TM

OPC Security Custom Interface

Version 1.0

October 17, 2000

Specification Type	Industry Standard Specification		
Title:	**OPC Security Custom Interface**	Date:	October 17, 2000
Version:	1.0	Soft	MS-Word
		Source:	Opc Security - Custom - R100.doc
Author:	OPC Foundation	Status:	

Synopsis:

This specification is the specification of the interface for developers of OPC clients and servers.. The specification is a result of an analysis and design process to develop a standard interface to facilitate the development of servers and clients by multiple vendors that shall inter-operate seamlessly together.

Trademarks:

Required Runtime Environment:

An OPC Server running on Windows 9X with DCOM, Windows CE 2.11, Windows NT 4.0 SP5 or later

Table of Contents

1. Introduction

1.1 Background

OLE for Process Control has defined interfaces for Data Access Servers, Event Servers, and Historical Data Access Servers. These servers provide information that is valuable to the enterprise and if improperly updated, could have significant consequences to plant processes. Therefore, there is a need to control client access to these servers in order to protect this sensitive information and to guard against unauthorized modification of process parameters.

1.2 Key Requirements

Security must be provided in a standard manner, consistent among implementations of OPC Servers by various vendors, to permit the implementation of portable client applications.

Security must be well integrated with Windows NT and be as transparent as possible to the client application. Ideally, security should "just be there" with no special actions by the client application required in order for security to be enforced.

1.3 Purpose

The purpose of this document is to specify how OPC Servers should implement security using operating system facilities. In addition, usage guidelines are provided for the OPC Client implementation to interact with a security aware OPC Server.

1.4 Relationship to Other OPC Specifications

This specification is analogous to the OPC Common specification in that it applies to all of the defined OPC Servers.

1.5 Scope of This Specification

1.5.1 Security Aspects

This specification focuses on client identification, that is the exchanging of trusted credentials to be used for access authorization decisions by the OPC Server. It does not address which objects are to be secured, but leaves this matter to the OPC Server implementers.

1.5.2 Levels of Security

This specification permits multiple levels of security to maintain coexistence with legacy OPC applications and to provide enhanced security capabilities.

1.5.3 Operating Systems

This specification applies to the following computer platforms:

Windows NT 4.0, Service Pack 5 and later, including Windows 2000 – Full client/server functions are supported.

Windows 95/98 – Full client/server functions are supported, with restrictions based on the limited security model.

Windows CE 2.11 and later - Full client/server functions are supported, with restrictions based on the limited security model. Third party support is required for DCOM transport.

Other Platforms – This specification may apply to other platforms (such as UNIX) based on the availability of DCOM implementations on such platforms.

1.6 Prerequisites

Readers are assumed to be familiar with the applicable OPC Specifications; for example, OPC Common, OPC Data Access, OPC Alarms and Events, and OPC Historical Data Access.

These following document titles and others can be found at the following Web address: http://www.opcfoundation.org/specs.asp:

OPC Data Access Custom Interface 2.0

OPC Data Access Automation Interface 2.0

OPC Alarms and Events 1.0

OPC Common Definitions and Interfaces 1.0

Readers should be familiar with DCOM and with Windows 2000 security features and security administration.

Information regarding Distributed COM and various links to related sites, white papers, specs, etc, can be found at the following Web address: http://www.microsoft.com/com/tech/DCOM.asp

Specifications on DCOM/COM and release notes on COM+ can be found in the MSDN Online Library at the following Web address: http://www.microsoft.com/com/resources/specs.asp

A white paper on third party DCOM implementations for Windows CE can be found at the following Web address: http://www.microsoft.com/ industry/man/whitepapers/whitepapers.asp

1.7 Deliverables

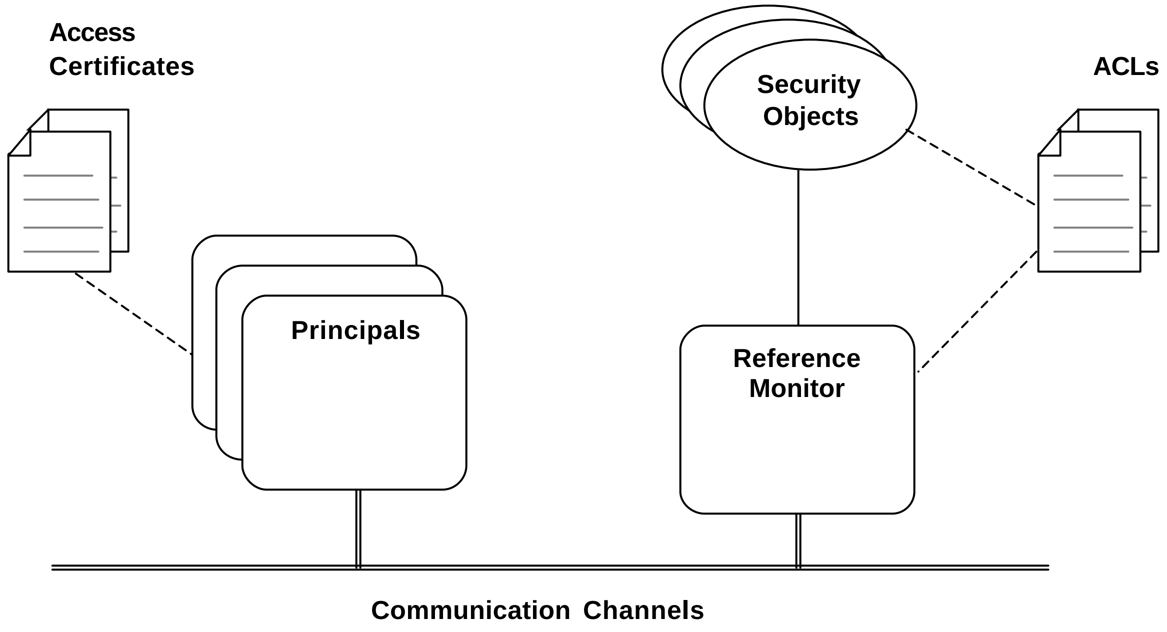
This document covers the analysis and design for a COM compliant custom interface. A separate document describes a related OLE Automation interface.

2. Fundamental Concepts

2.1 Security Reference Model

To understand this specification, it is helpful to first examine a security reference model. This model is intended to serve as a conceptual framework for the remainder of the specification. It is consistent with the Windows NT security model.

The figure below shows the elements of the security model.



The key elements of the security reference model are as follows:

Principal – An active entity which has a need to access one or more security objects. Some examples of Principals are human users, computer systems, etc. In Windows NT, NT processes represent all Principals, which can be either interactive or service.

Security Object – Any entity to which access is to be controlled. Some examples of security objects are files, directories, registry keys, key systems, etc.

Reference Monitor – An active entity which makes access authorization decisions for a set of security objects. A well-implemented security system ensures that all access requests for a security object are routed through its associated reference monitor, that is to say, no access to a security object can be granted except by the approval of its reference monitor. An example of a reference monitor is the NT reference monitor for NTFS files and directories.

Access Control List (ACL) – A structure associated with every security object which specifies those Principals and groups of Principals allowed to access the secure object along with the types of access each is allowed. Examples of ACLs are the NT Access Control Lists that are associated with files, directories, DCOM servers, registry keys, etc.

Channel – A communication path between any two active entities within the system. A channel may be entirely contained within a single system, for example inter-process communications between two processes co-located on the same system. Alternatively, a channel may span computer systems using a communications network. A channel that is contained within a single computer system is typically considered to be secure. A channel that spans computer systems is typically considered to be insecure, unless specific actions have been taken to secure it. A channel may be secured to one of three levels:

Authenticated – A recipient can be confident of the identity of a message sender.

Verifiable – A recipient can be confident that the message has not been modified in transit.

Private – A sender can be confident that the message will be read only by the intended recipients.

Access Certificate – A trusted credential associated with a Principal, which is used by a Reference Monitor as a basis for an access authorization decision. An example of an access certificate is the NT Access Token associated with every NT process.

2.1.1 Authorization vs. Authentication

Authorization is the process of granting a Principal access to a security object. The authorization decision is made by the Reference Monitor, based on a comparison of the Principal's access certificate to the security object's access control list.

Authentication is a property of the underlying communications channel, and refers to the ability of the Reference Monitor to be confident of the validity of the identity and credentials of a Principal requesting access to a security object.

