Windows Access Control List (ACL) 2

What do we have in this session?

1. Access Control Lists (ACLs)
2. Object-specific ACEs
3. Trustees
4. Access Rights and Access Masks
5. ACCESS_MASK
6. Access Mask format
7. Generic Access Rights
8. Standard Access Rights
9. SACL Access Right
10. Directory Services Access Rights
11. How Security Descriptors are Set on New Directory Objects
12. Default Security Descriptor

Abilities that supposed to be acquired in this session are:

1. Able to understand Access Control Lists (ACLs).
2. Able to understand Access Trustees.
3. Able to understand Trustee and Security Identifier (SID).
4. Able to understand Discretionary Access Masks, Access Rights.

Access Control Lists (ACLs)

An ACL is a list of ACE. Each ACE in an ACL identifies a trustee and specifies the access rights allowed, denied, or audited for that trustee. The security descriptor for a securable object can contain two types of ACLs:

1. Discretionary Access Control List (DACL).
2. Security Access Control List (SACL).

A DACL identifies the trustees that are allowed or denied access to a securable object. When a process tries to access a securable object, the system checks the ACEs in the object's DACL to determine whether to grant access to it.
The system checks the ACEs in sequence until it finds one or more ACEs that allow all the requested access rights, or until any of the requested access rights are denied. However the following are not good terms regarding the DACL:

1. **If the object does not have a DACL, the system grants full access to everyone**.
2. **If the object's DACL has no ACEs, the system denies all attempts to access the object because the DACL does not allow any access rights**.

Each ACE specifies the types of access attempts by a specified trustee that cause the system to generate a record in the security event log. An ACE in a SACL can generate audit records when an access attempt fails, when it succeeds, or both. In future releases, a SACL will also be able to raise an alarm when an unauthorized user attempts to gain access to an object. ACLs also provide access control to an Active Directory directory service objects. Active Directory Service Interfaces (ADSI) includes routines to create and modify the contents of these ACLs.

Access Control Entries (ACEs)

An ACE is an element in an ACL. An ACL can have zero or more ACEs. Each ACE controls or monitors access to an object by a specified trustee. There are six types of ACEs, three of which are supported by all securable objects. The other three types are Object-specific ACEs supported by directory service objects. All types of ACEs contain the following access control information:

1. A SID that identifies the trustee to which the ACE applies.
2. An access mask that specifies the access rights controlled by the ACE.
3. A flag that indicates the type of ACE.
4. A set of bit flags that determine whether child containers or objects can inherit the ACE from the primary object to which the ACL is attached.

The following table lists the three ACE types supported by all securable objects.

Type	Description
Access-denied ACE	Used in a DACL to deny access rights to a trustee.
Access-allowed ACE	Used in a DACL to allow access rights to a trustee.
System-audit ACE	Used in a SACL to generate an audit record when the trustee attempts to exercise the specified access rights.

Table 6

Object-specific ACEs

Object-specific ACEs are supported for directory service (DS) objects. An object-specific ACE contains a pair of globally unique identifiers (GUIDs) that expand the ways in which the ACE can protect an object.

GUID	Description
ObjectType	Identifies one of the following: A type of child object. The ACE controls the right to create a specified type of child object. A property set or property. The ACE controls the right to read or write the property or property set. An extended right. The ACE controls the right to perform the operation associated with the extended right. A validated write. The ACE controls the right to perform certain write operations. These validated write permissions, defined and exposed in the ACL Editor, provide permissions for validated writes of properties rather than unchecked low-level writes of any value to a property that is granted with a "write property" permission.
InheritedObjectType	Indicates the type of child object that can inherit the ACE. Inheritance is also controlled by the inheritance flags in the ACE_HEADER, as well as by any protection against inheritance placed on the child objects.

Table 7

Three types of object-specific ACEs are supported. System-alarm object ACEs are not currently supported.

Type	Description
Access-denied object ACE	Used in a DACL to deny a trustee access to a property or property set on the object, or to limit ACE inheritance to a specified type of child object. Uses the ACCESS_DENIED_OBJECT_ACE structure.
Access-allowed object ACE	Used in a DACL to allow a trustee access to a property or property set on the object, or to limit ACE inheritance to a specified type of child object. Uses the ACCESS_ALLOWED_OBJECT_ACE structure.
System-audit object ACE	Used in a SACL to log a trustee's attempts to access a property or property set on the object, or to limit ACE inheritance to a specified type of child object. Uses the SYSTEM_AUDIT_OBJECT_ACE structure.

