Authorization Services C Reference

Security > Authorization

2005-11-08

Contents

Authorization Services C Reference

Framework:	Security
Declared in	Authorization.h AuthorizationDB.h AuthorizationTags.h

Overview

Authorization Services is an API that facilitates access control to restricted areas of the operating system and allows you to restrict a user's access to particular features in your Mac OS X application. Authorization Services is used in

Note: This document was previously titled *Authorization Services Reference*.

- software that restricts access to its own tools
- applications that call system tools
- software installers that install privileged tools or require access to restricted areas of the operating system

A companion volume to *Authorization Services C Reference* is Performing Privileged Operations With Authorization Services, which explains the concepts behind authorization and provides examples of how to use Authorization Services. Objective-C methods that are equivalent to several of the functions in this document are described in Authorization Services Objective-C Reference.

Authorization Services is available in Mac OS X v10.0 and later as part of the Security framework.

Functions by Task

Creating and Releasing Authorization References

`AuthorizationCreate` (page 10)
    Creates a new authorization reference and provides an option to authorize or preauthorize rights.

`AuthorizationFree` (page 13)
    Frees the memory associated with an authorization reference.

Requesting Rights and Credentials

`AuthorizationCopyRights` (page 8)
: Authorizes and preauthorizes rights.

`AuthorizationCopyInfo` (page 6)
: Retrieves side-band data such as the user name and other information gathered during evaluation of authorization.

`AuthorizationFreeItemSet` (page 14)
: Frees the memory associated with an authorization set.

Externalizing and Internalizing Authorization References

`AuthorizationMakeExternalForm` (page 14)
: Creates an external representation of an authorization reference.

`AuthorizationCreateFromExternalForm` (page 11)
: Internalizes the external representation of an authorization reference.

Modifying the Policy Database

`AuthorizationRightGet` (page 15)
: Retrieves a right definition as a dictionary.

`AuthorizationRightSet` (page 16)
: Creates or updates a right entry in the policy database.

`AuthorizationRightRemove` (page 16)
: Removes a right from the policy database.

Executing With Root Privileges

`AuthorizationExecuteWithPrivileges` (page 12)
: Runs an executable tool with root privileges.

`AuthorizationCopyPrivilegedReference` (page 7)
: Retrieves the authorization reference passed by the `AuthorizationExecuteWithPrivileges` function.

Functions

AuthorizationCopyInfo

Retrieves side-band data such as the user name and other information gathered during evaluation of authorization.

```
OSStatus AuthorizationCopyInfo (
   AuthorizationRef authorization,
   AuthorizationString tag,
   AuthorizationItemSet **info
);
```

Parameters

`authorization`
: An authorization reference referring to the authorization session.

`tag`
: An authorization string specifying the type of data the Security Server should return. Pass `NULL` to retrieve all available information.

`info`
: A pointer to an authorization set the Security Server creates. On return, this set contains side-band authorization data. When this set is no longer needed, free the memory associated with it by calling the function `AuthorizationFreeItemSet` (page 14).

Return Value
A result code. See "Authorization Services Result Codes" (page 26).

Discussion
An authorization plug-in can store the results of an authentication operation by calling the `SetContextValue` function. You can use the `AuthorizationCopyInfo` function to retrieve this information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Authorization.h`

AuthorizationCopyPrivilegedReference

Retrieves the authorization reference passed by the `AuthorizationExecuteWithPrivileges` function.

```
OSStatus AuthorizationCopyPrivilegedReference (
   AuthorizationRef *authorization,
   AuthorizationFlags flags
);
```

Parameters

`authorization`
: A pointer to an authorization reference. The Security Server allocates the authorization reference for you, so you do not need to call the function `AuthorizationCreate` (page 10). On return, it points to a copy of the authorization reference used in the call to the `AuthorizationExecuteWithPrivileges` (page 12) function.

`flags`
: Reserved options. Pass the `kAuthorizationFlagDefaults` constant.

Return Value
A result code. See "Authorization Services Result Codes" (page 26).

Discussion

This function retrieves the authorization reference you pass in the function `AuthorizationExecuteWithPrivileges` (page 12). The new process can use the authorization reference to verify authorizations obtained by the calling process.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BSDLLCTest

MoreIsBetter

QISA

Declared In

`Authorization.h`

AuthorizationCopyRights

Authorizes and preauthorizes rights.

```
OSStatus AuthorizationCopyRights (
   AuthorizationRef authorization,
   const AuthorizationRights *rights,
   const AuthorizationEnvironment *environment,
   AuthorizationFlags flags,
   AuthorizationRights **authorizedRights
);
```

Parameters

`authorization`

An authorization reference referring to the authorization session.