2.2 Application of the Model in This Specification

2.2.1 Principals

The Principals covered in this specification are computer processes running on one of the client or server operating systems specified in Section 1.4.2. Programs or applications are not Principals in their own right, but are assigned the access rights of the processes in which they are running. In NT, all processes run under user accounts, which specify their access rights.

2.2.2 Access Certificates

In this specification, the Access Certificate is either:

1. The NT Access Token, which should be used wherever possible

or

2. A private credential specified by the OPC Server.

See section 2.4 *Private Credential vs. NT Access Token* for further explanation.

2.2.3 Security Objects

This specification deals only with OPC Server applications and Client applications (in the case of call-back interfaces), and any vendor specific security objects which are implemented by the OPC Server. Examples of such vendor specific security objects are server interfaces and methods, public groups, and individual or sets of data items. However, such security objects are vendor specific. Although the determination of which objects to secure is up to the OPC Server vendor, the manner in which they are secured must conform to the specification.

2.2.4 Access Control Lists

In this specification, Access Control Lists are the NT Access Control Lists, with the possible exception of OPC Security using private credentials discussed later.

2.2.5 Reference Monitor

This specification deals with the following reference monitors:

?? The NT Reference Monitor, which controls access to DCOM servers and client applications.

?? OPC Servers that make access authorization decisions for vendor specific objects. They will make their authorization decisions based on the NT Access Token or on a vendor specific private credentials. The NT Access Token is the preferred approach.

2.2.6 Channels

This specification deals only with Channels supported by DCOM. The channel security levels are those supported by DCOM, that is to say *connect*, *call*, *packet*, *packet integrity*, and *packet privacy*.

2.3 Levels of Security

An OPC Server may implement one of three levels of security:

Disabled Security – No security is enforced. Launch and Access permissions to the OPC Server are given to everyone, and Access permissions for clients are set for everyone. The OPC Server does not control access to any vendor specific security objects.

DCOM Security – Only NT *DCOM security* is enforced. Launch and Access permissions to the OPC Server are limited to selected clients, as are the Access permissions for client applications. However, the OPC Server does not control access to any vendor specific security objects. This is the default security level provided by DCOM.

OPC Security – The OPC Server serves as a reference monitor to control access to vendor specific security objects that are exposed by the OPC Server. An OPC Server may implement *OPC Security* in addition to *DCOM Security*, or implement *OPC Security* alone.

2.4 Private Credential vs. NT Access Token

Whenever possible an OPC Server which implements OPC Security should base its access authorization decisions upon the NT Access Token associated with the client application. This approach allows security to be transparent to client applications because there is no need to take explicit action to establish an additional access certificate. Also, since the NT Access Token is independent of the OPC Server, this approach makes it easier to write portable client applications.

However there are circumstances which preclude the use of the NT Access Token. Some examples of such situations include:

?? The OPC Server is running on a non-NT operating system that supports DCOM such as Windows CE or UNIX.

?? Client applications may be running on an operating system that does not support the creation of an NT Access Token.

?? The OPC Server and client applications are distributed on multiple devices outside the context of an NT Domain.

To address such situations, this specification defines the ability for the OPC Server to implement a private credential (access certificate) which it verifies. An example might be an OPC Server specific UserID and password combination. It is up to the OPC Server to use mechanisms that prevent the compromise of these credentials. For example, passwords should not be transmitted in clear text, nor should they be stored on disk in clear text.

OPC Servers that need to support client applications which have NT Access Tokens, as well as clients which do not have NT Access Tokens, may implement support for both types of access certificates, on a client-by-client basis. This approach provides a transparent security implementation for those clients with NT Access Tokens, while providing security for those clients without NT Access Tokens.

2.5 Optional IOPCSecurityNT and IOPCSecurityPrivate Interfaces

OPC Servers that implement *OPC Security* must implement one or both of the IOPCSecurityNT or IOPCSecurityPrivate interfaces. Existence of these interfaces allows client applications to determine if *OPC Security* is implemented, and if so, which types of access certificates are supported.

2.6 Backward Compatibility

OPC Servers that implement *Disabled Security* or *DCOM Security* are fully compatible with clients which are not security aware.

OPC Servers that implement *OPC Security* based on the NT Access Token are also fully compatible with clients that are not security aware. Certain authentication and impersonation levels are required, see section 6.3.1.1 *Recommended Security Setup*.

OPC Servers that implement *OPC Security* based on a private credential are compatible with clients which are not security aware only in a degenerate sense, in that such clients will always be denied access when attempting to access security objects.

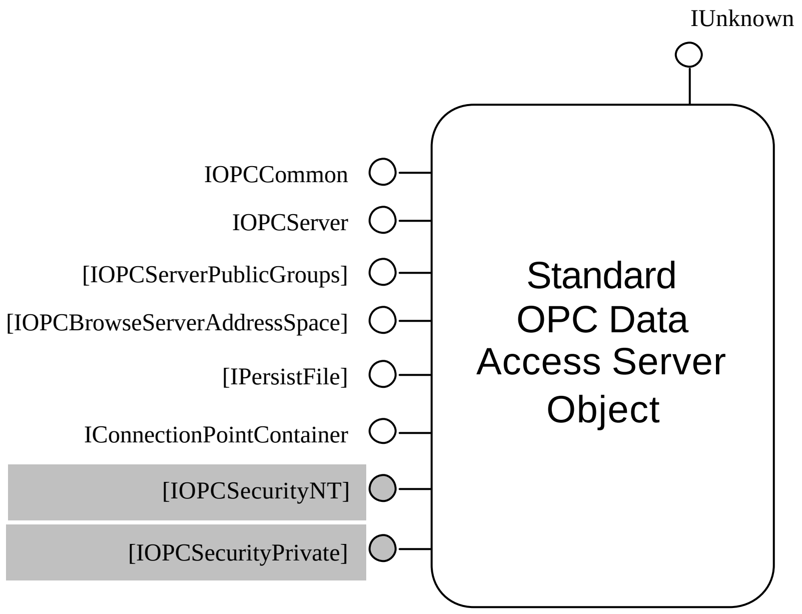
3. OPC Security Quick Reference

This section includes a quick reference for both Custom and Automation Interface methods. These interfaces, their parameters, and behavior are defined in more detail later in the reference sections.

3.1 *"OPCServer" Object*

"OPCServer" may be one of the following: OPC (Data Access) Server, OPC Event Server, or OPC HDA Server. The intent is that these interfaces be added to the above server objects, similar to the implementation of *IOPCCommon*.

This section does not show additional standard COM Interfaces such as *IUnknown*, *IEnumString* and *IEnumUnknown* used by other OPC Specifications.



Note: As an example, this OPC Data Access 2.0 server shows the relationship of the IOPCSecurity interface to the OPC server object

3.1.1 IOPCSecurityNT

HRESULT	IsAvailableNT([out] BOOL *pbAvailable);;
HRESULT	QueryMinImpersonationLevel([out] DWORD *pdwMinImpLevel);
HRESULT	ChangeUser(void);

3.1.2 IOPCSecurityPrivate

HRESULT	IsAvailablePriv([out] BOOL *pbAvailable);;
HRESULT	Logon([in, string] LPCWSTR szUserID, [in, string] LPCWSTR szPassword);
HRESULT	Logoff(void);

4. Custom Reference

4.1 Interface Issues

Please refer to the OPC Common Interface Issues section in the OPC Common specification for a description of issues which are common to all interfaces, and for some background information about how the designers of OPC expected these interfaces to be implemented and used.

4.2 Overview

Every OPC Server (DA, A&E, HDA, etc.) implementing OPC Security must implement either one or both interfaces completely.

IOPCSecurityNT	Control access using NT credentials
IOPCSecurityPrivate	Control access using private credentials

A client may not assume the presence of this interface on a server and is expected to handle the lack of either one or both gracefully. A client can call `QueryInterface()` for both security interfaces. The client knows that the OPC server is implementing *OPC Security* if at least one of the calls returns a valid interface pointer.

As already depicted in the OPC Server object (see figure above in section 3.1 *"OPC Server" Object*), both the NT and the private credential interfaces reside in the OPC server's root COM object. This root object defines a COM session; all dynamic security settings and changes affect all interfaces and child objects obtained from this root object.