Table 8.

Any ACL that contains an object-specific ACE must use the revision ACL_REVISION_DS.

Trustees

A trustee is the **user account**, **group account**, or **logon session to which an ACE applies**. Each ACE in an ACL has one SID that identifies a trustee. User accounts include accounts that human users or programs such as Windows Services use to log on to the local computer. Group accounts cannot be used to log on to a computer, but they are useful in ACEs to allow or deny a set of access rights to one or more user accounts. A logon SID that identifies the current logon session is useful to allow or deny access rights only until the user logs off. A logon SID is a SID that identifies a logon session. You can use the logon SID in a DACL to control access during a logon session. A logon SID is valid until the user logs off. A logon SID is unique while the computer is running; no other logon session will have the same logon SID. However, the set of possible logon SIDs is reset when the computer starts up. To retrieve the logon SID from an access token, call the GetTokenInformation() function for TokenGroups. The access control functions use the TRUSTEE structure to identify a trustee. The TRUSTEE structure enables you to use a name string or a SID to identify a trustee. If you use a name, the functions that create an ACE from the TRUSTEE structure perform the task of allocating the SID buffers and looking up the SID that corresponds to the account name. There are two helper functions, BuildTrusteeWithSid() and BuildTrusteeWithName(), that initialize a TRUSTEE structure with a specified SID or name. BuildTrusteeWithObjectsAndSid() and BuildTrusteeWithObjectsAndName() allow you to initialize a TRUSTEE structure with object-specific ACE information. Three other helper functions, GetTrusteeForm(), GetTrusteeName(), and GetTrusteeType(), retrieve the values of the various members of a TRUSTEE structure. The ptstrName member of the TRUSTEE structure can be a pointer to an OBJECTS_AND_NAME or OBJECTS_AND_SID structure. These structures specify information about an object-specific ACE in addition to a trustee name or SID. This enables functions such as SetEntriesInAcl() and GetExplicitEntriesFromAcl() to store object-specific ACE information in the Trustee member of the EXPLICIT_ACCESS structure.

Access Rights and Access Masks

An access right **is a bit flag that corresponds to a particular set of operations that a thread can perform on a securable object.** For example, a registry key has the KEY_SET_VALUE access right, which corresponds to the ability of a thread to set a value under the key. If a thread tries to perform an operation on an object, but does not have the necessary access right to the object, the system does not carry out the operation. An access mask is a 32-bit value whose bits correspond to the access rights supported by an object.

ACCESS_MASK Data Type

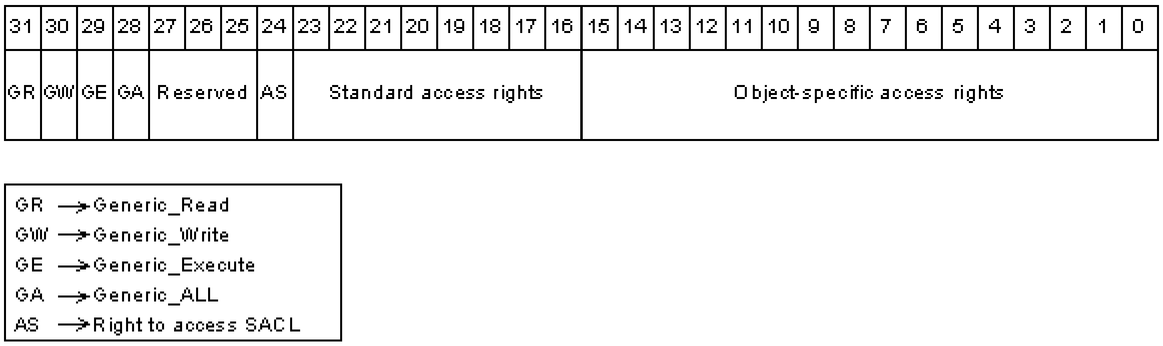
The ACCESS_MASK data type is a double word value that defines standard, specific, and generic rights.

```
typedef DWORD ACCESS_MASK;
```

These rights are used in ACEs and are the primary means of specifying the requested or granted access to an object.

Access Mask format

All securable objects arrange their access rights by using the access mask format as shown in the following Figure.



