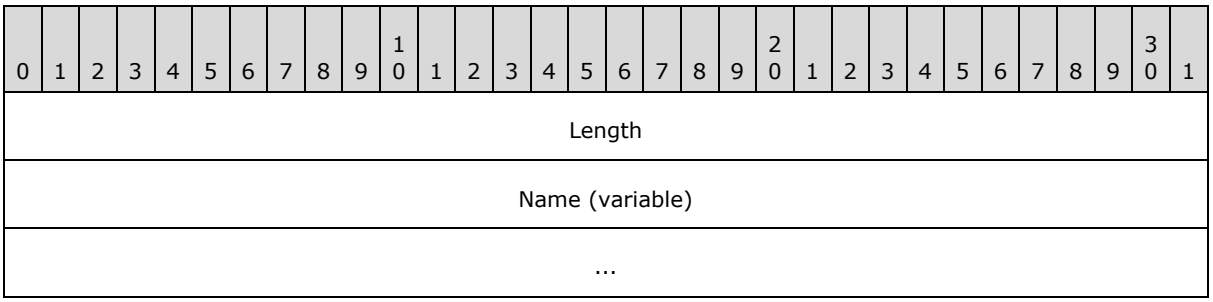
cCallsFailed: The number of method calls to component instances of the component that failed in an implementation-specific<12> time period. Whether a server considers a method call to have failed is implementation-specific.<13> This MUST be set to 0xffffffff if the server does not track this information.<14>

2.2.5 TrackingInfo Formats

The following sections specify the formats of structures related to the **IComTrackingInfoCollection::OnNewTrackingInfo** method (as specified in section 3.2.5.1.1).

2.2.5.1 LengthPrefixedName

The LengthPrefixedName type specifies an array of Unicode characters prefixed by the array length in characters.

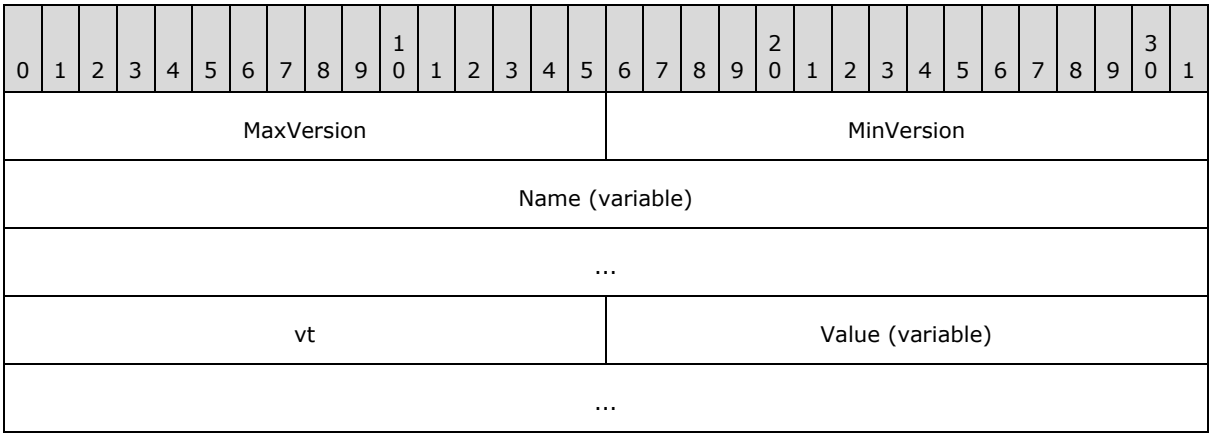


Length (4 bytes): An unsigned long that MUST contain the number of Unicode characters in the **Name** field and MUST NOT be zero.

Name (variable): This MUST contain an array of Unicode characters in UTF-16 encoding, as specified in [RFC2781]); the array SHOULD NOT end in a NULL terminator.

2.2.5.2 TrackingInfoPropertyValue

The TrackingInfoPropertyValue structure defines a single name/value pair.



MaxVersion (2 bytes): The major version number for this format; this field MUST be set to 0x0001.

MinVersion (2 bytes): The minor version number for this format; this field MUST be set to 0x0001.

Name (variable): A LengthPrefixedName (section 2.2.5.1) that contains the name of the UserProperty.

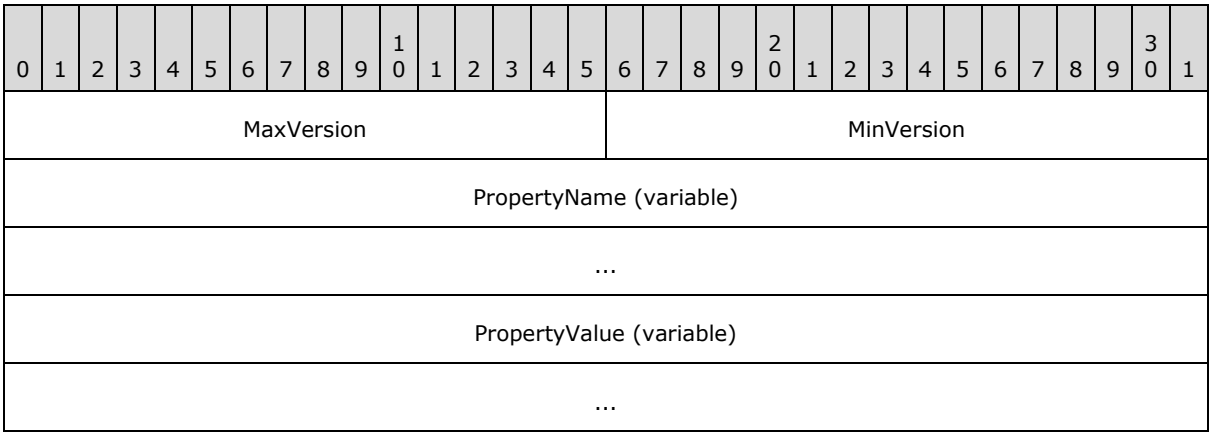
vt (2 bytes): The type of data contained in **Value**. It MUST be set to one of the following values.