A typical scenario is described below, using an OPC Data Access Server:

1. After the instantiation of the OPC server, a client commonly obtains a pointer to the *IOPCServer* interface. This server interface defines the session.
2. A pointer to one of the OPC Security interfaces can be returned to a call of `QueryInterface()`. This interface allows control of the dynamic security setting for the session.
3. Usually a client adds OPC groups by calls to the `IOPCServer::AddGroup()` method. Items can be added to and managed by the OPC groups. These child objects and interfaces are also part of the same session. A change in credentials not only affects the interfaces of the server object but of all OPC group objects also.

4.2.1 Behavior of Servers Implementing Both OPC Security Interfaces

If both interfaces are currently enabled, by default the OPC Server will assume that the client is operating using NT credentials, and will continue to do so until the client explicitly invokes the `IOPCSecurityPrivate::Logon` method. Once this has occurred, the OPC Server will make all access authorization decisions based on the private credential until the client invokes `IOPCSecurityPrivate::Logoff`. During the time between the `Logon()`and `Logoff()`, a call to `IOPCSecurityNT::ChangeUser` will be invalid and will result in an error return.

4.3 IOPCSecurityNT

This is an optional interface and will not be supported by servers not implementing *OPC Security*. A client may not assume the presence of this interface on a server.

By the presence of this interface, the server promises to provide *OPC Security* through NT credentials. This interface must be implemented completely. If authorization using NT credentials is disabled by server configuration, `QueryInterface()` may still return a valid `IOPCSecurityNT` pointer. All calls but `IsAvailableNT()` will return error codes, `IsAvailableNT()` will set the bAvailable to FALSE to signal that NT credentials are currently disabled.

Certain servers will need impersonation or even delegation for proper access to their data sources. Please see comments in the description of `IOPCSecurityNT::QueryMinImpersonationLevel()` below.

The client must not change user credentials without also calling `ChangeUser()`. The expected OPC Server behavior is to continue to base access decisions on the user credentials in effect at the last invocation of `ChangeUser()`, until the client makes a new call to that method.

4.3.1 IOPCSecurityNT::IsAvailableNT

```
HRESULT IsAvailableNT(
   [out] BOOL *pbAvailable
   );
```

Description

Query the current security configuration of the OPC server to determine if the current server configuration provides *OPC Security* by NT credentials.

Parameters	**Description**
pbAvailable	?? TRUE: current configuration allows authorization using NT credentials ?? FALSE: current configuration has authorization using NT credentials disabled

Return Codes

Return Code	**Description**
S_OK	The operation succeeded.
E_FAIL	The operation failed.
E_INVALIDARG	An argument to the function was invalid

Comments

All servers implementing *OPC Security* are expected to fully implement this method; that is to say, calls will not return E_NOTIMPL.

It is expected that a security aware client will call this method to query the current security settings a server has to offer.

This method pertains to server wide configuration settings by a system administrator, and does not reflect current state of the client session. Specifically, in the case where a server supports both *IOPCSecurityNT* and *IOPCSecurityPrivate* interfaces, the result of this method will not reflect any state change depending on which interface is currently being used.

4.3.2 IOPCSecurityNT::QueryMinImpersonationLevel

```
HRESULT QueryMinImpersonationLevel(
   [out] DWORD *pdwMinImpLevel
   );
```

Description

Information method to help a client to determine the minimal impersonation level the server requires to gain proper access to secured data sources.

Parameters	**Description**
pdwMinImpLevel	Least required impersonation level for proper access. ?? RPC_C_IMP_LEVEL_ANONYMOUS ?? RPC_C_IMP_LEVEL_IDENTIFY ?? RPC_C_IMP_LEVEL_IMPERSONATE ?? RPC_C_IMP_LEVEL_DELEGATE See also COM documentation for details.

Return Codes

Return Code	**Description**
S_OK	The operation succeeded.
E_FAIL	The operation failed.
E_INVALIDARG	An argument to the function was invalid

Comments

When using NT credentials, a server can control and enforce access to secured objects in different ways. Some servers might even need to retrieve data by another DCOM call to a remote COM server on a third machine. This scenario could require Delegation to gain access to the remote data source.

It is expected that clients will match the impersonation level of their proxy blanket with the requirements of the OPC server.

Servers should not change the minimal impersonation level dynamically while one session is still active.

If the client has not set the impersonation level appropriately, the server will behave as follows:

?? Return OPC_E_LOW_IMPERS_LEVEL for calls to `ChangeUser()`.

?? Return E_ACCESSDENIED to any calls which access security objects.

See section 6.3.4.5 *Impersonation Levels* for additional information.

4.3.3 IOPCSecurityNT::ChangeUser

```
HRESULT ChangeUser();
```

Description

Signal the server that the client has changed the user credentials of its proxy blanket.

Parameters	**Description**
Void	

Return Codes

Return Code	**Description**
S_OK	The operation succeeded.
E_FAIL	The operation failed.
OPC_E_PRIVATE_ACTIVE	An `IOPCSecurityPrivate::Logon` is currently active.
OPC_E_LOW_IMPERS_LEVEL	The client has failed to set the impersonation level as required by the server. See `QueryMinImpersonationLevel`.

Comments

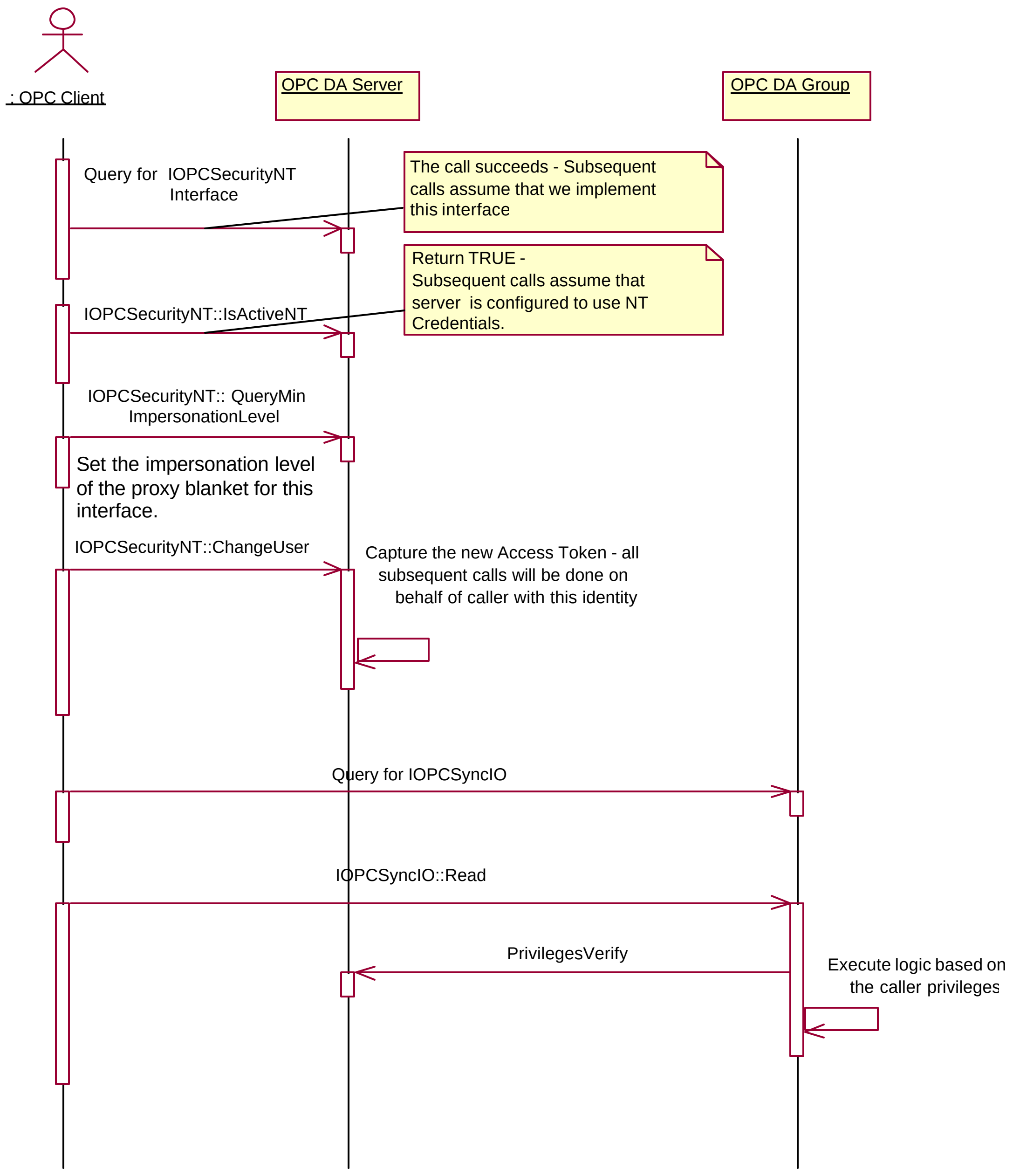
A call to this method delivers an “event” to the server to reassess the (potentially cached) user credentials at the server side. The client must set the user credentials and impersonation level of this interface’s proxy blanket before calling this method. A server will determine the new user credentials by querying its client blanket.