In this format, the low-order 16 bits are for object-specific access rights, the next 8 bits are for standard access rights, which apply to most types of objects, and the 4 high-order bits are used to specify generic access rights that each object type can map to a set of standard and object-specific rights. The ACCESS_SYSTEM_SECURITY bit corresponds to the right to access the object's SACL. The bits in this value are allocated as follows.

Bits	Meaning
0 through 15	Specific rights. Contains the access mask specific to the object type associated with the mask.
16 through 23	Standard rights. Contains the object's standard access rights.
24	Access system security (ACCESS_SYSTEM_SECURITY). It is used to indicate access to a SACL. This type of access requires the calling process to have the SE_SECURITY_NAME (Manage auditing and security log) privilege. If this flag is set in the access mask of an audit access ACE (successful or unsuccessful access), the SACL access will be audited.

25	**Maximum allowed (MAXIMUM_ALLOWED)**.
26 through 27	Reserved.
28	Generic all (GENERIC_ALL).
29	Generic execute (GENERIC_EXECUTE).
30	Generic write (GENERIC_WRITE).
31	Generic read (GENERIC_READ).

Table 9

The standard rights bits, 16 to 23, contain the object's standard access rights and can be a combination of the following predefined flags.

Bit	Flag	Meaning
16	DELETE	Delete access.
17	READ_CONTROL	Read access to the owner, group, and DACL of the security descriptor.
18	WRITE_DAC	Write access to the DACL.
19	WRITE_OWNER	Write access to owner.
20	SYNCHRONIZE	Synchronize access.

Table 10

The following constants represent the specific and standard access rights that can be used as #define preprocessor directives:

1.	#define SPECIFIC_RIGHTS_ALL	0x0000FFFF
2.	#define STANDARD_RIGHTS_REQUIRED	0x000F0000
3.	#define STANDARD_RIGHTS_ALL	0x001F0000

All Windows securable objects use an access mask format that includes bits for the following types of access rights:

1. Generic access rights.
2. Standard access rights.
3. SACL access right.
4. Directory services access rights.

Generic Access Rights

Securable objects use an access mask format in which the four high-order bits specify generic access rights. Each type of securable object maps these bits to a set of its standard and object-specific access rights. For example, a Windows file object maps the GENERIC_READ bit to the READ_CONTROL and SYNCHRONIZE standard access rights and to the FILE_READ_DATA, FILE_READ_EA, and FILE_READ_ATTRIBUTES object-specific access rights. Other types of objects map the GENERIC_READ bit to whatever set of access rights is appropriate for that type of object. You can use generic access rights to specify the type of access you need when you are opening a handle to an object. This is typically simpler than specifying all the corresponding standard and specific rights. The following table shows the constants defined for the generic access rights. Applications that define private securable objects can also use the generic access rights.

Constant	Generic meaning
GENERIC_ALL	Read, write, and execute access.
GENERIC_EXECUTE	Execute access.
GENERIC_READ	Read access.
GENERIC_WRITE	Write access.

Table 11

Standard Access Rights

Each type of securable object has **a set of access rights that correspond to operations specific to that type of object**. In addition to these object-specific access rights, there is a set of standard access rights that correspond to operations common to most types of securable objects. The access mask format includes a set of bits for the standard access rights. The following table shows the Windows constants defined for the standard access rights.

Constant	Meaning
DELETE	The right to delete the object.
READ_CONTROL	The right to read the information in the object's security descriptor, not including the information in the SACL.
SYNCHRONIZE	The right to use the object for synchronization. This enables a thread to wait until the object is in the signaled state. Some object types do not support this access right.
WRITE_DAC	The right to modify the DACL in the object's security descriptor.
WRITE_OWNER	The right to change the owner in the object's security descriptor.

Table 12

The Windows API also defines the following combinations of the standard access rights constants.

Constant	Meaning
STANDARD_RIGHTS_ALL	Combines DELETE, READ_CONTROL, WRITE_DAC, WRITE_OWNER, and SYNCHRONIZE access.
STANDARD_RIGHTS_EXECUTE	Currently defined to equal READ_CONTROL.
STANDARD_RIGHTS_READ	Currently defined to equal READ_CONTROL.
STANDARD_RIGHTS_REQUIRED	Combines DELETE, READ_CONTROL, WRITE_DAC, and WRITE_OWNER access.
STANDARD_RIGHTS_WRITE	Currently defined to equal READ_CONTROL.