Value	Meaning
0x0008	LengthPrefixedName (section 2.2.5.1)
0x000D	TrackingInfoCollection OBJREF_CUSTOM (section 2.2.5.5)
0x0013	An unsigned long integer.

Value (variable): The data for this name/value pair. The type of this field is specified by the **vt** field.

2.2.5.3 TrackingInfoProperty

The TrackingInfoProperty defines a structure for representing a property name/value pair.



MaxVersion (2 bytes): The major version of this marshaled format; this MUST be set to 0x0001.

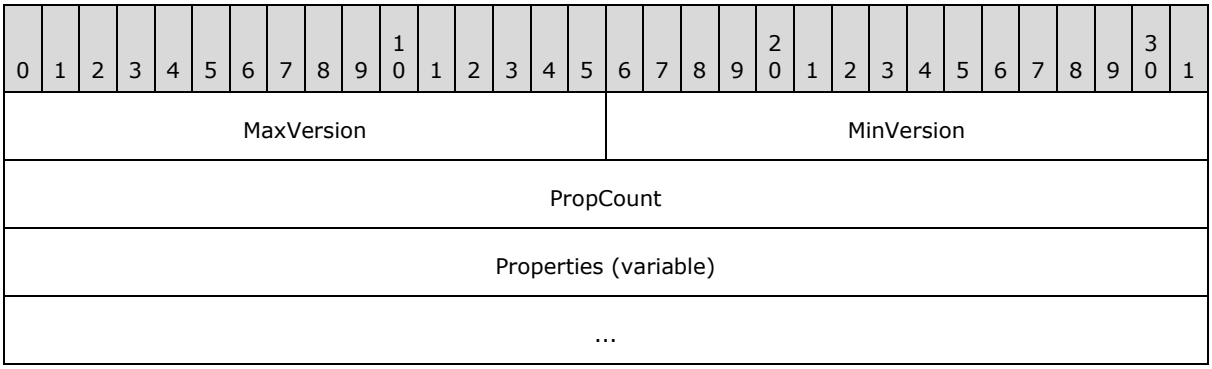
MinVersion (2 bytes): The minor version of this marshaled format; this MUST be set to 0x0001.

PropertyName (variable): A LengthPrefixedName (section 2.2.5.1) that contains the name of this property.

PropertyValue (variable): A TrackingInfoPropertyValue (section 2.2.5.2) that contains the value of this property.

2.2.5.4 TrackingInfoObject OBJREF_CUSTOM

The TrackingInfoObject MUST be marshaled using the OBJREF_CUSTOM format (as specified in [MS-DCOM] section 2.2.18.6). The **CLSID** field of the OBJREF_CUSTOM instance MUST be set to {ECABAFCE-7f19-11D2-978E-0000F8757E2A} (CLSID_TrkInfoObjUnmarshal). The format of the OBJREF_CUSTOM.pObjectData buffer for CLSID_TrkInfoObjUnmarshal is as follows.



MaxVersion (2 bytes): The major version of this marshaled format; this MUST be set to 0x0001.

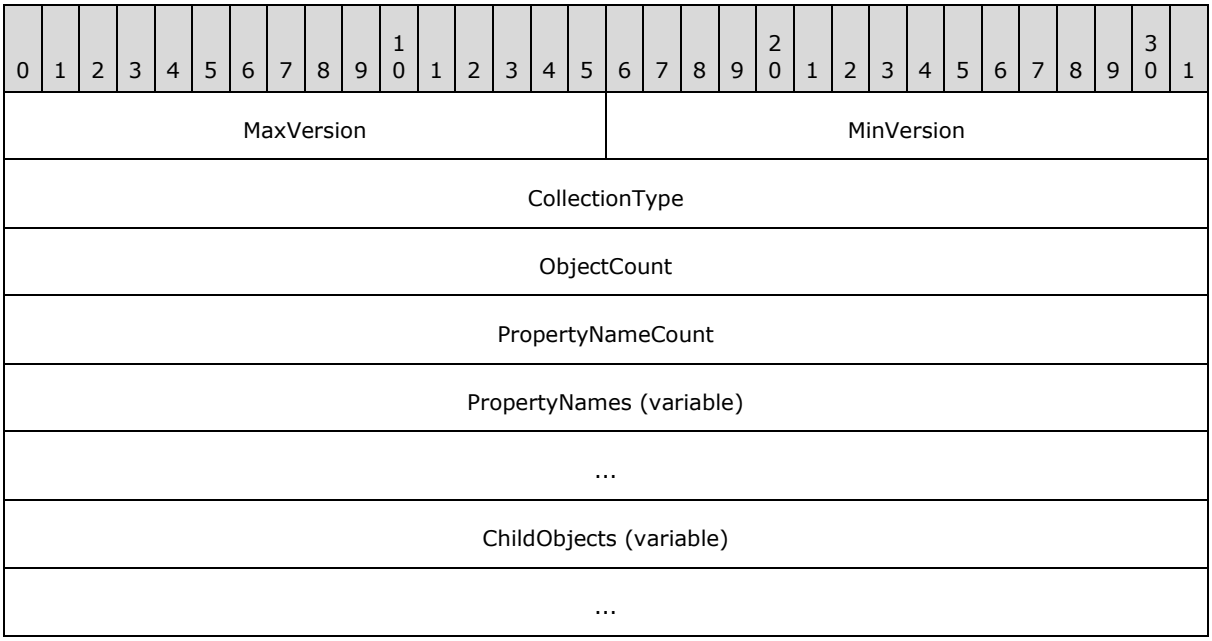
MinVersion (2 bytes): The minor version of this marshaled format; this MUST be set to 0x0001.