`rights`

A pointer to a set of authorization rights you create. Pass `NULL` if the application requires no rights at this time.

`environment`

Data used when authorizing or preauthorizing rights. Not used in Mac OS X v10.2 and earlier. In Mac OS X v10.3 and later, you can pass icon or prompt data to be used in the authentication dialog box. In Mac OS X v10.4 and later, you can also pass a user name and password in order to authorize a user without displaying the authentication dialog box. Possible values for this parameter are listed in `Security.framework/Headers/AuthorizationTags.h`. The data passed in this parameter is not stored in the authorization reference; it is used only during authorization. If you are not passing any data in this parameter, pass the constant `kAuthorizationEmptyEnvironment`.

flags

A bit mask for specifying authorization options. Use the following option sets.

- Pass the constant `kAuthorizationFlagDefaults` if no options are necessary.
- Specify the `kAuthorizationFlagExtendRights` mask to request rights. You can also specify the `kAuthorizationFlagInteractionAllowed` mask to allow user interaction.
- Specify the `kAuthorizationFlagPartialRights` and `kAuthorizationFlagExtendRights` masks to request partial rights. You can also specify the `kAuthorizationFlagInteractionAllowed` mask to allow user interaction.
- Specify the `kAuthorizationFlagPreAuthorize` and `kAuthorizationFlagExtendRights` masks to preauthorize rights.
- Specify the `kAuthorizationFlagDestroyRights` mask to prevent the Security Server from preserving the rights obtained during this call.

authorizedRights

A pointer to a newly allocated `AuthorizationRights` structure. On return, this structure contains the rights granted by the Security framework. If you do not require this information, pass `NULL`. If you specify the `kAuthorizationFlagPreAuthorize` mask in the `flags` parameter, the method returns all the requested rights, including those not granted, but the flags of the rights that could not be preauthorized include the `kAuthorizationFlagCanNotPreAuthorize` bit.

Free the memory associated with this set by calling the function `AuthorizationFreeItemSet` (page 14).

Return Value

A result code. See "Authorization Services Result Codes" (page 26).

Discussion

There are three main reasons to use this function. The first reason is to preauthorize rights by specifying the `kAuthorizationFlagPreAuthorize`, `kAuthorizationFlagInteractionAllowed`, and `kAuthorizationFlagExtendRights` masks as authorization options. Preauthorization is most useful when a right has a zero timeout. For example, you can preauthorize in the application and if it succeeds, call the helper tool and request authorization. This eliminates calling the helper tool if the Security Server cannot later authorize the specified rights.

The second reason to use this function is to authorize rights before performing a privileged operation by specifying the `kAuthorizationFlagInteractionAllowed`, and `kAuthorizationFlagExtendRights` masks as authorization options.

The third reason to use this function is to authorize partial rights. By specifying the `kAuthorizationFlagPartialRights`, `kAuthorizationFlagInteractionAllowed`, and `kAuthorizationFlagExtendRights` masks as authorization options, the Security Server grants all rights it can authorize. On return, the authorized set contains all the rights.

If you do not specify the `kAuthorizationFlagPartialRights` mask and the Security Server denies at least one right, then the status of this function on return is `errAuthorizationDenied`.

If you do not specify the `kAuthorizationFlagInteractionAllowed` mask and the Security Server requires user interaction, then the status of this function on return is `errAuthorizationInteractionNotAllowed`.

If you specify the `kAuthorizationFlagInteractionAllowed` mask and the user cancels the authentication process, then the status of this function on return is `errAuthorizationCanceled`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AuthForAll

BSDLLCTest

MoreIsBetter

Declared In

`Authorization.h`

AuthorizationCreate

Creates a new authorization reference and provides an option to authorize or preauthorize rights.

```
OSStatus AuthorizationCreate (
   const AuthorizationRights *rights,
   const AuthorizationEnvironment *environment,
   AuthorizationFlags flags,
   AuthorizationRef *authorization
);
```

Parameters

`rights`

A pointer to a set of authorization rights you create. Pass `NULL` if the application requires no rights at this time.