If a server needs a correct Access Token to access its data sources (e.g. a file or database with an Access Control List), the server may copy and cache the calling thread’s Access Token. This cached Access Token can be reapplied later when the client issues an OPC call requiring access to the secured object (see Win32 API for `DuplicateToken`, `SetThreadToken`).

If a client has not called this method prior to accessing a secured object, the expected behavior of the OPC Server is that it will base its access authorization decisions on the client credentials in effect at the time of connection to the server.

4.3.4 Use Scenario

The following interaction diagram shows a scenario in which a client accesses an OPC Data Access server to perform a synchronous read operation. The server implements *OPC Security* using NT credentials.


: OPC Client
OPC DA Server
OPC DA Group
Query for IOPCSecurityNT Interface
The call succeeds - Subsequent calls assume that we implement this interface
Return TRUE - Subsequent calls assume that server is configured to use NT Credentials.
IOPCSecurityNT::IsActiveNT
IOPCSecurityNT:: QueryMin ImpersonationLevel
Set the impersonation level of the proxy blanket for this interface.
IOPCSecurityNT::ChangeUser
Capture the new Access Token - all subsequent calls will be done on behalf of caller with this identity
Query for IOPCSyncIO
IOPCSyncIO::Read
PrivilegesVerify
Execute logic based on the caller privileges

4.4 IOPCSecurityPrivate

This is an optional interface and will not be supported by servers not implementing *OPC Security* using Private Credentials. A client may not assume the presence of this interface on a server.

By the presence of this interface, the server promises to provide *OPC Security* through private credentials. This interface must be implemented completely. If authentication by private credentials is disabled by server configuration, `QueryInterface()` may still return a valid IOPCSecurityPrivate pointer. All calls but `IsAvailablePriv()` will return error codes, `IsAvailablePriv()` will set the bAvailable to FALSE to signal that private credentials are currently disabled.

4.4.1 IOPCSecurityPrivate::IsAvailablePriv

```
HRESULT IsAvailablePriv(
   [out] BOOL *pbAvailable
   );
```

Description

Query the current security configuration of the OPC server to determine if the current server configuration provides *OPC Security* by private credentials.

Parameters	**Description**
pbAvailable	?? TRUE: current configuration allows authentication by private credentials ?? FALSE: current configuration has authentication by private credentials

Return Codes

Return Code	**Description**
S_OK	The operation succeeded.
E_FAIL	The operation failed.
E_INVALIDARG	An argument to the function was invalid

Comments

All servers implementing *OPC Security* using Private Credentials are expected to fully implement this method; that is to say, calls will not return E_NOTIMPL.

It is expected that a security aware client will call this method to query the current security settings a server has to offer.

This method pertains to server wide configuration settings by a system administrator, and does not reflect current state of the client session. Specifically, in the case where a server supports both *IOPCSecurityNT* and *IOPCSecurityPrivate* interfaces, the result of this method will not reflect any state change depending on which interface is currently being used.

4.4.2 IOPCSecurityPrivate::Logon

```
HRESULT Logon(
   [in, string] LPCWSTR szUserID,
   [in, string] LPCWSTR szPassword
   );
```

Description

Changes the identity of the client application's user. Future access to any security objects will be authorized with the new user's credentials until a subsequent call to `Logoff()` or `Logon()`. For OPC Servers which also implement IOPCSecurityNT, access checking with NT credentials will be disabled until `Logoff()` is called.

Parameters	**Description**
szUserID	The user's logon name; for logons in NT domains this string contains also the domain name, e.g., "OPC\Ben"
szPassword	The user's password.

Return Codes

Return Code	**Description**
S_OK	The operation succeeded.
E_FAIL	The operation failed.
E_ACCESSDENIED	The credentials passed in could not be authenticated.
OPC_S_LOW_AUTHN_LEVEL	Server expected higher level of packet privacy

Comments

To prevent clear text passwords from being transmitted across the wire, the client application is expected to call `CoSetProxyBlanket()` to raise the level of authentication on this interface to be RPC_C_AUTHN_LEVEL_PKT_PRIVACY before calling this function. The server will check the authentication level on entry to this method and will return a status code signaling low level of privacy. However, servers will NOT enforce this level of privacy. Instead, they will return a status code as an indication to the client. If the proper authentication level has not been set, the exposure of the password would already be done when the server receives the method call.

The server may use a vendor specific database to lookup the user ID and password. Initial registration of the user ID and password are also vendor specific.

Any further call of `Logon()` without any prior call to `Logoff()` will change to the newly supplied credentials immediately.

4.4.3 IOPCSecurityPrivate::Logoff

```
HRESULT Logoff();
```

Description

Remove the private credential established by the previous call to `Logon()`. OPC Security reverts to the state before the first call of `Logon()`, there are no private credentials active for the client.