PropCount (4 bytes): The (unsigned) number of elements in the **Properties** field.

Properties (variable): An array of TrackingInfoProperty (section 2.2.5.3) structures.

2.2.5.5 TrackingInfoCollection OBJREF_CUSTOM

The TrackingInfoCollection MUST be marshaled using the OBJREF_CUSTOM format (as specified in [MS-DCOM] section 2.2.18.6). The **CLSID** field of the OBJREF_CUSTOM instance MUST be set to {ECABAFCD-7f19-11D2-978E-0000F8757E2A} (CLSID_TrkInfoCollUnmarshal). The format of the OBJREF_CUSTOM.pObjectData buffer for CLSID_TrkInfoCollUnmarshal is as follows.



MaxVersion (2 bytes): The major version of this marshaled format; this MUST be set to 0x0001.

MinVersion (2 bytes): The minor version of this marshaled format; this MUST be set to 0x0001.

CollectionType (4 bytes): The type of collection; this MUST be one of the following values:

Value	Meaning
TRKCOLL_PROCESSES 0x00000000	A collection of processes.
TRKCOLL_CONTAINERS 0x00000001	A collection of instance containers.

Value	Meaning
TRKCOLL_COMPONENTS 0x00000002	A collection of components.

ObjectCount (4 bytes): The (unsigned) number of elements in the **ChildObjects** field.

PropertyNameCount (4 bytes): The (unsigned) number of elements in the **Properties** field.

PropertyNames (variable): An array of LengthPrefixedName (section 2.2.5.1) that contains the descriptive names for the elements in the **ChildObjects** field.

ChildObjects (variable): An array of TrackingInfoObject (section 2.2.5.4) structures.

3 Protocol Details

The client side of this protocol is a pass-through. That is, no additional timers or other state is required on the client side of this protocol. Calls made by the higher-layer protocol or application are passed directly to the transport, and the results that the transport returns are passed directly back to the higher-layer protocol or application.

A client application initiates a conversation with a COM+ Tracker Service Protocol server in one of three ways:

- If the client application wants to poll for tracking information, it performs DCOM activation (as specified in [MS-DCOM], section 3.2.4.1.1) of the tracker service CLSID (CLSID_TrackerService), as 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 that it supports. When complete, the client application performs a **release** on the interface pointer.
- If the client application wants to receive tracker events, it uses any implementation-specific mechanism <15> to supply an **IComTrackingInfoEvents (section 3.2.5.1)** callback interface to the server. Thereafter, the COMT server sends tracker events as a result of implementation-specific local events, and the client application receives these events in the form of DCOM calls to **OnNewTrackingInfo()** on the client application's IComTrackingInfoEvents interface. The credentials that are used for these calls are COMT server implementation-specific<16>. The conversation can be terminated by either the client application or the COMT server.
- If the client application wants to request a process dump, it performs a DCOM activation ([MS-DCOM] section 3.2.4.1.1) of the process dump service CLSID (CLSID_ProcessDump) as specified in section 1.9. Depending on the behavior of the underlying DCOM client implementation, the client application might need to override the default impersonation level to use RPC_C_IMPL_LEVEL_IMPERSONATE (as specified in [MS-RPCE] section 2.2.1.1.9). 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 that it supports. When done, the client application performs a release operation on the interface pointer.

When the client application no longer wants to receive tracker events, the client application uses any implementation-specific mechanism<17> to request that the server stop sending events and release the client's **IComTrackingInfoEvents** interface.

When the server no longer has to send tracker events, it performs a release on the interface pointer to **IComTrackingInfoEvents**.

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.

Conglomeration Table: A table of conglomerations. Each entry has the following fields.

- **Conglomeration Identifier:** The conglomeration identifier.

- **Partition ID:** An implementation-specific GUID that identifies a conceptual group (or type) of conglomerations to which this conglomeration belongs.<18>
- **Instance Container Table:** A table of instance containers that exist for the conglomeration. Each entry has the following fields.
  - **Container Identifier:** The container identifier.
  - **Container Legacy Iidentifier:** The container legacy identifier.
  - **Process Identifier:** The process identifier that contains the instance container.
  - **Container Statistics:** The container statistics, as specified in section 2.2.2.
  - **Component Table:** A table of components that are instantiated in the container. Each component entry has the following field.
    - **Component Data:** The component data, as specified in section 2.2.4.

3.1.2 Timers

None.

3.1.3 Initialization

None.

3.1.4 Message Processing Events and Sequencing Rules

3.1.4.1 IGetTrackingData

The **IGetTrackingData** interface provides methods for a client to poll for tracking information. This interface inherits from IUnknown, as specified in [MS-DCOM] section 3.1.1.5.8. 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_TrackerService (as specified in section 1.9) by using the UUID {B60040E0-BCF3-11D1-861D-0080C729264D} for this interface.

The **IGetTrackingData** interface includes the following methods beyond those of IUnknown.

Methods in RPC Opnum Order

Method	Description
Opnum3NotUsedOnWire	Reserved for local use. Opnum: 3
GetContainerData	Returns tracking information for instance containers. Opnum: 4
GetComponentDataByContainer	Returns tracking information for components by instance container. Opnum: 5
GetComponentDataByContainerAndCLSID	Returns tracking information for a component by instance

Method	Description
	container and CLSID. Opnum: 6
Opnum7NotUsedOnWire	Reserved for local use. Opnum: 7

In the preceding table, the phrase "Reserved for local use" means that the client MUST NOT send the **opnum** and that the server behavior is undefined<19> because it does not affect interoperability.

All methods MUST NOT throw exceptions.