`environment`

Data used when authorizing or preauthorizing rights. Not used in Mac OS X v10.2 and earlier. In Mac OS X v10.3 and later, you can pass icon or prompt data to be used in the authentication dialog box. In Mac OS X v10.4 and later, you can also pass a user name and password in order to authorize a user without user interaction. Possible values for this parameter are listed in `Security.framework/Headers/AuthorizationTags.h`. The data passed in this parameter is not stored in the authorization reference; it is used only during authorization. If you are not passing any data in this parameter, pass the constant `kAuthorizationEmptyEnvironment`.

`flags`

A bit mask for specifying authorization options. Use the following option sets.

- Pass the constant `kAuthorizationFlagDefaults` if no options are necessary.
- Specify the `kAuthorizationFlagExtendRights` mask to request rights. You can also specify the `kAuthorizationFlagInteractionAllowed` mask to allow user interaction.
- Specify the `kAuthorizationFlagPartialRights` and `kAuthorizationFlagExtendRights` masks to request partial rights. You can also specify the `kAuthorizationFlagInteractionAllowed` mask to allow user interaction.
- Specify the `kAuthorizationFlagPreAuthorize` and `kAuthorizationFlagExtendRights` masks to preauthorize rights.
- Specify the `kAuthorizationFlagDestroyRights` mask to prevent the Security Server from preserving the rights obtained during this call.

`authorization`

A pointer to an authorization reference. On return, this parameter refers to the authorization session the Security Server creates. Pass `NULL` if you require a function result but no authorization reference.

Return Value

A result code. See "Authorization Services Result Codes" (page 26).

Discussion
The primary purpose of this function is to create the opaque authorization reference structure associated with the authorization reference. You use the authorization reference in other authorization functions.

You can use this function to authorize all or partial rights. Authorizing rights with this function is most useful for applications that require a one-time authorization. By passing `NULL` to the `authorization` parameter, the Security Server attempts to authorize the requested rights and returns the appropriate result code without actually granting the rights. If you are not going to call any other authorization functions, use this method to determine if a user has authorization without granting any rights.

You can also use this function to preauthorize rights by specifying the `kAuthorizationFlagPreAuthorize` mask. Preauthorization is most useful when a right has a zero timeout. For example, you can preauthorize in the application and if it succeeds, call the helper tool and request authorization. This eliminates calling the helper tool if the user cannot later authorize the specified rights.

If you do not specify the `kAuthorizationFlagPartialRights` mask and the Security Server denies at least one right, then the status of this function on return is `errAuthorizationDenied`.

If you do not specify the `kAuthorizationFlagInteractionAllowed` mask and the Security Server requires user interaction, then the status of this function on return is `errAuthorizationInteractionNotAllowed`.

If you specify the `kAuthorizationFlagInteractionAllowed` mask and the user cancels the authentication process, then the status of this function on return is `errAuthorizationCanceled`.

When your application no longer needs the authorization reference, use the function `AuthorizationFree` (page 13) to free the memory associated with it.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AuthForAll

BSDLLCTest

MoreIsBetter

QISA

Declared In
`Authorization.h`

AuthorizationCreateFromExternalForm

Internalizes the external representation of an authorization reference.

```
OSStatus AuthorizationCreateFromExternalForm (
   const AuthorizationExternalForm *extForm,
   AuthorizationRef *authorization
);
```

Parameters

extForm

A pointer to the external representation of the authorization reference you retrieve from the calling process.

authorization

A pointer to an authorization reference. On return, this points to the local copy of the authorization reference. The Security Server allocates the authorization reference for you, so you do not need to call the function `AuthorizationCreate` (page 10).

Return Value

A result code. See "Authorization Services Result Codes" (page 26).

Discussion

When passing an authorization reference between processes, use this function to internalize the external representation of the authorization reference you created using the function `AuthorizationMakeExternalForm` (page 14).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BSDLLCTest

MoreIsBetter

QISA

Declared In

`Authorization.h`

AuthorizationExecuteWithPrivileges

Runs an executable tool with root privileges.

```
OSStatus AuthorizationExecuteWithPrivileges (
   AuthorizationRef authorization,
   const char *pathToTool,
   AuthorizationFlags options,
   char *const *arguments,
   FILE **communicationsPipe
);
```

Parameters

authorization

An authorization reference referring to the authorization session.

pathToTool

The full POSIX pathname of the tool to execute.

options

Reserved options. Pass the `kAuthorizationFlagDefaults` constant.

arguments

An argv-style vector of strings to send to the tool.

communicationsPipe

A pointer to a file structure. The Security Server creates the file, opens it for reading and writing, and connects it to the tool's standard input and output. On return, you must close and dispose of this file using `fclose` when your communication is complete. Pass `NULL` if you do not need a communications channel.

