**Access Control**

Because queues are *securable objects*, access to them is regulated by the access control model that governs access to all securable objects in Windows®. According to this model, specific operations on queues can be restricted to specific users or groups of users, called *trustees*. When a queue is created, a **SECURITY\_DESCRIPTOR** structure is associated with the queue to specify the *access rights* that are allowed or denied to specific trustees for performing specific operations on the queue.

The queue operations that can be restricted include creating, deleting, and opening the queue (for sending messages to and reading messages from the queue). Operations also include getting and setting the queue's properties and *security descriptor*.

The following are the default values of the components of the security information included in a queue's **SECURITY\_DESCRIPTOR** structure:

* Owner  
  The security identifier (SID) of the queue's creator.
* Group  
  The security identifier (SID) of the primary group to which the queue's creator belongs.
* Discretionary access control list (DACL)  
  Full control for the queue's creator. All other users (processes) can get queue properties, get queue security, and send messages to the queue. In MSMQ 3.0, anonymous users can only send messages to the queue, and the Computer\_Name$ account of a computer that belongs to a domain can only get the properties and security of the queue and receive or peek at messages in the queue.
* System access control list (SACL)  
  None.

For applications using Message Queuing Win32® APIs, the **SECURITY\_DESCRIPTOR** structure is specified by the *pSecurityDescriptor* parameter of [MQCreateQueue](http://msdn.microsoft.com/en-us/library/ms701768(v=VS.85).aspx) when the queue is created. For applications using Message Queuing COM components, the default security descriptor is automatically attached to the queue when it is created unless the optional *IsWorldReadable* parameter of the [MSMQQueueInfo.Create](http://msdn.microsoft.com/en-us/library/ms703983(v=VS.85).aspx) method is set. When this parameter is set to True, all members of the Everyone group can peek at or retrieve messages in the queue and its *journal*. You can determine whether the default security descriptor of the queue was changed to allow read access to all members of the Everyone group in the call to create the queue or at any time after the queue was created by retrieving the [MSMQQueueInfo.IsWorldReadable2](http://msdn.microsoft.com/en-us/library/ms705140(v=VS.85).aspx) property.

Before Message Queuing performs any operation on a queue, it checks the queue's security descriptor to determine whether the access rights for performing the requested operation are allowed for the user. To do this, Message Queuing checks whether the operation is restricted. If the operation is restricted, Message Queuing then checks the identity of the user to ascertain whether the restriction applies to that user. If it does, the operation is not allowed to continue.

With the exception of putting a new message in a queue, Message Queuing can verify the identity of the user from the access token attached to the process. Access tokens are produced by the system. When a user logs on, the system verifies the user's password by comparing it with information stored in the system's security database. If the password is valid, the system produces an access token and attaches it to each process started by the user.

However, Message Queuing cannot use the sender's access token to put a new message in a remote destination queue, because it is not available to the destination queue manager. Instead, it uses the security identifier (SID) that Message Queuing attached to the message when it was sent. Similar to the access token, the user's SID is attached by the application. When a message is sent by a local user account to a remote computer, no SID is attached to the message, and in MSMQ 3.0, the destination queue manager regards the sender as an anonymous user. For more information, see *access tokens* and *SIDs* in the Security section of the Microsoft® Platform SDK.

Applications can retrieve or modify the security descriptor of a queue if the users running them have sufficient permissions and privileges. For more information, see [MQGetQueueSecurity](http://msdn.microsoft.com/en-us/library/ms703223(v=VS.85).aspx) and [MQSetQueueSecurity](http://msdn.microsoft.com/en-us/library/ms705190(v=VS.85).aspx).