3.1.4.1.1 GetContainerData (Opnum 4)

A client calls this method to obtain tracking information for instance containers across all conglomerations.

```
HRESULT GetContainerData(
  [out] DWORD* nContainers,
  [out, size_is(,(*nContainers))]
    ContainerData** aContainerData
);
```

nContainers: A pointer to a variable that, upon successful completion, MUST contain the number of elements in *aContainerData*.

aContainerData: A pointer to a variable that, upon successful completion, MUST contain an array of zero or more **ContainerData (section 2.2.3)** structures. An array with zero elements MUST be represented by null.

Return Values: This method MUST return S_OK (0x00000000) on success and a failure result (as specified in [MS-ERREF] section 2.1) on failure.

When this method is invoked, the server MUST attempt to return an array of **ContainerData** structures, one for each instance container tracked by the server, or fail the call if it cannot.

3.1.4.1.2 GetComponentDataByContainer (Opnum 5)

A client calls this method to obtain tracking information for components that have one or more component instances in a given instance container.

```
HRESULT GetComponentDataByContainer(
  [in] DWORD idContainer,
  [out] DWORD* nComponents,
  [out, size_is(,*nComponents)] ComponentData** aComponentData
);
```

idContainer: The container legacy identifier of an instance container.

nComponents: A pointer to a variable that, upon successful completion, MUST contain the number of elements in *aComponentData*.

aComponentData: A pointer to a variable that, upon successful completion, MUST contain an array of zero or more **ComponentData (section 2.2.4)** structures. An array with zero elements MUST be represented by null.

Return Values: This method MUST return S_OK (0x00000000) on success and a failure result (as specified in [MS-ERREF] section 2.1) on failure.

When this method is invoked, the server MUST verify that the *idContainer* parameter identifies a tracked instance container and fail the call if not. The server then MUST attempt to return an array of zero or more **ComponentData** structures, one for each distinct component instance instantiated in the instance container, and fail the call if it cannot.

3.1.4.1.3 GetComponentDataByContainerAndCLSID (Opnum 6)

A client calls this method to obtain tracking information for a single component that has component instances in an instance container.

```
HRESULT GetComponentDataByContainerAndCLSID(
  [in] DWORD idContainer,
  [in] GUID clsid,
  [out] ComponentData** ppComponentData
);
```

idContainer: The container legacy identifier of an instance container.

clsid: A pointer to the CLSID of a component.

ppComponentData: A pointer to a variable that, upon successful completion, MUST contain a pointer to a single **ComponentData (section 2.2.4)** structure.

Return Values: This method MUST return S_OK (0x00000000) on success and a failure result (as specified in [MS-ERREF] section 2.1) on failure.

When this method is invoked, the server MUST verify that the *idContainer* parameter identifies a tracked instance container and that the CLSID received in the *clsid* parameter identifies a component that is instantiated in that instance container. If not, the server MUST fail the call; otherwise, the server MUST return a single **ComponentData** structure that represents the component instantiated in the instance container and return success.

3.1.4.2 IProcessDump

The **IProcessDump** interface provides methods for a client to request a process dump of a process containing an instance container on the COMT server. This interface inherits 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_ProcessDump (as specified in section 1.9) by using the UUID {23C9DD26-2355-4FE2-84DE-F779A238ADBD} for this interface.

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

Methods in RPC Opnum Order

Method	Description
IsSupported	Returns a result indicating whether or not process dump is supported. Opnum: 7
DumpProcess	Requests a process dump. Opnum: 8

All methods MUST NOT throw exceptions.

3.1.4.2.1 IsSupported (Opnum 7)

This method is called by a client to determine whether or not the COMT server supports process dump.

```
[id(1)] HRESULT IsSupported();
```

This method has no parameters.

Return Values: This method returns S_OK (0x00000000) if the COMT server supports process dump, and MUST return S_FALSE (0x00000001) if not.

3.1.4.2.2 DumpProcess (Opnum 8)

This method is called by a client to request a process dump for the process containing a particular instance container.

```
[id(2)] HRESULT DumpProcess(
  [in] BSTR bstrContainerID,
  [in] BSTR bstrDirectory,
  [in] DWORD dwMaxFiles,
  [out, retval] BSTR* pbstrDumpFile
);
```

bstrContainerID: The CurlyBraceGuidString (section 2.2.1) representation of a container identifier for a distinguished container.

bstrDirectory: Either a path, in the convention of the server's file system, to a location in which the file produced by process dump is to be written, or NULL to indicate that the client wants the COMT server to write the file to an implementation-specific default location.

dwMaxFiles: The maximum number of process dump files associated with the conglomeration of the instance container identified by the *bstrContainerID* parameter that the client requests the COMT server to leave in the location specified by the *bstrDirectory* parameter before the server begins deleting previously written files. A value of 0x00000000 indicates that the COMT server is to use an implementation-specific default limit.

pbstrDumpFile: A pointer to a variable that, upon successful completion, contains a fully qualified path, in the convention of the server's file system, to the process dump file written.

Return Values: This method MUST return S_OK (0x00000000) on success and a failure result (as specified in [MS-ERREF] section 2.1) on failure.

When this method is invoked, if the server does not support process dump, it MUST simply return E_NOTIMPL (0x80004001). Otherwise, the server MUST verify that the *bstrContainerID* parameter identifies a tracked instance container, and that this instance container is a distinguished container, and fail the call if not.

The server then MUST verify that the *bstrDirectory* parameter, if not NULL, is in a file path syntax supported<20> by the server and fail the call if not. If *bstrDirectory* is not NULL, the server MUST select the location specified by this parameter as the dump file location, the use of which is described later in this section. Otherwise, the server MUST select an implementation-specific <21> default location.

The server then MUST select the dump file limit, the use of which is described later in this section, as the value specified by the *dwMaxFiles* parameter if this parameter is nonzero, and an implementation-specific <22> default limit if this parameter is zero.