Return Value

A result code. See "Authorization Services Result Codes" (page 26).

Discussion
This function enables you to execute the tool you specify in the `pathToTool` parameter as a separate, privileged process. The new process will run with root privileges regardless of the privileges of the invoking process. The new process can retrieve the authorization reference by calling the function `AuthorizationCopyPrivilegedReference` (page 7). The arguments you pass in the `arguments` parameter are relayed to the new process's `argv` parameter. A set of file descriptors is linked to the new process's standard input and output so that your process may communicate with the new process.

To check if the user is authorized to perform this operation, you should preauthorize the `kAuthorizationRightExecute` right. See `AuthorizationItem` (page 19) for a description of what information is included in the authorization item for this right.

Special Considerations

You should use this function only to allow installers to run as root and to allow a `setuid` tool to repair its `setuid` bit if lost. This function works only if the Security Server establishes proper authorization.

This function poses a security concern because it will indiscriminately run any tool or application, severely increasing the security risk. You should avoid the use of this function if possible. One alternative is to split your code into two parts—the application and a setuid tool. The application invokes the setuid tool using standard methods. The setuid tool can then perform the privileged operations. If the tool loses its setuid bit, use the `AuthorizationExecuteWithPrivileges` function to repair it. Factoring your program minimizes the use of this function and reduces the risk of harm. Read *Inside Mac OS X: Performing Privileged Operations With Authorization Services*.

Note that this function respects the setuid bit, if it is set. That is, if the tool you are executing has its setuid bit set and its owner set to foo, the tool will be executed with the user foo's privileges, not root privileges. To ensure that your call to the `AuthorizationExecuteWithPrivileges` function works as intended, make sure the setuid bit of the tool you wish to execute is cleared before calling `AuthorizationExecuteWithPrivileges` to execute the tool.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BSDLLCTest

MoreIsBetter

QISA

Declared In
`Authorization.h`

AuthorizationFree

Frees the memory associated with an authorization reference.

```
OSStatus AuthorizationFree (
   AuthorizationRef authorization,
   AuthorizationFlags flags
);
```

Parameters

`authorization`

  The authorization reference to free.

flags

A bit mask. In most cases, pass the constant `kAuthorizationFlagDefaults`. To remove all shared and nonshared authorizations, pass the constant `kAuthorizationFlagDestroyRights`.

Return Value

A result code. See "Authorization Services Result Codes" (page 26).

Discussion

Call this function when your application no longer needs the authorization reference you created using the function `AuthorizationCreate` (page 10).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BSDLLCTest

MoreIsBetter

QISA

Declared In

`Authorization.h`

AuthorizationFreeItemSet

Frees the memory associated with an authorization set.

```
OSStatus AuthorizationFreeItemSet (
   AuthorizationItemSet *set
);
```

Parameters

set

A pointer to the authorization set to free.

Return Value

A result code. See "Authorization Services Result Codes" (page 26).

Discussion

When your application no longer needs the authorization item sets created by the Security Server in the AuthorizationCopyRights and AuthorizationCopyInfo functions, you should call this function to free it.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Authorization.h`

AuthorizationMakeExternalForm

Creates an external representation of an authorization reference.

```
OSStatus AuthorizationMakeExternalForm (
   AuthorizationRef authorization,
   AuthorizationExternalForm *extForm
);
```

Parameters

`authorization`

An authorization reference referring to the authorization session.

`extForm`

A pointer to an external authorization reference. On return, this points to the external representation of the authorization reference.

Return Value

A result code. See "Authorization Services Result Codes" (page 26).

Discussion

This function creates an external representation of an authorization reference so that you can transmit it between processes. Authorizations are bound by session, process, and time limits, so you cannot store the authorization reference for another process to use. Instead, you must create an external representation of the authorization reference and pass it securely to the other process. Use the function `AuthorizationCreateFromExternalForm` (page 11) to internalize the external representation of the authorization reference.

If it is necessary for your application to perform some privileged operations, it is good programming practice to isolate all of the privileged operations in a separate process, referred to as a *helper tool* (see *Authorization Services Programming Guide* for details). In this case, you must pass your authorization reference to the helper tool so that Authorization Services can tell that the helper tool is operating on behalf of your application. Doing so allows the authorization dialog to show your application's path rather than the path to the helper tool and it allows the system to determine whether the authorization dialog should have keyboard focus.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BSDLLCTest

MoreIsBetter

QISA

Declared In

`Authorization.h`

AuthorizationRightGet

Retrieves a right definition as a dictionary.

```
OSStatus AuthorizationRightGet (
   const char *rightName,
   CFDictionaryRef *rightDefinition
);
```

Parameters

`rightName`

An ASCII character string representing the rightname. Wildcard right names are valid.

rightDefinition
A reference to a dictionary. On return, this points to a dictionary of keys that define the right. Passing `NULL` checks if the right is defined. You should release the memory used by the returned dictionary.