Parameters	Description
Void	

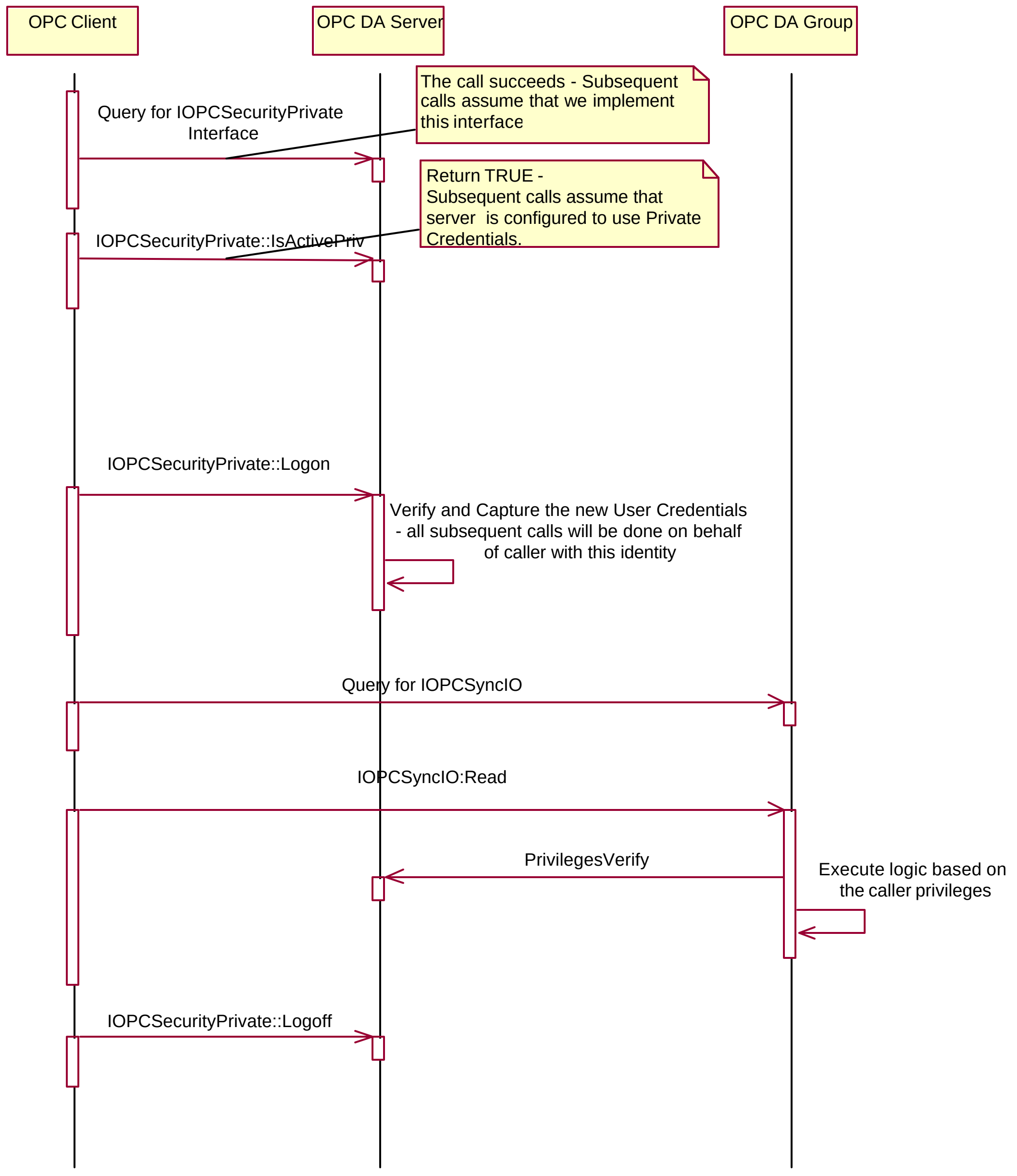
Return Codes

Return Code	Description
S_OK	The operation succeeded.
E_FAIL	The operation failed.

Comments

4.4.4 Use Scenario

The following interaction diagram shows a scenario in which a client accesses an OPC Data Access server to perform a synchronous read operation. The server implements *OPC Security* using private credentials.


OPC Client
OPC DA Server
OPC DA Group
The call succeeds - Subsequent calls assume that we implement this interface
Query for IOPCSecurityPrivate Interface
Return TRUE - Subsequent calls assume that server is configured to use Private Credentials.
IOPCSecurityPrivate::IsActivePriv
IOPCSecurityPrivate::Logon
Verify and Capture the new User Credentials - all subsequent calls will be done on behalf of caller with this identity
Query for IOPCSyncIO
IOPCSyncIO:Read
PrivilegesVerify
Execute logic based on the caller privileges
IOPCSecurityPrivate::Logoff

4.5 NT Credential Approach

The concepts and code snippets are part of the source code of the OPC security sample server as offered by the OPC foundation (see http://www.opcfoundation.org).

4.5.1 Server Security

The mechanism of using NT credentials relies on the user credentials inherently sent along with COM calls. This implies minimal settings of DCOM/RPC authentication to allow proper identification of the calling user.

A security aware OPC server must have a chance to identify and authenticate the calling user. The server requires at least an authentication level of RPC_C_AUTH_LEVEL_CONNECT, which allows the server to properly identify a client. For the server to be able to act on behalf of the client, it is recommended that the server requires an impersonation level of RPC_C_IMP_LEVEL_IMPERSONATE. The server has to enforce the correct minimum authentication level by calling `CoQueryClientBlanket()` on entry of the secured method.

4.5.1.1 Using a Private ACL to Control Access

A typical OPC server could use the following Win32 calls to determine the credentials of the caller , using private ACLs to grant and/or deny access to the OPC Server's secured objects (any error handling is omitted for clarity):

```
  //impersonate the calling client
::CoImpersonateClient();

::CoQueryClientBlanket(NULL, NULL, NULL, NULL, &dwAuthLevel, NULL, NULL);
if (dwAuthLevel < RPC_C_AUTHN_LEVEL_CONNECT) {
    return E_ACCESSDENIED;
}

HANDLE  hToken;
//retrieve the access token handle for this thread
::OpenThreadToken(::GetCurrentThread(), TOKEN_QUERY, FALSE, &hToken));
DWORD                   dwMappedAccess;
GENERIC_MAPPING         mapping;
//dwReqAccess is the required access to check for, and is declared and set
//externally to this code snippet.
dwMappedAccess = dwReqAccess;
memset(&mapping, 0xff, sizeof(GENERIC_MAPPING));
mapping.GenericRead = SEC_ACCESS_READ;
mapping.GenericWrite = SEC_ACCESS_WRITE;
mapping.GenericExecute = 0;
mapping.GenericAll = SEC_ACCESS_READ | SEC_ACCESS_WRITE;
::MapGenericMask(&dwMappedAccess, &mapping);

PRIVILEGE_SET           PrivilegeSet;
DWORD                   lenPrivSet = sizeof(PRIVILEGE_SET);
DWORD                   dwAccessGranted;

// check thread access token against ACL in SD
::AccessCheck(psdAccess, hToken, dwMappedAccess, &mapping,
              &PrivilegeSet, &lenPrivSet, &dwAccessGranted, &bAccessOK));
```

This code first impersonates the calling user for the current server thread, and opens a thread access token to represent the calling user's security context. It also checks if the least required authentication level was set by the client prior to the call.

psdAccess in the above code is a security descriptor which holds a list of accepted and denied users and or groups (also called an ACL). This security descriptor is typically created from access information read from the server's configuration when the server starts up. There may be even many security descriptors to represent different access controls for the secured OPC objects.

The thread access token and the security descriptor are then handed to the `AccessCheck()` call of NT. The `AccessCheck()` function determines whether a security descriptor grants a specified set of access rights to the client identified by an access token.

4.5.1.2 Using a System ACL to Control Access

The typical use to employ system ACLs could be a server accessing system objects, such as files, etc. These objects have system ACLs assigned and controlled by the operating system. These system ACLs can commonly be controlled through standard system means, e.g. for NTFS files use the *Properties* dialog within the File Explorer, selecting the tab *Security*. A typical code snippet for `ChangeUser()` could look as below:

```
  //impersonate the calling client
::CoImpersonateClient();

::CoQueryClientBlanket(NULL, NULL, NULL, NULL, &dwAuthLevel, NULL, NULL);
if (dwAuthLevel < RPC_C_AUTHN_LEVEL_IMPERSONATE) {
    return E_ACCESSDENIED;
}

HANDLE  hToken;
//retrieve the access token handle for this thread
::OpenThreadToken(::GetCurrentThread(),
    TOKEN_QUERY | TOKEN_DUPLICATE | TOKEN_IMPERSONATE, FALSE, &hToken));

::DuplicateToken(hToken, SecurityImpersonation, &m_hCachedToken));
```

After impersonating the calling client, this snippet first checks for the correct authentication level of the calling proxy. In this case it would at least have to be *Impersonation* level. It then opens the thread token and copies the calling thread token into a cache. This cached handle could then be used in future calls to impersonate the same identity again while trying to access system objects:

```
// reassume the cached identity:
::SetThreadToken(NULL, m_hCachedToken);
FILE *hFile = fopen("someFileWithValueForOPC_Item.txt", "r");
if (hFile == NULL) {
    // failed to open, could be "access denied" or simply "file not found"
    return HRESULT_FROM_WIN32(GetLastError());
}
// do something to retrieve new value
....................
fclose(hFile);
// revert to previous caller context:
::SetThreadToken(NULL, NULL);
```

4.5.2 Client Security

4.5.2.1 General Approach

When the client calls a server method, the COM layer keeps track of the user's credentials executing the call if a minimal authentication level is set. To change the user context to a different user, the client program must change the client proxy blanket before the next call. This change in context affects for all methods in one COM interface and remains in effect until the proxy blanket is changed again.

The client could implement the following code snippet (again error handling is skipped for brevity):

```
DWORD nAuthnLevel , nAuthnSvc, nAuthzSvc, nImpLevel, grfCap;
RPC_AUTH_IDENTITY_HANDLE    hAuth = NULL;
OLECHAR* pszServerPrincipal;
CoQueryProxyBlanket(pUnk, &nAuthnSvc, &nAuthzSvc, &pszServerPrincipal,
                                  &nAuthnLevel, &nImpLevel, &hAuth,
                                       &grfCap);

// using NTLMSSP, e.g. WinNT authorization
nAuthnSvc = RPC_C_AUTHN_WINNT;
g_pauthIdentity = new SEC_WINNT_AUTH_IDENTITY_W;
// pauthIdentity must remain allocated until we set a different (or NULL)
identity
// to proxy (see also documentation to CoSetProxyBlanket() )
g_pauthIdentity->Domain = pszDomain;
g_pauthIdentity->User = pszUser;
g_pauthIdentity->Password = pszPass;// Clear text password
g_pauthIdentity->DomainLength = wcslen(pszDomain);
g_pauthIdentity->UserLength = wcslen(pszUser);
g_pauthIdentity->PasswordLength = wcslen(pszPass);
g_pauthIdentity->Flags = SEC_WINNT_AUTH_IDENTITY_UNICODE;

CoSetProxyBlanket(pUnk, nAuthnSvc, nAuthzSvc, pszServerPrincipal,
            nAuthnLevel, nImpLevel, g_pauthIdentity, grfCap);
```

Note that the memory for the SEC_WINNT_AUTH_IDENTITY structure must remain allocated until the proxy blanket is set to another user or reset to process identity.