**Examples**

|  |  |
| --- | --- |
| For an example of | See |
| Retrieving the domain name (if applicable) and account name of a queue's owner | [C/C++ Code Example: Retrieving the Queue Owner's Name](http://msdn.microsoft.com/en-us/library/ms699826(v=VS.85).aspx) |
| Displaying lists of the permissions that are granted or denied to trustees for a queue | [C/C++ Code Example: Retrieving the Access Rights of a Queue](http://msdn.microsoft.com/en-us/library/ms700142(v=VS.85).aspx) |
| Changing the security settings of a queue to allow all members of the Everyone group to receive messages from a queue | [C/C++ Code Example: Granting Additional Access Rights](http://msdn.microsoft.com/en-us/library/ms706911(v=VS.85).aspx) |
| Creating a security descriptor that gives a specified trusted user full control over a queue | [C/C++ Code Example: Creating a Security Descriptor](http://msdn.microsoft.com/en-us/library/ms707085(v=VS.85).aspx) |

[Send comments about this topic to Microsoft.](mailto:sdkfdbk@microsoft.com?subject=Documentation%20Feedback:%20Access%20Control%20-%201e4ebaf8-9eb6-40f9-b402-505efc76f5c0)

**Component Automation**

**MQGetQueueSecurity**

The **MQGetQueueSecurity** function retrieves information from the access control *security descriptor* of the specified queue.

[Copy](javascript:CodeSnippet_CopyCode('CodeSnippetContainerCode0');" \o "Copy to clipboard.)

HRESULT APIENTRY MQGetQueueSecurity(

LPCWSTR lpwcsFormatName,

SECURITY\_INFORMATION SecurityInformation,

PSECURITY\_DESCRIPTOR pSecurityDescriptor,

DWORD nLength,

LPDWORD lpnLengthNeeded

);

## Parameters

***lpwcsFormatName***

[in] Pointer to the *format name* string of the queue whose security information will be retrieved. A public or private format name can be used to specify the queue. You can use a direct format name only to obtain information from the *security descriptor* of a local private queue. See "Remarks" for limitations on private queues.

***SecurityInformation***

[in] Specifies a **SECURITY\_INFORMATION** value that identifies the component or components of access control information being requested. The information requested is placed in the security descriptor buffer pointed to by the ***pSecurityDescriptor*** parameter. The following **SECURITY\_INFORMATION** bit flags or a combination of these bit flags can be used to identify the components of security information being retrieved.

DACL\_SECURITY\_INFORMATION

Indicates that the discretionary access control list (DACL) is being retrieved.

GROUP\_SECURITY\_INFORMATION

Indicates that the primary group *security identifier* is being retrieved.

OWNER\_SECURITY\_INFORMATION

Indicates that the owner's security identifier (SID) is being retrieved.

SACL\_SECURITY\_INFORMATION

Indicates that the system access control list (SACL) is being retrieved.

**Not Supported**

Message Queuing does not support the use of the following bit flags in this parameter.

PROTECTED\_DACL\_SECURITY\_INFORMATION

PROTECTED\_SACL\_SECURITY\_INFORMATION

UNPROTECTED\_DACL\_SECURITY\_INFORMATION

UNPROTECTED\_SACL\_SECURITY\_INFORMATION

***pSecurityDescriptor***

[out] Pointer to the **SECURITY\_DESCRIPTOR** structure (security descriptor buffer) that receives the components of the queue's security information requested. The **SECURITY\_DESCRIPTOR** structure is returned in the self-relative format. Setting this parameter to NULL is not recommended.

***nLength***

[in] Specifies the size, in bytes, of the security descriptor buffer (see ***pSecurityDescriptor***).

***lpnLengthNeeded***

[out] Pointer to a variable that indicates if any additional length is needed for the security descriptor. If the security descriptor fits in the buffer, this variable indicates the actual size of the security descriptor.

If the security descriptor buffer is too small for the security descriptor (the value of ***lpnLengthNeeded*** is greater than the value of ***nLength***), this variable indicates the size of the buffer needed to hold the security descriptor. When this happens, the security descriptor is not copied to the buffer and MQ\_ERROR\_SECURITY\_DESCRIPTOR\_TOO\_SMALL is returned.

## Return Values

MQ\_OK

Indicates success.

MQ\_ERROR\_ACCESS\_DENIED (0xC00E0025)

The access rights for getting the queue's security descriptor are not allowed for the calling process. For more information on queue access rights, see [Access Control](http://msdn.microsoft.com/en-us/library/ms700281(v=VS.85).aspx).

To change access rights of the queue, call [MQSetQueueSecurity](http://msdn.microsoft.com/en-us/library/ms705190(v=VS.85).aspx).

MQ\_ERROR\_ILLEGAL\_FORMATNAME (0xC00E001E)

The ***lpwcsFormatName*** parameter specified an illegal format name.