The server SHOULD<23> impersonate the client while performing any file access specified in the remainder of the method behavior.

The server then SHOULD attempt to determine the number of previously written process dump files that are associated with the conglomeration of the instance container identified in *bstrContainerID* in the dump file location, and MAY fail the call if it cannot do so. If the number of previously written files is greater than or equal to the dump file limit, the server SHOULD attempt to delete at least one of the previously written files, and MAY fail the call if it cannot do so. If more than one previously written file exists, the server SHOULD attempt to determine which of these files was written least recently, and SHOULD select that file for deletion.

The server then MUST attempt to perform an implementation-specific <24> process dump procedure by writing a file to the dump file location, and fail the call if it cannot.

The server then MUST set the *pbstrDumpFile* parameter to the fully qualified path to the file written, and return success.

3.1.5 Timer Events

None.

3.1.6 Other Local Events

None.

3.2 Client Details

A client that uses only the polling capabilities that **IGetTrackingData (section 3.1.4.1)** provides is simply a pass-through.

A client that is to receive tracker events MUST implement the **IComTrackingInfoEvents (section 3.2.5.1)** interface.

3.2.1 Abstract Data Model

None.

3.2.2 Timers

None.

3.2.3 Initialization

None.

3.2.4 Higher-Layer Triggered Events

Calls that the higher-layer protocol or application make MUST be passed directly to the transport, and the results that the transport returns MUST be passed directly back to the higher-layer protocol or application.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 IComTrackingInfoEvents

The **IComTrackingInfoEvents** interface provides a method for a server to send the client tracker events. This interface inherits from IUnknown, as specified in [MS-DCOM] section 3.1.1.5.8. The version for this interface is 0.0.

This interface includes the following method beyond those of IUnknown.

Methods in RPC Opnum Order

Method	Description
OnNewTrackingInfo	Handles new tracking info. Opnum: 3

This method MUST NOT throw exceptions.

3.2.5.1.1 OnNewTrackingInfo (Opnum 3)

The **OnNewTrackingInfo** method handles a tracker event from the server.

```
HRESULT OnNewTrackingInfo(
  [in] IUnknown* pToplevelCollection
);
```

pToplevelCollection: An interface pointer of a DCOM object. This MUST be a TrackingInfoCollection OBJREF_CUSTOM (section 2.2.5.5). This collection SHOULD be of type TRKCOLL_PROCESSES (as specified in section 2.2.5.5), and each TrackingInfoObject in the collection SHOULD represent a process on the server. Each process TrackingInfoObject structure SHOULD have the following properties.

Property name	vt value	Meaning
ProcessID	0x00000013	The process identifier.
ExeName	0x00000008	Implementation-specific identifier of the type of process.<25>
Paused	0x00000013	TRUE (0x00000001) if the distinguished container for the process is paused; otherwise, FALSE (0x00000000).
Recycling	0x00000013	TRUE (0x00000001) if the distinguished instance container for the

Property name	vt value	Meaning
		process is recycled; otherwise, FALSE (0x00000000).
IsService	0x00000013	TRUE (0x00000001) if the process is a system service; otherwise, FALSE (0x00000000). The definition of system service is implementation-specific.<26>
Applications	0x0000000D	A TrackingInfoCollection (section 2.2.5.5) of type TRKCOLL_CONTAINERS that represents the instance containers in the process.

Each instance container TrackingInfoObject SHOULD have the following properties.

Property name	vt value	Meaning
ApplicationID	0x00000008	The CurlyBraceGuidString (section 2.2.1) representation of the conglomeration identifier of the conglomeration that is associated with the instance container.
ApplInstanceID	0x00000008	The CurlyBraceGuidString (section 2.2.1) representation of the container identifier of the instance container.
ApplicationType	0x00000013	An implementation-specific<27> integer that identifies the type of instance container.
PartitionID	0x00000008	The CurlyBraceGuidString (section 2.2.1) representation of the Partition ID of the conglomeration.
Name	0x00000008	An implementation-specific<28> Unicode string that provides a human-readable name for the conglomeration that is associated with the instance container.
Components	0x0000000D	A TrackingInfoCollection (section 2.2.5.5) of type TRKCOLL_COMPONENTS that represents the components instantiated in the instance container.

Each component TrackingInfoObject SHOULD have the following properties:

Property name	vt value	Meaning
CLSID	0x00000008	The CurlyBraceGuidString (section 2.2.1) representation of the CLSID of the component.
Objects	0x00000013	The number of component instances for the component in an instance container.
Activated	0x00000013	The number of active component instances for the component in an instance container.
Pooled	0x00000013	The number of pooled component instances for the component in an instance container.
InCall	0x00000013	The number of component instances for the component in an instance container that are currently performing a method call.

Property name	vt value	Meaning
CallTime	0x00000013	A value that indicates the average amount of time, in milliseconds, that it takes to complete method calls to component instances for the component. The calculation of this value is implementation-specific.<29>
Name	0x00000008	An implementation-specific Unicode string that provides a human-readable name for the component.<30>

Return Values: The **OnNewTrackingInfo** method MUST return S_OK (0x00000000) on success and a failure result (as specified in [MS-ERREF] section 2.1) on failure.