4.5.2.2 Client and Server on the Same Machine

The above approach based on `CoSetProxyBlanket()` is not applicable when both the client and server are on the same machine, because calls are made using LRPC. In this case, the implicit creation of an NT Access Token based on a user account specification via `CoSetProxyBlanket()` will not work. Instead, an approach like the following must be used:

1. An NT Access Token must be explicitly generated, for example by using `LogonUser()`. Note that the caller of `LogonUser()` must have the SE_TCB_NAME privilege, which may be problematic in many applications.

2. The NT Access Token must then be assigned to the calling thread, e.g. by using `SetThreadToken()` or `ImpersonateLoggedOnUser()`.

3. The client proxy blanket can then be set, e.g. by using `IClientSecurity::SetBlanket()`.

Since the case of a client needing to change user credential running on the same machine as the server is expected to be extremely rare, no sample code has been implemented.

5. Installation

5.1 Server Installation Issues

OPC Security does not need additional registry entries. However, the security proxy DLL *opcSec_PS.dll* must be installed and registered for the server.

The installation should be done as specified in section, 6.5, "Installing OPC Binaries" in the *OPC Common Definitions and Interfaces* document.

5.2 Client Installation

The security proxy DLL *opcSec_PS.dll* must be installed and registered for the client.

The installation should be done as specified in section, 6.5, "Installing OPC Binaries" in the *OPC Common Definitions and Interfaces* document.

6. Appendix

6.1 IDL File

6.1.1 OPC Security Custom Interface IDL Specification

The current files require MIDL compiler 3.00.15 or later and the WIN NT 4.0 release SDK.

Use the command line: `MIDL /ms_ext /c_ext /app_config opSec.idl`.

The resulting OPCSEC.H file should be included in all clients and servers.

The resulting OPCSEC_I.C file defines the interface IDs and should be linked into all clients and servers.

NOTE: This IDL file and the Proxy/Stub generated from it should NEVER be modified in any way. If you add vendor specific interfaces to your server (which is allowed) you must generate a SEPARATE vendor specific IDL file to describe only those interfaces and a separate vendor specific ProxyStub DLL to marshall only those interfaces.

```
// opcSec.IDL
// REVISION:        16.09.99 13:41 (PST)
// VERSIONINFO     0.91
//

import "oaidl.idl" ;


//****************************************************
//Interface Definition
//****************************************************
[
    object,
    uuid(7AA83A01-6C77-11d3-84F9-00008630A38B),
    helpstring("Optional OPC server interface to access OPC extended
    security with NT credentials"),
    pointer_default(unique)
]
interface IOPCSecurityNT : IUnknown
{
    [
        helpstring("Query if current server configuration makes NT
        credentials available")
    ]
    HRESULT IsAvailableNT([out] BOOL *pbAvailable);

    [
        helpstring("Query minimal server requirements for DCOM
        impersonation level")
    ]
    HRESULT QueryMinImpersonationLevel([out] DWORD *pdwMinImpLevel);

    [
        helpstring("Notify current OPC session of change in user
        credentials in proxy blanket")
    ]
```

```
        HRESULT ChangeUser(void);

    };

    [
        object,
        uuid(7AA83A02-6C77-11d3-84F9-00008630A38B),
        helpstring("Optional OPC server interface to access OPC extended
        security with private credentials"),
        pointer_default(unique)
    ]
    interface IOPCSecurityPrivate : IUnknown
    {
        [
            helpstring("Query if current server configuration makes private
            credentials available")
        ]
        HRESULT IsAvailablePriv([out] BOOL *pbAvailable);

        [
            helpstring("Logon current OPC session using these private
            credentials; this commonly disables server's use of NT
            credentials")
        ]
        HRESULT Logon([in, string] LPCWSTR szUserID, [in, string] LPCWSTR
        szPassword);

        [
            helpstring("Clear previous credential and revert to default
            (initial state before first logon call)")
        ]
        HRESULT Logoff(void);
    };

    //****************************************************
    // This TYPELIB is generated as a convenience to users of high level tools
    // which are capable of using or browsing TYPELIBs.
    // 'Smart Pointers' in VC6 is one example.
    //****************************************************
    [
        uuid(7AA83AFF-6C77-11d3-84F9-00008630A38B),
        version(1.0),
        helpstring("OPC Security 1.0 Custom Interface Type Library")
    ]
    library opcSec
    {
        importlib("stdole32.tlb");
        importlib("stdole2.tlb");

        interface IOPCSecurityNT;
        interface IOPCSecurityPrivate;
    };
```

6.2 OpcErrSec.h

```
/*++

Module Name:
 OpcErrSec.h

Author:
OPC Security Task Force

Revision History:
Release 1.0 08/18/00
     OPC security HRESULTs
--*/

/*
Facility Code Assignments:
  0000 to 0200 are reserved for Microsoft use
  (although some were inadvertently used for OPC 1.0 errors).
  0200 to 8000 are reserved for future OPC use.
  8000 to FFFF can be vendor specific.

*/

#ifndef __OPC_ERR_SEC_H
#define __OPC_ERR_SEC_H

//
//  Values are 32 bit values laid out as follows:
//
//   3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
//   1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
//  +---+-+-+-----------------------+-------------------------------+
//  |Sev|C|R|     Facility          |               Code            |
//  +---+-+-+-----------------------+-------------------------------+
//
//  where
//
//      Sev - is the severity code
//
//          00 - Success
//          01 - Informational
//          10 - Warning
//          11 - Error
//
//      C - is the Customer code flag
//
//      R - is a reserved bit
//
//      Facility - is the facility code
//
//      Code - is the facility's status code
//
//

// TODO/dj:  Although we believe there are no current conflicts, the
// HRESULT values need to be coordinated with those of other OPC
// specifications.

//
// MessageId: OPC_E_PRIVATE_ACTIVE
//
```

```
// MessageText:
//
//  OPC Security: A session using private OPC credentials is already
active.
//
#define OPC_E_PRIVATE_ACTIVE            ((HRESULT)0xC0040301L)

//
// MessageId: OPC_E_LOW_IMPERS_LEVEL
//
// MessageText:
//
//  OPC Security: Server requires higher impersonation level to access
secured data.
//
#define OPC_E_LOW_IMPERS_LEVEL          ((HRESULT)0xC0040302L)

//
// MessageId: OPC_S_LOW_AUTHN_LEVEL
//
// MessageText:
//
//  OPC Security: Server expected higher level of package privacy.
//
#define OPC_S_LOW_AUTHN_LEVEL           ((HRESULT)0x00040303L)


#endif // __OPC_ERR_SEC_H
```

6.3 Guidelines

6.3.1 DCOM Security Setup and Settings

6.3.1.1 Recommended Security Setup

The general problem of the DCOM setup has already been explained in a white paper available on the OPC Foundation Website (*OPC DCOM White Paper*, Revision 2 – Thursday, April 9, 1998). This section provides guidelines about the DCOM setup, regardless of whether or not *OPC Security* is going to be used.

There are several security configuration settings that affect application (Client or Server) calls in DCOM. These settings can be controlled either:

?? Programmatically by each individual application by calling `CoInitializeSecurity()` immediately after calling `CoInitializeEx()`.

?? Using the DCOMCNFG.EXE tool to configure settings for a given application (using what is called the application AppId).

?? Using the DCOMCNFG.EXE tool to configure default settings for all applications that don't rely on specific settings or/and don't have an AppId.

The DCOMCNFG.EXE tool is an offline tool that makes modifications to the DCOM configuration keys stored into the registry. If an application doesn't call `CoInitializeSecurity()`, the system reads the DCOM configuration from the registry at application startup time.