MQ\_ERROR\_NO\_DS (0xC00E0013)

A connection with the *directory service* cannot be established. Verify permissions for accessing the directory service.

MQ\_ERROR\_PRIVILEGE\_NOT\_HELD (0xC00E0026)

The process does not have the proper privilege to read the queue's system access control list.

MQ\_ERROR\_SECURITY\_DESCRIPTOR\_TOO\_SMALL (0xC00E0023)

The buffer pointed to by ***pSecurityDescriptor*** is too small to hold the security descriptor; the returned value of ***lpnLengthNeeded*** is greater than the supplied value of ***nLength***.

MQ\_ERROR\_UNSUPPORTED\_FORMATNAME\_OPERATION (0xC00E0020)

The format name specified in the ***lpwcsFormatName*** parameter cannot be used. You cannot reference a public queue or a remote private queue in a call to this function using a direct format name, nor can you get the security descriptor of a *journal*, *dead-letter*, or *connector queue*.

|  |
| --- |
| **http://i.msdn.microsoft.com/ms703223.note(en-us,VS.85).gifNote:** |
| Apart from generic Message Queuing error codes, this function may return ADSI and LDAP error codes. For example, LDAP\_BUSY (0x8007200E) is returned when the directory service server is busy. |

## Remarks

The queue's *security descriptor* is initially set when the queue is created (see [MQCreateQueue](http://msdn.microsoft.com/en-us/library/ms701768(v=VS.85).aspx)). Access to the following queue operations can be controlled: creating, deleting, and opening the queue for sending messages to and reading messages from the queue; getting and setting the queue's properties; and getting and setting the queue's security descriptor.

The security descriptor of a local or remote public queue can be retrieved, but you can only retrieve the security descriptor of a private queue if it is located on your local computer.

The security descriptor of a *journal*, *dead-letter*, *connector*, and *foreign queue* cannot be retrieved.

To read the security descriptor of a queue, the calling process must be allowed READ\_CONTROL access or be the owner of the queue. Access rights such as READ\_CONTROL are set when the queue is created and can be modified by calling [MQSetQueueSecurity](http://msdn.microsoft.com/en-us/library/ms705190(v=VS.85).aspx).

To read the queue's system access control list, the caller must have SE\_SECURITY\_NAME privileges on the *directory service* server (for public queues) or on the local computer (for private queues).

A public queue's security descriptor cannot be retrieved if there is no connection to the *directory service*. This restriction applies to *dependent client* computers, *independent client* computers that are working offline, and Message Queuing routing servers (FRS).

## Reading the Access Mask of an Access Control Entry

The following flags can be used to interpret access rights in the access mask of an access control entry (ACE) in the security descriptor.

* MQSEC\_DELETE\_MESSAGE
* MQSEC\_PEEK\_MESSAGE
* MQSEC\_WRITE\_MESSAGE
* MQSEC\_DELETE\_JOURNAL\_MESSAGE
* MQSEC\_SET\_QUEUE\_PROPERTIES
* MQSEC\_GET\_QUEUE\_PROPERTIES
* MQSEC\_DELETE\_QUEUE
* MQSEC\_GET\_QUEUE\_PERMISSONS
* MQSEC\_CHANGE\_QUEUE\_PERMISSIONS
* MQSEC\_TAKE\_QUEUE\_OWNERSHIP
* MQSEC\_RECEIVE\_MESSAGE   
  (MQSEC\_DELETE\_MESSAGE | MQSEC\_PEEK\_MESSAGE)
* MQSEC\_RECEIVE\_JOURNAL\_MESSAGE   
  (MQSEC\_DELETE\_JOURNAL\_MESSAGE | \  
  MQSEC\_PEEK\_MESSAGE)
* MQSEC\_QUEUE\_GENERIC\_READ  
  (MQSEC\_GET\_QUEUE\_PROPERTIES | \  
  MQSEC\_GET\_QUEUE\_PERMISSIONS | \  
  MQSEC\_GET\_RECEIVE\_MESSAGE | \  
  MQSEC\_GET\_RECEIVE\_JOURNAL\_MESSAGE)
* MQSEC\_QUEUE\_GENERIC\_WRITE  
  (MQSEC\_GET\_QUEUE\_PROPERTIES | \  
  MQSEC\_GET\_QUEUE\_PERMISSIONS | \  
  MQSEC\_GET\_WRITE\_MESSAGE)
* MQSEC\_QUEUE\_GENERIC\_ALL  
  (MQSEC\_RECEIVE\_MESSAGE | \  
  MQSEC\_RECEIVE\_MESSAGE\_JOURNAL | \  
  MQSEC\_WRITE\_MESSAGE | \  
  MQSEC\_SET\_QUEUE\_PROPERTIES | \  
  MQSEC\_GET\_QUEUE\_PROPERTIES | \  
  MQSEC\_GET\_DELETE\_QUEUE | \  
  MQSEC\_GET\_QUEUE\_PERMISSIONS | \  
  MQSEC\_GET\_CHANGE\_QUEUE\_PERMISSIONS | \  
  MQSEC\_TAKE\_QUEUE\_OWNERSHIP)