Table 13

SACL Access Right

The ACCESS_SYSTEM_SECURITY access right controls the ability to get or set the SACL in an object's security descriptor. The system grants this access right only if the SE_SECURITY_NAME privilege is enabled in the access token of the requesting thread. A SACL contains ACEs that specify the types of access attempts that generate audit reports. Each ACE identifies a trustee, a set of access rights, and a set of flags that indicate whether the system generates audit messages for failed access attempts, successful access attempts, or both. SACL will be discussed in detail under Audit topic later on.

Directory Services Access Rights

Each Active Directory object has a **security descriptor assigned to it**. A set of trustee rights specific to directory service objects can be set within these security descriptors. These rights are listed in the following table.

Rights	Meaning
ACTRL_DS_OPEN	Open a DS object.
ACTRL_DS_CREATE_CHILD	Create a child DS object.
ACTRL_DS_DELETE_CHILD	Delete a child DS object.
ACTRL_DS_LIST	Enumerate a DS object.
ACTRL_DS_READ_PROP	Read the properties of a DS object.
ACTRL_DS_WRITE_PROP	Write properties for a DS object.

ACTRL_DS_SELF	Access allowed only after validated rights checks supported by the object are performed. This flag can be used alone to perform all validated rights checks of the object or it can be combined with an identifier of a specific validated right to perform only that check.
ACTRL_DS_DELETE_TREE	Delete a tree of DS objects.
ACTRL_DS_LIST_OBJECT	List a tree of DS objects.
ACTRL_DS_CONTROL_ACCESS	Access allowed only after extended rights checks supported by the object are performed. This flag can be used alone to perform all extended rights checks on the object or it can be combined with an identifier of a specific extended right to perform only that check.

Table 14

When a thread tries to open a handle to an object, the thread typically specifies an access mask to request a set of access rights. For example, an application that needs to set and query the values of a registry key can open the key by using an access mask to request the KEY_SET_VALUE and KEY_QUERY_VALUE access rights.

How Security Descriptors are Set on New Directory Objects

When you create a new object in the Active Directory, you can explicitly create a security descriptor and then set that security descriptor as the object's nTSecurityDescriptor property. Active Directory uses the following rules to set the DACL in the new object's security descriptor:

1. If you explicitly specify a security descriptor when you create the object, the system merges any inheritable ACEs from the parent object into the specified DACL unless the SE_DACL_PROTECTED bit is set in the security descriptor's control bits.
2. If you do not specify a security descriptor, the system builds the object's DACL by merging any inheritable ACEs from the parent object into the default DACL from the classSchema object for the object's class.
3. If the schema does not have a default DACL, the object's DACL is the default DACL from the primary or impersonation token of the creator.
4. If there is no specified, inherited, or default DACL, the system creates the object with no DACL, which allows everyone full access to the object.

The system uses a similar algorithm to build a SACL for a directory service object. The owner and primary group in the new object's security descriptor are set to the values you specify in the nTSecurityDescriptor property when you create the object. If you do not set these values, Active Directory uses the rules, listed in the following table, to set them.

Rule	Description
Owner	The owner in a default security descriptor is set to the default owner SID from the primary or impersonation token of the creating process. For most users, the default owner SID is the same as the SID that identifies the user's account. Be aware that for users who are members of the built-in administrators group, the system automatically sets the default owner SID in the access token to the administrators group; therefore, objects created by a member of the administrators group are typically owned by the administrators group. To get or set the default owner in an access token, call the GetTokenInformation() or SetTokenInformation() function with the TOKEN_OWNER structure.
Primary Group	The primary group in a default security descriptor is set to the default primary group from the creator's primary or impersonation token. Be aware that primary group is not used in the context of Active Directory.

Table 15

Default Security Descriptor

With Active Directory you can also specify default security for each type of object. This is specified in the defaultSecurityDescriptor attribute in the classSchema object definition in the Active Directory schema. This security descriptor is used to provide default protection on the object if there is no security descriptor specified during the creation of the object. Take note that ACEs from a default security descriptor are handled as if they were specified as part of object creation. Therefore, the default ACEs are placed preceding inherited ACEs and override them as appropriate. The defaultSecurityDescriptor is specified in a special string format using the Security Descriptor Definition Language (SDDL). Two functions can be used to convert binary form of the security descriptor to string format and vice versa. These functions are:

1. ConvertSecurityDescriptorToStringSecurityDescriptor().
2. ConvertStringSecurityDescriptorToSecurityDescriptor().