Return Value
A result code. See "Authorization Services Result Codes" (page 26).

Discussion
You do not need an authorization reference to use this function because the policy database is world readable.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
AuthForAll

Declared In
`AuthorizationDB.h`

AuthorizationRightRemove

Removes a right from the policy database.

```
OSStatus AuthorizationRightRemove (
   AuthorizationRef authRef,
   const char *rightName
);
```

Parameters

authRef
A valid authorization reference used to authorize modifications.

rightName
An ASCII character string representing the right name. This function does not accept wildcard right names.

Return Value
A result code. See "Authorization Services Result Codes" (page 26).

Discussion
The right you remove must be an explicit right with no wildcards. Wildcard rights are for use by system administrators for site configuration.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`AuthorizationDB.h`

AuthorizationRightSet

Creates or updates a right entry in the policy database.

```
OSStatus AuthorizationRightSet (
   AuthorizationRef authRef,
   const char *rightName,
   CFTypeRef rightDefinition,
   CFStringRef descriptionKey,
   CFBundleRef bundle,
   CFStringRef localeTableName
);
```

Parameters

authRef
: A valid authorization reference used to authorize modifications.

rightName
: An ASCII character string representing the right name. The policy database does not accept wildcard right names.

rightDefinition
: Either a CFDictionary containing keys defining the rules or a CFString representing the name of another right whose rules you wish to duplicate. See `Policy Database Constants` (page 24) for some possible values.

descriptionKey
: A CFString reference used as a key for looking up localized descriptions. If no localization is found, this is the description itself. This parameter is optional; pass `NULL` if you do not require it.

bundle
: A bundle to get localizations from if not the main bundle. This parameter is optional; pass `NULL` if you do not require it.

localeTableName
: A CFString representing a table name from which to get localizations. This parameter is optional; pass `NULL` if you have no localizations or you wish to use the localizations available in Localizable.strings.

Return Value
A result code. See "Authorization Services Result Codes" (page 26).

Discussion
The right you create must be an explicit right with no wildcards. Wildcard rights are for use by system administrators for site configuration.

You can use this function to create a new right or modify an existing right. For example,

```
AuthorizationRightSet(NULL, "com.ifoo.ifax.send",
CFSTR(kAuthorizationRuleIsAdmin), CFSTR("Authorize sending  of a fax"), NULL,
NULL);
```

adds a rule for letting administrators send faxes. This example creates a right named "`com.ifoo.ifax.send`" and sets the rules to require the user to be an administrator by using the `kAuthorizationRuleIsAdmin` constant. This example also sets a comment to let the system administrator know that the right authorizes administrators to send a fax.

To specify additional attributes for the right, you can pass an `CFDictionary` type in the *rightDefinition* parameter as shown in the following example.

```
CFStringRef keys[2] = {CFSTR(kRightRule), CFSTR(kRightComment)};
CFStringRef values[2] = {CFSTR(kAuthorizationRuleIsAdmin), CFSTR("authorizes
sending of 1 fax message")};
CFDictionaryRef aDict;
```

```
aDict = CFDictionaryCreate(NULL, (void *)keys, (void *)values, 2,
&kCFCopyStringDictionaryKeyCallBacks, &kCFTypeDictionaryValueCallBacks);
AuthorizationRightSet(NULL, "com.ifoo.ifax.send", aDict,  CFSTR("Authorize
sending  of a fax"), NULL, NULL);
CFRelease(aDict);
```

This call creates the same right as before, but adds a specific right comment to the rules definition.

When you specify comments, you should be specific about what you need to authorize. For example, the means of proof required for `kAuthorizationRuleAuthenticateAsAdmin` (a username and password) should not be included here since that rule might be configured differently.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
AuthForAll

Declared In
`AuthorizationDB.h`

Data Types

AuthorizationEnvironment

Represents a set of data about the environment, such as user name and other information gathered during evaluation of authorization.

```
typedef AuthorizationItemSet AuthorizationEnvironment;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Authorization.h`

AuthorizationExternalForm

The external representation of an authorization reference.