|  |  |
| --- | --- |
| For information on | See |
| Obtaining a format name for the queue | [Obtaining Format Names](http://msdn.microsoft.com/en-us/library/ms699813(v=VS.85).aspx) |
| Offline operations | [Message Queuing Offline Support](http://msdn.microsoft.com/en-us/library/ms700235(v=VS.85).aspx) |
| Access control and security descriptors | [Access Control](http://msdn.microsoft.com/en-us/library/ms700281(v=VS.85).aspx) |

## Example Code

The following code examples are included in [Using Message Queuing](http://msdn.microsoft.com/en-us/library/ms705205(v=VS.85).aspx).

|  |  |
| --- | --- |
| For an example of | See |
| Retrieving the domain name (if applicable) and account name of a queue's owner | [C/C++ Code Example: Retrieving the Queue Owner's Name](http://msdn.microsoft.com/en-us/library/ms699826(v=VS.85).aspx) |
| Displaying lists of the permissions of a queue that are granted or denied to trustees | [C/C++ Code Example: Retrieving the Access Rights of a Queue](http://msdn.microsoft.com/en-us/library/ms700142(v=VS.85).aspx) |

## Requirements

**Windows NT/2000/XP**: Included in Windows NT 4.0 SP3 and later.

**Windows 95/98/Me**: Included in Windows 95 and later.

**Header**: Declared in Mq.h.

**Library**: Use Mqrt.lib.

## See Also

#### Concepts

[Message Queuing Functions](http://msdn.microsoft.com/en-us/library/ms704996(v=VS.85).aspx)  
[MQCreateQueue](http://msdn.microsoft.com/en-us/library/ms701768(v=VS.85).aspx)  
[MQSetQueueSecurity](http://msdn.microsoft.com/en-us/library/ms705190(v=VS.85).aspx)

[Send comments about this topic to Microsoft.](mailto:sdkfdbk@microsoft.com?subject=Documentation%20Feedback:%20MQGetQueueSecurity%20-%204dfc4e3c-ceca-4f49-955f-bca4e93a5b46)

**C/C++ Code Example: Retrieving the Access Rights of a Queue**

The following example provides three application-defined functions that can be used for retrieving and displaying lists of the permissions granted to each user account mentioned in the *security descriptor* of a queue.

The first function, **GetQSecurity**, receives the computer name and queue name specified by the user as input parameters and calls [MQGetQueueSecurity](http://msdn.microsoft.com/en-us/library/ms703223(v=VS.85).aspx) with the *SecurityInformation* parameter set to DACL\_SECURITY\_INFORMATION to obtain the discretionary access control list (DACL) from the queue's security descriptor and store it in a **SECURITY\_DESCRIPTOR** structure. **MQGetQueueSecurity** can be used to obtain security information for a public queue on the local or a remote computer, or for a private queue on the local computer.

The second function, **DisplayDaclInfo**, retrieves the DACL from the security descriptor buffer and stores it in an **ACL** structure and then retrieves the individual access control entries (ACEs) from the ACL structure. This function can retrieve data only from simple access-allowed and access-denied ACEs, and not from callback and object-specific ACEs.

The third function, **DisplayPermissions**, displays the permissions granted or denied to the applicable *trustee* based on the *access rights* contained in the access mask that is passed as an input parameter.

For information about queue security, see [Access Control](http://msdn.microsoft.com/en-us/library/ms700281(v=VS.85).aspx).