When using the notification mechanism between the server and the DCOM client, in terms of DCOM settings, the server is the client (caller) and the client is the server (callee). If the DCOM client is not configured to allow the server to access its call back methods, notifications will not work. The client application can avoid these problems by calling `CoInitializeSecurity()` to allow any server to access its notification interfaces.

6.3.1.1.1 Summary of DCOM Configuration Methods

DCOM settings by :	**For an OPC Server**	**For an OPC Client Application**
Calling `CoInitializeSecurity`	Recommended only for debugging purposes ; See the section *DCOM Security Bypass* below.	Recommended method to avoid any notification problems; see below for recommended arguments.
Using DCOMCNFG to configure application settings	Recommended, but don't forget (in the Identity tab) to avoid selecting the launching user	Not recommended, since not all client applications will be visible in DCOMCNFG.
Using the DCOMCNFG to configure system-wide default settings	Not recommended	Not recommended

Recommended `CoInitializeSecurity()` arguments for the client application:

```
CoInitializeSecurity(NULL, -1, NULL, NULL, RPC_C_AUTHN_LEVEL_NONE,
RPC_C_IMP_LEVEL_IDENTIFY, NULL, EOAC_NONE, NULL);
```

6.3.1.1.2 Summary of DCOM Configuration Values:

6.3.1.1.2.1 *OPC Security Using NT Credentials and DCOM Security*

The DCOM setup should be done both on the client machine and the server machine. The values indicated below assume that both client and server machines are members of an NT Domain or of NT Domains which have a trust relationship.

	Client Machine	**Server Machine**
Authentication Level(negotiated between client and server, higher level wins)	None	Connect
Impersonation Level(client level only taken into account)	Impersonate(or Delegate if required by server, see `QueryMinImpersonationLevel`)	COM selects the impersonation level as set by the client
Access Permission	None, to avoid call back problems (see previous section)	List of users defined using DCOMCNFG
Launch Permission	Not applicable	List of users defined using DCOMCNFG

In order to facilitate the first usage of an OPC server with DCOM, the install process of this server may set some DCOM registry keys. An example is to create an AppId and apply the default values defined above.

6.3.1.1.2.2 *OPC Security Using Private Credentials and Disabled Security*

The table below summarizes the DCOM security settings for clients and servers, where the server is implementing either *Disabled Security* or *OPC Security using Private Credentials* only. Servers which implement *OPC Security using Private Credentials* together with *DCOM Security* or *OPC Security Using NT Credentials* should use the security settings recommended in the previous table.

	Client Machine	**Server Machine**
Authentication Level(negotiated between client and server, higher level wins)	None, except that the client should raise the level to Privacy when calling `Logon` to protect the private credential.	None
Impersonation Level(client level only taken into account)	Identify	Identify
Access Permission	None, to avoid call back problems	None, to allow all clients to connect
Launch Permission	Not applicable	List of users defined using DCOMCNFG (if the server is running on NT)

6.3.1.2 Windows 95/98 DCOM

The latest version of DCOM for Windows 95 and Windows 98 is available on the Web at the following location:

http://www.microsoft.com/com/dcom/

6.3.1.2.1 Differences from DCOM on Windows NT

The following table summarizes the differences between DCOM95/98 and Windows NT.

For more information regarding DCOM for Windows 95 and 98 see the release notes found in the above referenced Web location.

Service Call/API/Functionality	**NT4 / W2000**	**DCOM95/98**
CoInitializeSecurity	*Yes*	*Yes*
CoQueryAuthenticationService	*Yes*	*Yes*
CoQueryProxyBlanket	*Yes*	*Yes*
CoSetProxyBlanket	*Yes*	*Yes*
CoQueryClientBlanket	*Yes*	*Yes*
IClientSecurity Interface	*Yes*	*Yes*
IserverSecurity Interface	*Yes*	*Yes*
AccessCheck	*Yes*	*No*
Impersonation	*Yes*	*No*
CoImpersonateClient	*Yes*	*No*
CoRevertToSelf	*Yes*	*No*
CLSID_DCOMAccessControl	*Yes*	*Yes*
IAccessControl Interface	*Yes*	*Yes*
Launching Servers	*Yes*	*No*
Access Security	*Yes*	*Yes*
AccessPermissions Registry Key	*Yes*	*Yes*
Authentication Levels for Clients	*All Levels*	*All Levels*
Authentication Levels for Servers/Clients Receiving callbacks		
RPC_C_AUTHN_LEVEL_NONE	*Yes*	*Yes*
RPC_C_AUTHN_LEVEL_CONNECT	*Yes*	*Yes*

6.3.1.2.1.1 Security Capabilities of DCOM95/98

The core functionality and application programming interface (API) for DCOM95/98 are identical in both Windows 95/98 and Windows NT 4.0. However, certain capabilities related to security are different because of the different security infrastructures of the operating systems.

6.3.1.2.1.2 Launch and Access Security

Controlling who can launch server-class code is not supported in DCOM95/98, because launching servers is not supported. Servers/classes must already be running in order for remote clients to connect to them and make use of their services.

6.3.1.2.1.3 Authentication Levels

DCOM95/98 clients can make DCOM calls using any authentication level. DCOM95/98 servers or clients receiving callbacks can accept only DCOM calls using RPC_C_AUTHN_LEVEL_NONE or RPC_C_AUTHN_LEVEL_CONNECT authentication levels.

6.3.1.2.1.4 Transports

DCOM95/98 supports only TCP connectivity.

6.3.2 In-Process Server Considerations

6.3.2.1 Private Credentials

There is no difference in the behavior of methods included in the *IOPCSecurityPrivate* interface for OPC Servers which execute in-process than for OPC Servers which execute locally or remotely.

6.3.2.2 NT Credentials

Some calls to NT functions which impersonate a client may behave differently when invoked by an in-process server than when invoked by a server which executes locally or remotely. In order to avoid such problems, it is recommended that servers which are configured to be invoked in-process should not attempt to impersonate the client, but should base their access decisions on the NT Access Token of the thread in which they are executing.

In addition, it is recommended that clients of in-process servers not implement any capability to dynamically change a user's NT Credentials during a session, since the behavior will be unpredictable.

Specific implications for in-process servers which support OPC Security with NT Credentials are:

?? Server implementation of *IOPCSecurityNT* can be the same as for local/remote servers, with the exception of `ChangeUser()`. In-process servers should perform no action in response to this call other than to return S_OK.

?? Clients of in-process servers should never call `ChangeUser()`, since it has no effect in this context.

?? Clients of in-process servers should not explicitly set the security context (e.g. via `CoSetProxyBlanket()`).

6.3.3 Local/Remote Server Configuration Parameters

These parameters are not mandatory. This section is only a guideline.

However, it is expected that most servers that implement the *IOPCSecurityNT* and/or *IOPCSecurityPrivate* interface (that is to say those that support *OPC Security*), support also most of the parameters described below.

6.3.3.1 DCOM Security Bypass

The goal of this parameter is to tell the server whether or not it should disable all the DCOM security in order for example, to allow a quick test.

If this parameter is ON, the server should call `CoInitializeSecurity()` method in order to grant access to its interfaces to any OPC client. If the server is not running when the client tries to get access, DCOM settings of the registry are always used to determine whether or not the client is authorized to launch the server. The access permissions will not be disabled via the call to `CoInitializeSecurity()` until the server has been launched.

If this parameter is OFF, the server should do nothing special, it should NOT call the `CoInitializeSecurity()` method, but rely on the settings of the DCOM configuration utility (Dcomcnfg).

This parameter may be supported even if the OPC server doesn't support any *OPC Security* options.

6.3.3.1.1 Sample code to disable any DCOM security

```
CoInitializeSecurity(NULL, -1, NULL, NULL,
                               RPC_C_AUTHN_LEVEL_NONE,
                               RPC_C_IMP_LEVEL_IDENTIFY, NULL,
                               EOAC_NONE, NULL);
```

6.3.3.1.2 DCOM Security and OPC Security Using NT Credentials

The goal of *DCOM Security* is to grant or to deny access to the server interfaces. The goal of *OPC Security* is to grant or to deny access to some server internal resources.

If you are dealing with *OPC Security*, it means you have access to the server interfaces: either you meet the *DCOM Security* requirements or the latter is actually disabled.

For debugging purposes it is possible to enable *OPC Security* while *DCOM Security* is disabled, but in normal situations, if *OPC Security* is enabled, *DCOM Security* should be enabled as well.