```
struct AuthorizationExternalForm {
    char bytes[kAuthorizationExternalFormLength];
};
```

Fields

`bytes`
An array of characters representing the external form of an authorization reference.

Discussion
Authorization references are bound by session, process, and time limits, so you cannot store the authorization references for another process to use. Use the functions `AuthorizationMakeExternalForm` (page 14) and `AuthorizationCreateFromExternalForm` (page 11) to externalize and internalize the authorization reference. Applications should take care not to disclose the external authorization reference to potential attackers since any process can use this external authorization reference to access the authorization reference.

AuthorizationFlags

Represents a bit mask of authorization options.

```
typedef UInt32 AuthorizationFlags;
```

Discussion
See "Authorization Options" (page 21) for a description of masks that you can use.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Authorization.h`

AuthorizationItem

Contains information about an authorization right or the authorization environment.

```
typedef struct {
    AuthorizationString name;
    UInt32 valueLength;
    void *value;
    UInt32 flags;
}AuthorizationItem;
```

Fields

`name`

The required name of the authorization right or environment data. The name of a right is something that you create. You should name rights in a style similar to Java package names. For example, `"com.myOrganization.myProduct.myRight"`. Set this field to `kAuthorizationRightExecute` when requesting a right for use in the function `AuthorizationExecuteWithPrivileges` (page 12).

See the `Security.framework/Headers/AuthorizationTags.h` header file for possible values for environment data.

`valueLength`

An unsigned 32-bit integer that represents the number of bytes in the `value` field. Set the `valueLength` field to `0` if you set the `value` field to `NULL`.

`value`

A pointer to information pertaining to the `name` field. For example, if the `name` field is set to the value represented by the constant `kAuthorizationRightExecute`, then set the `value` field to the full POSIX pathname of the tool you want to execute. In most other cases, set this field to `NULL`.

`flags`

Reserved option bits. Set to `0`.

Discussion

When using an authorization item to contain a right, set the `name` field to the name of the right—for example, `"com.myOrganization.myProduct.myRight"`, the `valueLength` and `flags` fields to `0`, and the `value` field to `NULL`. For more information on naming rights, read *Authorization Services Programming Guide*

When using an authorization item for the `AuthorizationExecuteWithPrivileges` function, set the `name` field to `kAuthorizationRightExecute`, and the `flags` field to `0`. Set the `value` field to the full POSIX pathname of the tool to execute and the `valueLength` field to the byte length of the value in the `value` field.

When using an authorization item to contain environment data, set the `name` field to the name of the environment data—for example, `kAuthorizationEnvironmentUsername`—and the `flags` field to `0`. Set the `value` field, in this case, to the actual user name and the `valueLength` field to the byte length of the value in the `value` field.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Authorization.h`

AuthorizationItemSet

Represents a set of authorization items.

```
typedef struct {
    UInt32 count;
    AuthorizationItem *items;
}AuthorizationItemSet;
```

Fields

`count`

The number of elements in the `items` array.

`items`

A pointer to an array of authorization items. If `count` is greater than `1`, `items` points to the first item in an array of such items. You should set this parameter to `NULL` if there are no items.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Authorization.h`

AuthorizationRef

Represents a pointer to an opaque authorization reference structure.

```
typedef const struct AuthorizationOpaqueRef* AuthorizationRef;
```

Discussion

This data type points to a structure the Security Server uses to store information about the authorization session. Use the functions described in "Authorization Services Functions" (page 6) to create, access, and free the authorization reference.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Authorization.h`

AuthorizationRights

Represents a set of authorization rights.

```
typedef AuthorizationItemSet AuthorizationRights;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Authorization.h`

AuthorizationString

Represents a zero-terminated string in UTF-8 encoding.

```
typedef const char* AuthorizationString;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Authorization.h`

Constants

Authorization Options

Define valid authorization options.