An application using these functions must include the Windows.h, Stdio.h, and Mq.h header files.

**To retrieve the queue's DACL and store it in a security descriptor buffer**

1. Validate the input strings.

|  |
| --- |
| **http://i.msdn.microsoft.com/ms700142.note(en-us,VS.85).gifNote:** |
| It is the responsibility of the caller to ensure that these strings contain only valid characters and are null-terminated. |

1. Define the variables needed to obtain the format name.
2. Define the variables needed to retrieve information from the queue's security descriptor.
3. Generate the complete path name of the queue from the computer name and queue name passed to the function.
4. Call [MQPathNameToFormatName](http://msdn.microsoft.com/en-us/library/ms704126(v=VS.85).aspx) to obtain the public or private format name of the queue from its path name. This format name is used to obtain the information from the queue's security descriptor.

|  |
| --- |
| **http://i.msdn.microsoft.com/ms700142.note(en-us,VS.85).gifNote:** |
| A direct format name can be used to obtain information from the security descriptor of a local private queue only. |

1. In a loop, call [MQGetQueueSecurity](http://msdn.microsoft.com/en-us/library/ms703223(v=VS.85).aspx) twice with the *SecurityInformation* parameter set to DACL\_SECURITY\_INFORMATION to obtain the discretionary access control list (DACL) from the queue's security descriptor. The first call is made with a one-byte buffer to find out the size of the buffer needed. The second call, which is made after memory is allocated for the self-relative **SECURITY\_DESCRIPTOR** structure needed to store the information, retrieves the component of security information requested (the DACL) and stores it in the security descriptor buffer.
2. **Call DisplayDaclInfo**, the second function described in this example, to retrieve the DACL from the security descriptor buffer.
3. Free the memory allocated for the security descriptor buffer.

**Code Example**

The following code example can be run on computers with Windows NT® 4.0, Windows® 2000, and newer operating systems that have Message Queuing installed.

[Copy](javascript:CodeSnippet_CopyCode('CodeSnippetContainerCode0');)

HRESULT GetQSecurity(

LPCWSTR wszComputerName,

LPCWSTR wszQueueName

)

{

// Validate the input strings.

if (wszComputerName == NULL || wszQueueName == NULL)

{

return MQ\_ERROR\_INVALID\_PARAMETER;

}

// Define the variables needed to obtain the format name.

DWORD dwPathNameLength = 0;

WCHAR \* wszPathName = NULL;

DWORD dwFormatNameBufferLength = 256; // Format name buffer length

WCHAR wszFormatNameBuffer[256]; // Format name buffer

// Define the variables needed to retrieve the security information.

PSECURITY\_DESCRIPTOR pSecurityDescriptor = NULL; // Pointer to the security descriptor buffer

DWORD dwBufferLength = 1;

DWORD dwBufferLengthNeeded = 1;

HRESULT hr = MQ\_OK; // Define results

// Generate the complete path name of the queue.

dwPathNameLength = wcslen(wszComputerName) + wcslen(wszQueueName) + 2;

wszPathName = new WCHAR[dwPathNameLength];

if (wszPathName == NULL)

{

return MQ\_ERROR\_INSUFFICIENT\_RESOURCES;

}

memset(wszPathName, 0, dwPathNameLength\*sizeof(WCHAR));

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// You must concatenate wszComputerName, "\", and wszQueueName into

// the wszPathName buffer.

// wszPathName = wszComputerName + "\" + wszQueueName

// If the computer name is to long for the bugger, return MQ\_ERROR.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

wszPathName[dwPathNameLength - 1] = L'\0';

// Obtain the queue's format name from its path name.

hr = MQPathNameToFormatName(

wszPathName,

wszFormatNameBuffer,

&dwFormatNameBufferLength

);

if (FAILED(hr))

{

wprintf(L"The call to MQPathNameToFormatName failed. Error code: 0x%X\n", hr);

delete [] wszPathName;

return hr;

}

// Display some header information and free the memory allocated

// for the path name buffer.

wprintf(L"Retrieving queue security information for %s...\n\n", wszPathName );

delete [] wszPathName;

// Retrieve the DACL from the queue's security descriptor.

for ( ; ; )