Upon receiving a call to the **OnNewTrackingInfo** method, the client MUST attempt to unmarshal the TrackingInfoCollection OBJREF_CUSTOM received in *pToplevelCollection* and fail the call if it cannot. The client SHOULD then return before performing any further actions. Any further implementation-specific processing SHOULD be done asynchronously.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

4 Protocol Examples

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

4.1 Polling for Tracking Data

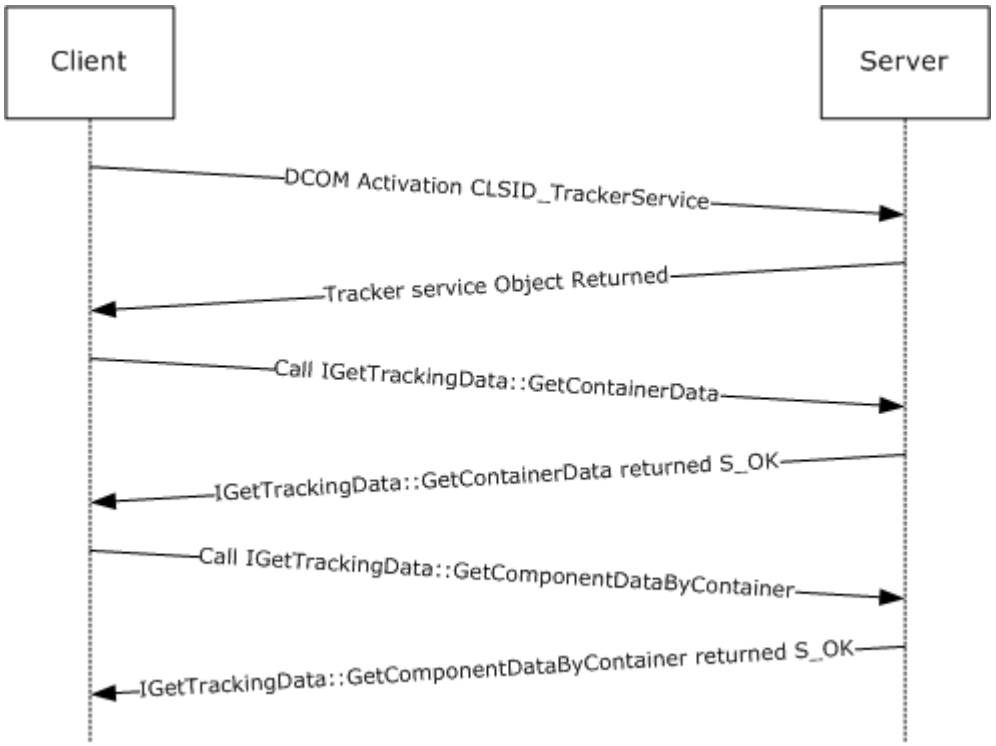


Figure 2: Polling for tracking data

This example shows a client application polling for tracking data on instance containers and the components instantiated in a particular instance container.

1. The client first performs a DCOM activation for the tracker service DCOM object on the server by using the CLSID CLSID_TrackerService.

2. The server returns an object reference to the tracker service DCOM object.

3. Using the tracker service DCOM object, the client retrieves tracking data for all instance containers on the server.

4. The server returns an array of **ContainerData (section 2.2.3)** structures in *aContainerData* and returns S_OK.

5. The client then finds the container legacy identifier (such as, 0x00000173) for an instance container of interest in one of the **ContainerData** structures and uses it to retrieve tracking data for all components instantiated in that instance container.

```
HRESULT GetComponentDataByContainer(
  [in] DWORD idContainer = 0x00000173,
  [out] DWORD* nComponents,
```

```
    [out, size_is(,*nComponents)] ComponentData** aComponentData
);
```

6. The server returns an array of **ComponentData (section 2.2.4)** structures in *aComponentData* and returns S_OK.

4.2 Receiving a Tracker Event

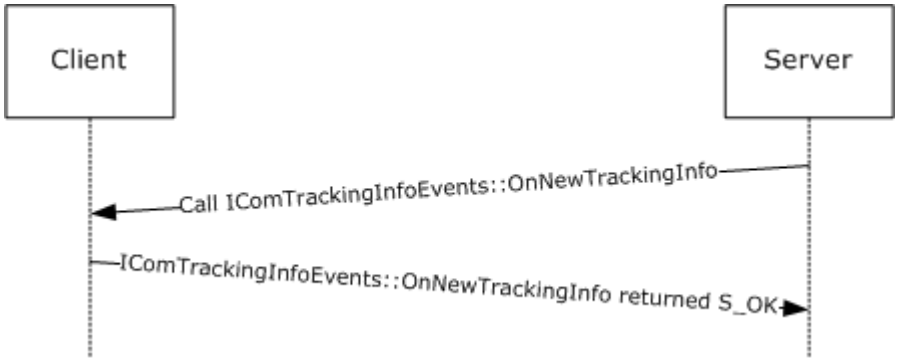


Figure 3: Receiving a tracker event

This example shows a client application that receives a tracker event. For this example, the client role acts as a DCOM server and the server role acts as a DCOM client.

This example assumes that the client has already sent its callback interface to the server.

- The server creates a TrackingInfoCollection OBJREF_CUSTOM (section 2.2.5.5) that represents the tracker event and sends it to the client.
- The client unmarshals the TrackingInfoCollection OBJREF_CUSTOM and returns S_OK.

5 Security

5.1 Security Considerations for Implementers

Implementers should review the security considerations referenced in [MS-DCOM] section 5.1 because these are also valid for the COM+ Tracker Service Protocol.

5.2 Index of Security Parameters

None.

6 Appendix A: Full IDL

For ease of implementation, the full **IDL** is provided as follows, where "ms-dcom.idl" refers to the IDL found in [MS-DCOM] Appendix A, "ms-dtyp.idl" refers to the IDL found in [MS-DTYP] Appendix A, and "ms-oaut.idl" refers to the IDL found in [MS-OAUT] Appendix A.