```
enum {
    kAuthorizationFlagDefaults = 0,
    kAuthorizationFlagInteractionAllowed = (1 << 0),
    kAuthorizationFlagExtendRights = (1 << 1),
    kAuthorizationFlagPartialRights = (1 << 2),
    kAuthorizationFlagDestroyRights = (1 << 3),
    kAuthorizationFlagPreAuthorize = (1 << 4),
    kAuthorizationFlagNoData = (1 << 20)
};
```

Constants

`kAuthorizationFlagDefaults`

If no bits are set, none of the following features are available.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

`kAuthorizationFlagInteractionAllowed`

If the bit specified by this mask is set, you permit the Security Server to interact with the user when necessary.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

`kAuthorizationFlagExtendRights`

If the bit specified by this mask is set, the Security Server attempts to grant the rights requested. Once the Security Server denies one right, it ignores the remaining requested rights.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

`kAuthorizationFlagPartialRights`

If the bit specified by this mask and the `kAuthorizationFlagExtendRights` mask are set, the Security Server grants or denies rights on an individual basis and all rights are checked.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

`kAuthorizationFlagDestroyRights`

If the bit specified by this mask is set, the Security Server revokes authorization from the process as well as from any other process that is sharing the authorization. If the bit specified by this mask is not set, the Security Server revokes authorization from the process but not from other processes that share the authorization.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

`kAuthorizationFlagPreAuthorize`

If the bit specified by this mask is set, the Security Server preauthorizes the rights requested.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

`kAuthorizationFlagNoData`

Private bits. Do not use.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

Discussion

The bits represented by these masks instruct the Security Server how to proceed with the function in which you pass them. Set all unused bits to `0` to allow for future expansion.

Authorization Rights Mask

Defines values the Security Server sets in an authorization item's `flag` field.

```
enum {
    kAuthorizationFlagCanNotPreAuthorize = (1 << 0)
};
```

Constants

`kAuthorizationFlagCanNotPreAuthorize`

Indicates the Security Server could not preauthorize the right.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

Discussion

Empty Environment

Defines an empty environment.

```
#define kAuthorizationEmptyEnvironment NULL
```

Constants

`kAuthorizationEmptyEnvironment`

Indicates an empty environment. You should pass this constant in functions with an environment parameter if you have no environment data to provide.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

External Authorization Reference Length

Defines the byte length of the external authorization reference.

```
enum {
    kAuthorizationExternalFormLength = 32
};
```

Constants

`kAuthorizationExternalFormLength`

Indicates, in number of bytes, the length of the array in the `AuthorizationExternalForm` (page 18) structure.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

Name Tags

Specify the type of environment data or right.

```
  #define kAuthorizationEnvironmentUsername "username"
#define kAuthorizationEnvironmentPassword "password"
#define kAuthorizationEnvironmentShared "shared"
#define kAuthorizationRightExecute "system.privilege.admin"
#define kAuthorizationEnvironmentPrompt  "prompt"
#define kAuthorizationEnvironmentIcon  "icon"
```

Constants

`kAuthorizationEnvironmentUsername`

Specifies a user name.

Available in Mac OS X v10.0 and later.

Declared in `AuthorizationTags.h`.

`kAuthorizationEnvironmentPassword`

Specifies a password.

Available in Mac OS X v10.0 and later.

Declared in `AuthorizationTags.h`.

`kAuthorizationEnvironmentShared`

Specifies a shared right.

Available in Mac OS X v10.0 and later.

Declared in `AuthorizationTags.h`.

`kAuthorizationRightExecute`

Specifies the name of the right associated with the function `AuthorizationExecuteWithPrivileges` (page 12).

Available in Mac OS X v10.0 and later.

Declared in `AuthorizationTags.h`.

`kAuthorizationEnvironmentPrompt`

Specifies the name of the authorization item that should be passed into the environment when specifying invocation-specific additional text. The value should be a localized UTF8 string.

Available in Mac OS X v10.3 and later.

Declared in `AuthorizationTags.h`.

`kAuthorizationEnvironmentIcon`

Specifies the name of the authorization item that should be passed into the environment when specifying an alternate icon. The value should be a full path to an image compatible with the `NSImage` class.

Available in Mac OS X v10.3 and later.

Declared in `AuthorizationTags.h`.

Discussion

These tags are possible values for the `name` field of an authorization item. This is not an all-inclusive set. You determine the name of the right to request. These environment tags are for future use.

Policy Database Constants

Defines constants for use in settting rights and rules in the policy database.