{

pSecurityDescriptor = (PSECURITY\_DESCRIPTOR) new byte[dwBufferLength];

hr = MQGetQueueSecurity(

wszFormatNameBuffer,

DACL\_SECURITY\_INFORMATION, // Retrieving only the DACL

pSecurityDescriptor,

dwBufferLength,

&dwBufferLengthNeeded

);

if (SUCCEEDED(hr))

{

break;

}

if (hr == MQ\_ERROR\_SECURITY\_DESCRIPTOR\_TOO\_SMALL)

{

// Allocate the memory needed for the security descriptor buffer.

delete [] pSecurityDescriptor;

dwBufferLength = dwBufferLengthNeeded;

pSecurityDescriptor = (PSECURITY\_DESCRIPTOR) new byte[dwBufferLength];

if(pSecurityDescriptor == NULL)

{

wprintf(L"Memory could not be allocated for the security descriptor buffer.\n" );

return MQ\_ERROR\_INSUFFICIENT\_RESOURCES;

}

memset(pSecurityDescriptor, 0, dwBufferLength);

continue;

}

wprintf(L"The call to MQGetQueueSecurity failed. Error code: 0x%X\n", hr);

delete [] pSecurityDescriptor;

return hr;

}

hr = DisplayDaclInfo(pSecurityDescriptor, wszComputerName);

if (FAILED(hr))

{

wprintf(L"DisplayDaclInfo failed. Error code: 0x%X\n", hr);

}

else hr = MQ\_OK;

//Free the memory allocated for the security descriptor buffer.

delete [] pSecurityDescriptor;

return hr;

}

**To retrieve the DACL from the security descriptor buffer and store it in an ACL structure**

1. Validate the input parameters.
2. Create buffers that may be large enough for retrieving the domain name and account name of a trustee.
3. Call **GetSecurityDescriptorDacl** to obtain a pointer to the DACL in the security descriptor buffer. If the security descriptor contains a DACL, the function sets pDacl to the address of the DACL in the security descriptor buffer. If the security descriptor does not contain a DACL, the function sets pDacl to NULL.
4. Call **GetAclInformation** to retrieve information about the DACL in an **ACL\_SIZE\_INFORMATION** structure. This information includes the number of ACEs in the DACL.
5. Create buffers for the domain name and the account name.
6. In a loop, call **GetAce** to retrieve a pointer to each successive ACE in the DACL, call **LookupAccountSid** to obtain the domain name and account name that correspond to the SID in the ACE, and call **DisplayPermissions** (the third function described in this example) to display the permissions that are contained in the access mask found in the ACE.
7. Free the memory allocated for the domain name and account name buffers.

[Copy](javascript:CodeSnippet_CopyCode('CodeSnippetContainerCode1');" \o "Copy to clipboard.)

HRESULT DisplayDaclInfo(

PSECURITY\_DESCRIPTOR pSecurityDescriptor,

LPCWSTR wszComputerName

)

{

// Validate the input parameters.

if (pSecurityDescriptor == NULL || wszComputerName == NULL)

{

return MQ\_ERROR\_INVALID\_PARAMETER;

}

PACL pDacl = NULL;

ACL\_SIZE\_INFORMATION aclsizeinfo;

ACCESS\_ALLOWED\_ACE \* pAce = NULL;

SID\_NAME\_USE eSidType;

DWORD dwErrorCode = 0;

HRESULT hr = MQ\_OK;

// Create buffers that may be large enough.

const DWORD INITIAL\_SIZE = 256;

DWORD cchAccName = 0;

DWORD cchDomainName = 0;

DWORD dwAccBufferSize = INITIAL\_SIZE;

DWORD dwDomainBufferSize = INITIAL\_SIZE;

DWORD cAce;

WCHAR \* wszAccName = NULL;

WCHAR \* wszDomainName = NULL;

// Retrieve a pointer to the DACL in the security descriptor.