6.3.3.2 Security Auditing

The server implementer might consider providing a parameter which would allow the administrator of the OPC server to enable or disable the logging by the server of any security events into the system (for example, Windows NT) security log file.

In order to facilitate troubleshooting (in case of any problem, the server always returns E_ACCESSDENIED), the server implementer might consider providing some additional parameters to configure the level of details that should be reported.

This parameter may be only an enable/disable switch or may allow the administrator to choose which kind of event should be reported and makes sense even if the OPC server doesn't support the *OPC Security*.

6.3.4 Windows 2000 and Windows NT Considerations

All Windows 2000 guidelines are based on testing with standard released versions of the operating system.

6.3.4.1 Compatibility With Windows NT4

On the COM level the security model stays the same in Windows 2000. Only the underlying Security Service Provide (SSP) was exchanged with the superior Kerberos (RFC 1510). COM will be affected by Kerberos introduction of *delegation* and *cloaking*. Windows 2000 uses the same namespace model as NT4 and will keep using the term *domain* for the Kerberos equivalent *realm*.

6.3.4.1.1 Kerberos Credentials

The primary authentication protocol for the Windows 2000 domain is Kerberos authentication. Kerberos credentials consist of the domain and the user name (which could be in the form of Internet friendly names, such as Security@opcfoundation.org), and the Kerberos-style encrypted password. When the user logs on to the system, Windows 2000 obtains one or more Kerberos tickets to connect to network services. The Kerberos tickets represent a user's network credentials in Kerberos-based authentication.

Windows 2000 automatically manages the Kerberos ticket cache for connections to all network services. Tickets have an expiration time and occasionally need to be renewed. Ticket expiration and renewal are handled by the Kerberos security provider and associated application services. Most services, such as file system Redirector, will automatically keep session tickets up-to-date. Regular ticket renewal gives added session security by changing the sessions keys periodically.

6.3.4.2 API Compatibility

All API's used in our demo code are fully compatible:

```
LookupAccountName()
InitializeSecurityDescriptor()
InitializeAcl()
AddAccessAllowedAce()
SetSecurityDescriptorDacl()
SetSecurityDescriptorGroup()
SetSecurityDescriptorOwner()
IsValidAcl()
IsValidSecurityDescriptor()
CoQueryClientBlanket()
CoSetProxyBlanket()
CoImpersonateClient()
CoRevertToSelf()
OpenThreadToken()
AccessCheck()
```

6.3.4.2.1 New valid parameter values in Windows 2000:

`CoInitializeSecurity():`

pAuthList

The parameter *pAuthList* is required to be NULL in Windows NT 4. On Windows 2000, this parameter is a pointer to a SOLE_AUTHENTICATION_LIST, which is an array of SOLE_AUTHENTICATION_INFO structures. This list indicates the default authentication information to use for each authentication service. It applies only to clients.

dwCapabilities

Additional capabilities of the client or server. If you want to use cloaking you need to specify EOAC_STATIC_CLOAKING or EOAC_DYNAMIC_CLOAKING.

dwImpLevel

The default impersonation level for proxies. The value of this parameter applies when the process is the client. It should be a value from the RPC_C_IMP_LEVEL_xxx enumeration. Outgoing calls from the client always use the impersonation level as specified (it is not negotiated). Incoming calls to the client can be at any impersonation level. By default, all *IUnknown* calls are made with this impersonation level so even security-aware applications should set this level carefully. Only the RPC_C_IMP_LEVEL_IDENTIFY and RPC_C_IMP_LEVEL_IMPERSONATE levels are supported in Windows NT 4.0. In Windows 2000, RPC_C_IMP_LEVEL_DELEGATE is supported as well.

6.3.4.3 Delegation and impersonation differences

With the Kerberos protocol, the impersonation levels *identify*, *impersonate*, and *delegate* can be used. When a server calls `CoImpersonateClient()`, the token returned is valid off the machine for some time period between 5 minutes and 8 hours. After this time, it can be used on the server machine only. If a server is "run as activator" and the activation is done with the Kerberos protocol, the server's token will expire between 5 minutes and 8 hours after activation.

6.3.4.4 Cloaking

Prior to the release of Windows 2000, when a COM server impersonated a COM client, the process token was used to represent the client's identity. Therefore, during impersonation the client's identity was always perceived to be that of the immediate calling process. Now that cloaking is available in Windows 2000, COM servers (or any COM process) can set the cloaking capability flag to *dynamic cloaking* in a call to `CoInitializeSecurity()`.

Dynamic cloaking causes the thread token to be used to represent the client's identity during impersonation. This means that servers called on the client's behalf during impersonation see the identity of the COM client that originated the call, which is generally the desired behavior. Of course, for impersonation to succeed, the client must have given the server authority to impersonate by setting an appropriate impersonation level.

6.3.4.4.1 Types of Cloaking:

There are two types of cloaking, *static* cloaking and *dynamic* cloaking.

?? With static cloaking (EOAC_STATIC_CLOAKING), the server sees the thread token from the first call from a client to the server. For the first call, if the proxy identity was previously set during a call to `CoSetProxyBlanket()`, that proxy identity is used. However, if the proxy identity was not previously set, the thread token is used. If no thread token is present, the process token is used. For all future calls, the identity set on the first call is used.

?? When dynamic cloaking (EOAC_DYNAMIC_CLOAKING) is set, on each call the current thread token (if there is a thread token) is used to determine the client's identity. If there is no thread token, the process token is used. This type of cloaking is expensive.

6.3.4.5 Impersonation Levels

If impersonation succeeds, it means that the client has agreed to let the server "be" the client to some degree. The varying degrees of impersonation are called impersonation levels, and they indicate how much authority is given to the server when it is impersonating the client.

Currently, there are four impersonation levels: *anonymous*, *identify*, *impersonate*, and *delegate*. Prior to Windows 2000, the only supported impersonation levels were *identify* and *impersonate*. In Windows 2000, *delegate*-level impersonation is supported. The following briefly describes each impersonation level:

?? At the *anonymous* level (not currently supported) the client is anonymous to the server. The server process can impersonate the client but the impersonation token does not contain any information about the client.

?? At the *identify* level (RPC_C_IMP_LEVEL_IDENTIFY), which is the system default level, the server can obtain the client's identity. The server can impersonate the client to do ACL checks. Note: `GetUserName()` will fail while impersonating at identify level. The workaround is to impersonate, `OpenThreadToken()`, revert, call `GetTokenInformation()`, and finally, call `LookupAccountSid()`.

?? At the *impersonate* level (RPC_C_IMP_LEVEL_IMPERSONATE), the server can impersonate the client's security context while acting on behalf of the client. The server can access local resources as the client. If the server is local, it can access network resources as the client. If the server is remote, it can only access resources that are on the same machine as the server. In order for the impersonation token to be passed, you must use Cloaking, which is available in Windows 2000.

?? The *delegate* level (RPC_C_IMP_LEVEL_DELEGATE) is the most powerful impersonation level. When this level is selected, the server (whether local or remo te) can impersonate the client's security context while acting on behalf of the client. The server process can also make outgoing calls to other servers while acting on behalf of the client, using Cloaking. During impersonation, the client's credentials (both local and network) can be passed to any number of machines. This level is supported only in Windows 2000 and later versions. In order for impersonation to work at the delegate level, the following requirements must be met:

– The client must set the imp ersonation level to RPC_C_IMP_LEVEL_DELEGATE.

– The client account must not be marked "Account is sensitive and cannot be delegated" in the Active Directory Service.

– The server account must be marked with the "Trusted for delegation" attribute in the Active Directory Service.

– The computers hosting the client, the server, and any "downstream" servers must all be running Windows 2000 in a Windows 2000 domain (because the Kerberos protocol is required).

– All Win32 processes involved must be running as valid Kerberos principals., i.e. local machine accounts don’t count.

By choosing the impersonation level, the client tells the server how far it can go in impersonating the client. The client sets the impersonation level on the proxy it uses to communicate with the server.

6.3.4.6 Special Considerations

Default Authentication

Windows 2000 has a new behavior for the default authentication value RPC_C_AUTHN_LEVEL_DEFAULT. In Windows NT 4.0, this value defaults to RPC_C_AUTHN_CONNECT. In Windows 2000 this value tells DCOM to choose the authentication level using its normal security blanket negotiation algorithm.