```
#define kAuthorizationRightRule "rule"
#define kAuthorizationRuleIsAdmin "is-admin"
#define kAuthorizationRuleAuthenticateAsAdmin "authenticate-admin"
#define kAuthorizationRuleAuthenticateAsSessionUser "authenticate-session-user"
#define kAuthorizationRuleClassAllow "allow"
#define kAuthorizationRuleClassDeny "deny"
#define kAuthorizationComment "comment"
```

Constants

`kAuthorizationRightRule`

Indicates a rule delegation key. Instead of specifying exact behavior, some rules are shipped with the system and may be used as delegate rules. Use this with any of the delegate rule definition constants.

Available in Mac OS X v10.3 and later.

Declared in `AuthorizationDB.h`.

`kAuthorizationRuleIsAdmin`

Indicates a delegate rule definition constant specifying that the user must be an administrator.

Available in Mac OS X v10.3 and later.

Declared in `AuthorizationDB.h`.

`kAuthorizationRuleAuthenticateAsAdmin`

Indicates a delegate rule definition constant specifying that the user must authenticate as an administrator.

Available in Mac OS X v10.3 and later.

Declared in `AuthorizationDB.h`.

`kAuthorizationRuleAuthenticateAsSessionUser`

Indicates a delegate rule definition constant specifying that the user must authenticate as the session owner (logged-in user).

Available in Mac OS X v10.3 and later.

Declared in `AuthorizationDB.h`.

`kAuthorizationRuleClassAllow`

Indicates a delegate rule definition constant that always allows the specified right.

Available in Mac OS X v10.3 and later.

Declared in `AuthorizationDB.h`.

`kAuthorizationRuleClassDeny`

Indicates a deleage rule definition constant that always denies the specified right.

Available in Mac OS X v10.3 and later.

Declared in `AuthorizationDB.h`.

`kAuthorizationComment`

Indicates comments for a rule. The comments appear in the policy database for the administrator to understand what the rule is for. Rule comments are not the same as localized descriptions which are presented to the user.

Available in Mac OS X v10.3 and later.

Declared in `AuthorizationDB.h`.

Discussion

You can use these constants when creating or modifying a rule in the policy database using the `AuthorizationRightSet` (page 16) function.

Result Codes

The most common result codes returned by the Security Server are listed in the table below.

Result Code	Value	Description
`errAuthorizationSuccess`	0	The operation completed successfully. Available in Mac OS X v10.0 and later.
`errAuthorizationInvalidSet`	-60001	The `set` parameter is invalid. Available in Mac OS X v10.0 and later.
`errAuthorizationInvalidRef`	-60002	The `authorization` parameter is invalid. Available in Mac OS X v10.0 and later.
`errAuthorizationInvalidTag`	-60003	The `tag` parameter is invalid. Available in Mac OS X v10.0 and later.
`errAuthorizationInvalidPointer`	-60004	The `authorizedRights` parameter is invalid. Available in Mac OS X v10.0 and later.
`errAuthorizationDenied`	-60005	The Security Server denied authorization for one or more requested rights. This error is also returned if there was no definition found in the policy database, or a definition could not be created. Available in Mac OS X v10.0 and later.
`errAuthorizationCanceled`	-60006	The user canceled the operation. Available in Mac OS X v10.0 and later.
`errAuthorizationInteractionNotAllowed`	-60007	The Security Server denied authorization because no user interaction is allowed. Available in Mac OS X v10.0 and later.
`errAuthorizationInternal`	-60008	An unrecognized internal error occurred. Available in Mac OS X v10.0 and later.
`errAuthorizationExternalizeNotAllowed`	-60009	The Security Server denied externalization of the authorization reference. Available in Mac OS X v10.0 and later.
`errAuthorizationInternalizeNotAllowed`	-60010	The Security Server denied internalization of the authorization reference. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`errAuthorizationInvalidFlags`	-60011	The `flags` parameter is invalid. Available in Mac OS X v10.0 and later.
`errAuthorizationToolExecuteFailure`	-60031	The tool failed to execute. Available in Mac OS X v10.0 and later.
`errAuthorizationToolEnvironmentError`	-60032	The attempt to execute the tool failed to return a success or an error code. Available in Mac OS X v10.0 and later.

Document Revision History

This table describes the changes to *Authorization Services C Reference*.

Date	Notes
2005-11-08	Updates and corrections for Mac OS X v10.4. This book was previously titled Authorization Services Reference.
	Revised description of environment parameter in `AuthorizationCreate` (page 10) and `AuthorizationCopyRights` (page 8) functions.
2004-09-30	Revised description of `AuthorizationCopyInfo` (page 6) function for Mac OS X v10.4.
2003-10-21	Added new constants to "Name Tags" (page 24).
	Added new constants to "Policy Database Constants" (page 24).
	Clarified use of setuid bit in "AuthorizationExecuteWithPrivileges" (page 12).
2003-06-01	Added documentation for AuthorizationDB.h.
2002-11-01	Updated the EDD and repaired tags for better searchability.
2002-06-01	First version of this document.

Index