The syntax uses the IDL syntax extensions, as specified in [MS-RPCE] sections 2.2.4 and 3.1.1.5.1, and the Automation IDL syntax extensions specified in [MS-OAUT] section 2.2.49. For example, as noted in [MS-RPCE] section 2.2.4.9, a pointer_default declaration is not required and pointer_default(unique) is assumed.

```
import "ms-dcom.idl";
import "ms-dtyp.idl";
import "ms-oaut.idl";

typedef struct
{
  DWORD cCalls;
  DWORD cComponentInstances;
  DWORD cComponents;
  DWORD cCallsPerSecond;
} ContainerStatistics;

typedef struct
{
  DWORD dwLegacyId;
  WCHAR wszApplicationIdentifier[40];
  DWORD dwProcessId;
  ContainerStatistics statistics;
} ContainerData;

typedef struct
{
  GUID  clsid;
  DWORD cTotalReferences;
  DWORD cBoundReferences;
  DWORD cPooledInstances;
  DWORD cInstancesInCall;
  DWORD dwResponseTime;
  DWORD cCallsCompleted;
  DWORD cCallsFailed;
} ComponentData;

[
    object,
    uuid(B60040E0-BCF3-11D1-861D-0080C729264D),
    pointer_default(unique)
]
interface IGetTrackingData: IUnknown
{
    HRESULT Opnum3NotUsedOnWire();

    HRESULT GetContainerData(
        [out] DWORD* nContainers,
        [out, size_is(,(*nContainers))]
           ContainerData** aContainerData
    );
```

```
    HRESULT GetComponentDataByContainer(
        [in]  DWORD  idContainer,
        [out] DWORD* nComponents,
        [out, size_is(,*nComponents)]
           ComponentData** aComponentData
    );

    HRESULT GetComponentDataByContainerAndCLSID(
        [in] DWORD idContainer,
        [in] GUID  clsid,
        [out] ComponentData** ppComponentData
    );

    HRESULT Opnum7NotUsedOnWire();
};


[
    object,
    uuid(4E6CDCC9-FB25-4FD5-9CC5-C9F4B6559CEC),
    pointer_default(unique)
]
interface IComTrackingInfoEvents: IUnknown
{
    HRESULT OnNewTrackingInfo(
        [in] IUnknown* pToplevelCollection
    );
};


[
    object,
    uuid(23C9DD26-2355-4FE2-84DE-F779A238ADBD),
    dual
]
interface IProcessDump: IDispatch
{
    [id(1)]
    HRESULT IsSupported();

    [id(2)]
    HRESULT DumpProcess(
        [in] BSTR bstrContainerID,
        [in] BSTR bstrDirectory,
        [in] DWORD dwMaxFiles,
        [out,retval] BSTR* pbstrDumpFile
    );
};
```

7 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft Windows® 2000 operating system
- Windows® XP operating system
- Windows Server® 2003 operating system
- Windows Vista® operating system
- Windows Server® 2008 operating system
- Windows® 7 operating system
- Windows Server® 2008 R2 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 2.2.2: Windows collects method call statistics in 1-second intervals and calculates the statistics over the four most recent intervals.

<2> Section 2.2.4: On Windows, the COM+ Tracker Service Protocol server tracks out-of-process COM references (for more information, see [MSDN-COM]).

<3> Section 2.2.4: On Windows, reference-count statistics are collected for tracked conglomerations. Windows determines whether or not to track a conglomeration based on its per-conglomeration configuration, which can be modified by using the COM+ Remote Administration Protocol EventsEnabled property. For more information about this property, see the Conglomerations Table in [MS-COMA] section 3.1.1.3.6.

<4> Section 2.2.4: On Windows, reference-count statistics are collected for tracked conglomerations. Windows determines whether or not to track a conglomeration based on its per-conglomeration configuration, which can be modified by using the COM+ Remote Administration Protocol EventsEnabled property. For more information about this property, see the Conglomerations Table in [MS-COMA] section 3.1.1.3.6.

<5> Section 2.2.4: On Windows, pooling statistics are collected for tracked conglomerations. Windows determines whether or not to track a conglomeration based on its per-conglomeration configuration, which can be modified by using the COM+ Remote Administration Protocol EventsEnabled property. For more information about this property, see the Conglomerations Table in [MS-COMA] section 3.1.1.3.6.

<6> Section 2.2.4: On Windows, method call statistics are collected for tracked conglomerations. Windows determines whether or not to track a conglomeration based on its per-conglomeration configuration, which can be modified by using the COM+ Remote Administration Protocol

EventsEnabled property. For more information about this property, see the Conglomerations Table in [MS-COMA] section 3.1.1.3.6.

<7> Section 2.2.4: Windows collects method call statistics in 1-second intervals and calculates the statistics over the four most recent intervals. The average response time for a component is calculated as the slowest average response time over all component instances in the instance container.

<8> Section 2.2.4: On Windows, method call statistics are collected for tracked conglomerations. Windows determines whether or not to track a conglomeration based on its per-conglomeration configuration, which can be modified by using the COM+ Remote Administration Protocol EventsEnabled property. For more information about this property, see the Conglomerations Table in [MS-COMA] section 3.1.1.3.6.

<9> Section 2.2.4: Windows collects method call statistics in 1-second intervals and calculates the statistics over the four most recent intervals.

<10> Section 2.2.4: Windows considers a method call successful if it returns a success result, as specified in [MS-ERREF] section 2.1.

<11> Section 2.2.4: On Windows, method call statistics are collected for tracked conglomerations. Windows determines whether or not to track a conglomeration based on its per-conglomeration configuration, which can be modified by using the COM+ Remote Administration Protocol EventsEnabled property. For more information about this property, see the Conglomerations Table in [MS-COMA] section 3.1.1.3.6.

<12> Section 2.2.4: Windows collects method call statistics in 1-second intervals and calculates the statistics over the four most recent intervals.

<13> Section 2.2.4: Windows considers a method call failed if it returns a failure result, as specified in [MS-ERREF] section 2.1.