BOOL fDaclPresent = FALSE;

BOOL fDaclDefaulted = TRUE;

if (GetSecurityDescriptorDacl(

pSecurityDescriptor,

&fDaclPresent,

&pDacl,

&fDaclDefaulted

) == FALSE)

{

dwErrorCode = GetLastError();

wprintf(L"GetSecurityDescriptorDacl failed. GetLastError returned: %d\n", dwErrorCode);

return HRESULT\_FROM\_WIN32(dwErrorCode);

}

// Check whether no DACL or a NULL DACL was retrieved from the security descriptor buffer.

if (fDaclPresent == FALSE || pDacl == NULL)

{

wprintf(L"No DACL was found (all access is denied), or a NULL DACL (unrestricted access) was found.\n");

return MQ\_OK;

}

// Retrieve the ACL\_SIZE\_INFORMATION structure to find the number of ACEs in the DACL.

if (GetAclInformation(

pDacl,

&aclsizeinfo,

sizeof(aclsizeinfo),

AclSizeInformation

) == FALSE)

{

dwErrorCode = GetLastError();

wprintf(L"GetAclInformation failed. GetLastError returned: %d\n", dwErrorCode);

return HRESULT\_FROM\_WIN32(dwErrorCode);

}

// Create buffers for the account name and the domain name.

wszAccName = new WCHAR[dwAccBufferSize];

if (wszAccName == NULL)

{

return MQ\_ERROR\_INSUFFICIENT\_RESOURCES;

}

wszDomainName = new WCHAR[dwDomainBufferSize];

if (wszDomainName == NULL)

{

return MQ\_ERROR\_INSUFFICIENT\_RESOURCES;

}

memset(wszAccName, 0, dwAccBufferSize\*sizeof(WCHAR));

memset(wszDomainName, 0, dwDomainBufferSize\*sizeof(WCHAR));

// Set the computer name string to NULL for the local computer.

if (wcscmp(wszComputerName, L".") == 0)

{

wszComputerName = L"\0";

}

// Loop through the ACEs and display the information.

for (cAce = 0; cAce < aclsizeinfo.AceCount && hr == MQ\_OK; cAce++)

{

// Get ACE info

if (GetAce(

pDacl,

cAce,

(LPVOID\*)&pAce

) == FALSE)

{

wprintf(L"GetAce failed. GetLastError returned: %d\n", GetLastError());

continue;

}

// Obtain the account name and domain name for the SID in the ACE.

for ( ; ; )

{

// Set the character-count variables to the buffer sizes.

cchAccName = dwAccBufferSize;

cchDomainName = dwDomainBufferSize;

if (LookupAccountSidW(

wszComputerName, // NULL for the local computer

&pAce->SidStart,

wszAccName,

&cchAccName,

wszDomainName,

&cchDomainName,

&eSidType

) == TRUE)

{

break;

}

// Check if one of the buffers was too small.

if ((cchAccName > dwAccBufferSize) || (cchDomainName > dwDomainBufferSize))

{

// Reallocate memory for the buffers and try again.

wprintf(L"The name buffers were too small. They will be reallocated.\n");

delete [] wszAccName;

delete [] wszDomainName;

wszAccName = new WCHAR[cchAccName];

if (wszAccName == NULL)

{

return MQ\_ERROR\_INSUFFICIENT\_RESOURCES;

}

wszDomainName = new WCHAR[cchDomainName];

if (wszDomainName == NULL)

{

return MQ\_ERROR\_INSUFFICIENT\_RESOURCES;

}

memset(wszAccName, 0, cchAccName\*sizeof(WCHAR));

memset(wszDomainName, 0, cchDomainName\*sizeof(WCHAR));

dwAccBufferSize = cchAccName;

dwDomainBufferSize = cchDomainName;

continue;

}

// Something went wrong in the call to LookupAccountSid.

// Check if an unexpected error occurred.

if (GetLastError() == ERROR\_NONE\_MAPPED)

{

wprintf(L"An unexpected error occurred during the call to LookupAccountSid. A name could not be found for the SID.\n" );

wszDomainName[0] = L'\0';

if (dwAccBufferSize > wcslen(L"!Unknown!"))

{

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// You must copy the string "!Unknown!" into the

// wszAccName buffer.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

wszAccName[dwAccBufferSize - 1] = L'\0';

}

break;

}

else

{

dwErrorCode = GetLastError();

wprintf(L"LookupAccountSid failed. GetLastError returned: %d\n", dwErrorCode);

delete [] wszAccName;

delete [] wszDomainName;

return HRESULT\_FROM\_WIN32(dwErrorCode);

}

}

switch(pAce->Header.AceType)

{

case ACCESS\_ALLOWED\_ACE\_TYPE:

if (wszDomainName[0] == 0)

{

wprintf(L"\nPermissions granted to %s\n", wszAccName);

}

else wprintf(L"\nPermissions granted to %s\\%s\n", wszDomainName, wszAccName);

DisplayPermissions(pAce->Mask);

break;

case ACCESS\_DENIED\_ACE\_TYPE:

if (wszDomainName[0] == 0)

{

wprintf(L"\nPermissions denied to %s\n", wszAccName);

}

else wprintf(L"\nPermissions denied to %s\\%s\n", wszDomainName, wszAccName);

DisplayPermissions(pAce->Mask);

break;

default:

wprintf(L"Unknown ACE Type");

}

}

// Free memory allocated for buffers.

delete [] wszAccName;

delete [] wszDomainName;

return MQ\_OK;

}

**To display the permissions contained in an access mask**

1. For each permission, compare the corresponding Message Queuing constant to the access mask passed to the function.
2. If the bits of a permission are contained in the access mask, display the name of the permission.

[Copy](javascript:CodeSnippet_CopyCode('CodeSnippetContainerCode2');" \o "Copy to clipboard.)

HRESULT DisplayPermissions(

ACCESS\_MASK amMask

)

{

if ((amMask & MQSEC\_QUEUE\_GENERIC\_ALL) == MQSEC\_QUEUE\_GENERIC\_ALL)

{

wprintf(L"\tFull Control\n");

}

if ((amMask & MQSEC\_DELETE\_QUEUE) == MQSEC\_DELETE\_QUEUE)

{

wprintf(L"\tDelete\n");

}

if ((amMask & MQSEC\_RECEIVE\_MESSAGE) == MQSEC\_RECEIVE\_MESSAGE)

{

wprintf(L"\tReceive Message\n");

}

if ((amMask & MQSEC\_DELETE\_MESSAGE) == MQSEC\_DELETE\_MESSAGE)

{

wprintf(L"\tDelete Message\n");

}

if ((amMask & MQSEC\_PEEK\_MESSAGE) == MQSEC\_PEEK\_MESSAGE)

{

wprintf(L"\tPeek Message\n");

}

if ((amMask & MQSEC\_RECEIVE\_JOURNAL\_MESSAGE) == MQSEC\_RECEIVE\_JOURNAL\_MESSAGE)

{

wprintf(L"\tReceive Journal Message\n");

}

if ((amMask & MQSEC\_DELETE\_JOURNAL\_MESSAGE) == MQSEC\_DELETE\_JOURNAL\_MESSAGE)

{

wprintf(L"\tDelete Journal Message\n");

}

if ((amMask & MQSEC\_GET\_QUEUE\_PROPERTIES) == MQSEC\_GET\_QUEUE\_PROPERTIES)

{

wprintf(L"\tGet Properties\n");

}

if ((amMask & MQSEC\_SET\_QUEUE\_PROPERTIES) == MQSEC\_SET\_QUEUE\_PROPERTIES)

{

wprintf(L"\tSet Properties\n");

}

if ((amMask & MQSEC\_GET\_QUEUE\_PERMISSIONS) == MQSEC\_GET\_QUEUE\_PERMISSIONS)

{

wprintf(L"\tGet Permissions\n");

}

if ((amMask & MQSEC\_CHANGE\_QUEUE\_PERMISSIONS) == MQSEC\_CHANGE\_QUEUE\_PERMISSIONS)

{

wprintf(L"\tSet Permissions\n");

}

if ((amMask & MQSEC\_TAKE\_QUEUE\_OWNERSHIP) == MQSEC\_TAKE\_QUEUE\_OWNERSHIP)

{

wprintf(L"\tTake Ownership\n");

}

if ((amMask & MQSEC\_WRITE\_MESSAGE) == MQSEC\_WRITE\_MESSAGE)

{

wprintf(L"\tSend Message\n");

}

return S\_OK;

}

[Send comments about this topic to Microsoft.](mailto:sdkfdbk@microsoft.com?subject=Documentation%20Feedback:%20C/C++%20Code%20Example:%20Retrieving%20the%20Access%20Rights%20of%20a%20Queue%20-%2019843119-6057-4394-bd99-1e9ea0cf47e8)