<14> Section 2.2.4: On Windows, method call statistics are collected for tracked conglomerations. Windows determines whether or not to track a conglomeration based on its per-conglomeration configuration, which can be modified by using the COM+ Remote Administration Protocol EventsEnabled property. For more information about this property, see the Conglomerations Table in [MS-COMA] section 3.1.1.3.6.

<15> Section 3: On the following Windows versions, the COMT server exposes tracker events as a COM+ Event System Protocol event class (for more information, see [MS-COMEV] section 3.1.1.1) with the EventClassID {ECABB0C3-7F19-11D2-978E-0000F8757E2A}:

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

To receive tracker events from a Windows server, a client application creates a COM+ Event System Protocol subscription object (for more information, see [MS-COMEV] section 3.1.1.2) with the following properties:

Property	Value
EventClassID	{ECABB0C3-7F19-11D2-978E-0000F8757E2A}

Property	Value
InterfaceID	{4E6CDCC9-FB25-4FD5-9CC5-C9F4B6559CEC}

The client application can set other subscription properties to values that are appropriate to the application. The client application then stores the subscription by calling the **IEventSystem::Store** method (for more information, see [MS-COMEV] section 3.1.4.1.2).

Note that the COMEV server on Windows might be unable to create a subscription if the subscriber interface is located on an object server that does not accept anonymous incoming calls. See [MS-COMEV] Appendix B Windows Behavior (section 7) for more information.

On Windows 2000, the COMT server does not support tracker events.

On the following Windows versions, the Component Services administrative tool creates such a subscription, setting the SubscriberInterface property to its implementation of IComTrackingInfoEvents by calling IEventSubscription::put_SubscriberInterface (for more information, see [MS-COMEV] section 3.1.4.4.14):

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

On Windows 2000, the Component Services administrative tool does not create a subscription to tracker events.

<16> Section 3: Windows COMT servers make calls to the IComTrackingInfoEvents interface as the machine account of the server.

<17> Section 3: On the following Windows versions, the COMT server stops sending tracker events to a client application when that application removes the subscription object it created:

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

A client application can do this by calling either the **IEventSystem::Remove** method or the **IEventSystem::RemoveS** method (for more information, see [MS-COMEV] sections 3.1.4.1.3 and 3.1.4.1.6, respectively).

On Windows 2000, the COMT server does not support tracker events.

On the following Windows versions the Component Services administrative tool removes the subscription it created by calling **IEventSystem::Remove**:

- Windows XP
- Windows Server 2003
- Windows Vista

- Windows Server 2008

On Windows 2000, the Component Services administrative tool does not create a subscription to tracker events.

<18> Section 3.1.1: On Windows, the Partition ID property is the ID property of the Partition (for more information, see [MSDN-Partitions]) for the COM+ application (for more information, see [MSDN-Applications]).

<19> Section 3.1.4.1: Opnums reserved for local use apply to Windows as follows:

opnum	Description
3	Returns E_NOTIMPL only. It is never used.
7	Returns E_NOTIMPL only. It is never used.

<20> Section 3.1.4.2.2: Windows COMT servers accept a value for the *bstrDirectory* parameter that is a path to a directory in the server's local file system (for more information, see [MSDN-FILE]) or in **Universal Naming Convention (UNC)** representing a local or remote path. In addition, Windows COMT servers require paths to have a maximum length of 260 (MAX_PATH) Unicode characters.

<21> Section 3.1.4.2.2: Windows COMT servers enable the default dump file location to be configured for a conglomeration via the COMA protocol [MS-COMA]. This location is the conglomeration's **DumpPath** property as specified in [MS-COMA] section 3.1.1.3.6.

<22> Section 3.1.4.2.2: Windows COMT servers enable the default dump file limit to be configured for a conglomeration via the COMA protocol [MS-COMA]. This location is the conglomeration's **MaxDumpCount** property as specified in 3.1.1.3.6.

<23> Section 3.1.4.2.2: Windows COMT servers do not impersonate the client when attempting to determine the number of previously written dump files or when deleting a previously written dump file. Windows COMT servers do impersonate the client when writing the dump file.

<24> Section 3.1.4.2.2: Windows COMT servers write user-mode mini-dumps (for more information, see [MSDN-MDWD]). The file name used for these mini-dumps is of the form "ConglomerationIdentifier-YYYY_MM_DD_HH_mm_ss", where ConglomerationIdentifier is the CurlyBraceGuidString representation of the conglomeration identifier of the conglomeration corresponding to the specified instance container, and the remaining fields represent the date and time when the process dump was requested.

<25> Section 3.2.5.1.1: On Windows, a process is a Windows process. The ExeName property is the executable image file name.

<26> Section 3.2.5.1.1: The IsService property is TRUE (0x00000001) or FALSE (0x000000000) to indicate whether the process is a Windows service.

<27> Section 3.2.5.1.1: On Windows, an instance container is an instance of a COM+ application (for more information, see [MSDN-Applications]). The ApplicationType property is one of the following values:

Value	Meaning
0	A COM+ library application.

Value	Meaning
1	A COM+ server application.
2	Services without components.

<28> Section 3.2.5.1.1: On Windows, the Name property is the COM+ application name (for more information, see [MSDN-Applications]).

<29> Section 3.2.5.1.1: Windows collects method call statistics in 1-second intervals and calculates the statistics over the four most recent intervals. The average response time for a component is calculated as the slowest average response time over all component instances in the instance container.

<30> Section 3.2.5.1.1: On Windows, the Name property is the ProgId of the component (for more information, see [MSDN-COM]).

8 Change Tracking

This section identifies changes that were made to the [MS-COMT] protocol document between the May 2011 and June 2011 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1.2 References	Added explanatory statement regarding the removal of the publishing year from Microsoft Open Specification document references.	N	Content updated.

9 Index

T

V