Action Request System™ Programmer's Manual

2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

SunSoft Part No: 875-1777-10
Revision A, December 1995

Printed in the U.S.A.

Cover design by Carlick Advertising.

Part Number: PGU-210-001

Contents

Preface

Audience

This guide is intended for software developers who wish to use an application programming interface (API) to interact with the Action Request (AR) System. You should be familiar with UNIX and/or Windows NT, the C programming language, and the AR System. Experience with distributed applications is helpful.

Terminology Note: Be aware in this guide that "NT" is sometimes used as an abbreviation for the AR System Notification Tool. When this guide refers to the Windows NT operating system, it is described either as "the Windows NT operating system" or simply as "Windows NT."

Overview of this Document

Chapter 1, "Overview of the AR System API," provides a general overview of the operations that can be performed on AR System objects.

Chapter 2, "Using the AR System API," describes important libraries, data structures, conventions for memory and error management, and sample code.

Chapter 3, "Overview of the Notification Subsystem API," provides a general overview of the operations that can be performed in the Notification Tool subsystem.

Chapter 4, "Using the Notification Subsystem API," describes important libraries, include files, data structures and conventions for memory and error management, and information about initialization and termination.

Chapter 5, "AR System and Notification Subsystem Manual Pages," provides a complete set of manual pages for the AR System and Notification subsystem.

AR System Documents

The ***Action Request System Installation Guide*** describes how to install and license the AR System software. There are separate Installation Guides for the UNIX and Windows NT environments.

The ***Action Request System User's Guide*** is a how-to description of the operations performed by all users of the AR System. There are separate User's Guides for environments supporting the Motif, Macintosh, and Windows graphical user interfaces (GUIs) and for ASCII terminals.

The ***Action Request System Administrator's Guide*** describes how the AR System administrator can use the Administrator Tool to set up the AR System and define its local operations. This manual is also a reference of advanced AR System concepts. There are separate Administrator's Guides for the Motif and Windows environments.

The ***Action Request System Programmer's Guide*** (this document) is a reference guide for programming with the APIs that come with the AR System.

The ***Action Request System Troubleshooting and Error Messages Guide*** provides information to help you identify and solve problems with the AR System.

The ***Action Request System Getting Started Guide and Sample Schemas*** provides an online demonstration showing the use of the AR System in a sample help desk environment and describes how you can use the sample schemas provided with the AR System.

The ***Action Request System Distributed Server Option Administrator's Guide*** provides information about operating the AR System in a distributed, UNIX multi-server environment. Included are instructions for creating all of the necessary mappings and filters that you use along with the Distributed Server Option to keep AR System entries synchronized across multiple servers.

The ***ARWeb Administrator's Guide*** provides details about installing, using, and customizing the ARWeb application, so that you can provide access to your company's AR System applications through the World-Wide Web.

The ***Action Request System Help Desk Template Guide*** describes the Helpdesk application that runs in conjunction with the AR System to help you manage your internal help desk organization. The template takes full advantage of the rich feature set of the AR System and implements workflow and reporting mechanisms to simplify the task of working in or managing a help desk.

Conventions Used in this Manual

bold font

Indicates that a word is a new or important term.

Example: **filters**.

Initial Caps

Button and menu names and items have the first letter capitalized.

Example: File.

`computer font`

Indicates computer output, including UNIX prompts, an explicit directory, or a file name.

Example: `prompt%.`

Indicates data to be entered by the user.

Example: `aruser &.`

<small italic font>

Indicates a variable directory, file name, or string that you replace with an appropriate directory, file name, or string.

Example: *<ar_config_dir>*.

italics

Indicates a reference to another manual or to a different section within the current manual.

Example: see AR System Documents

Italic type is also used for emphasis.

Example: *All* users will be affected.

Overview of the AR System API 1

This chapter provides an overview of the Action Request (AR) System application programming interface (API) functions. The API functions can be used to perform operations on the following AR System objects:

- Entries.
- Schemas.
- Fields.
- Menus.
- Filters.
- Escalations.
- Active links.
- Administrator extensions. (For your reference, administrator extensions are referred to as administrator *commands* in the AR System Administrator Tool for OSF/Motif.)

For example, you can create entries in the database for a specified schema. Or, if you have administrator access to the AR System you can create or delete schemas from a specified server. (For detailed information on access control, refer to the *Action Request System Administrator's Guide*.)

1

Note – For information on API functions that are used exclusively to interact with the Notification Tool (NT) subsystem, see Chapter 3, “Overview of the Notification Subsystem API,” and Chapter 4, “Using the Notification Subsystem API.”

There are five main operations that can be performed for each AR System object:

- Create an object.
- Delete an object.
- Retrieve information (get) about an object.
- Modify (set) an object.
- Retrieve a list of objects (get list).

There are also miscellaneous operators for importing and exporting definitions, verifying users, and initializing and terminating interaction with the AR System.

The remainder of this chapter briefly describes each of the available API functions. The sections are organized by the type of object. For detailed information on each operation, see the appropriate manual page in Chapter 5, “AR System and Notification Subsystem Manual Pages.”

Entry Operations

ARCreateEntry

Adds a new entry to the database for a specified schema. You can specify any number of fields and associated values. The system checks permissions for each field and reports errors if a field does not exist, or if you do not have access.

If any one of the fields is in error, the entire create operation is rejected and no change is made to the database.

ARDeleteEntry

Deletes an entry from the database for a specified schema. You must have administrator access to perform this function.

ARGetEntry

Retrieves information about the entry, as indicated by the `entryId` in the specified schema. You can request the values for a specific list of fields, for all fields that are accessible, or for no fields (to verify the existence of an entry). If you do not have read access to the specified field(s), then no data is returned.

ARSetEntry

Updates information about the entry, as indicated by the `entryId` in the specified schema. You can specify any number of fields and associated values. The system will check permissions for each field and report errors if a field does not exist, or if you do not have access.

If any one of the fields is in error, the entire set operation is rejected and no change is made to the entry.

ARGetListEntry

Performs a high-performance database search. The system retrieves the `entryId` and a short description of all the entries in the schema that meet the specified conditions and are accessible by you. The system returns only entries to which you have read access.

ARMergeEntry

Merges an existing entry from another source into the database for a specified schema. You can specify any number of fields and associated values. The system checks permissions for each field and reports errors if a field does not exist or if you do not have access. You are expected to supply information for some of the system fields as they were set in the previous environment.

If any one of the fields is in error, the entire operation is rejected and no change is made to the database.

ARGetEntryStatistics

Performs a high-performance statistical calculation. The system returns a statistic computed on all entries that meet the specified condition and are accessible to you.

Schema Operations

ARCreateSchema

Creates a new schema on the specified server. The schema created will contain the core fields. You must have administrator access to perform this function.

ARDeleteSchema

Deletes an existing schema from the specified server. The operation deletes the schema, its associated fields, filters, escalations, and active links. In addition, the delete operation removes any entries for the schema from the database. You must have administrator access to perform this function.

ARGetSchema

Retrieves information about a schema on the specified server. The information returned is global schema information (for instance, when the schema was last modified, or indexes on the schema), not information on specific fields within the schema. Only schemas that you have access to are returned.

You must have administrator access to retrieve information about permissions.

ARSetSchema

Updates information about a schema on the specified server. The information that is updated is global schema information (for instance, changing the schema name), not information about individual fields within the schema. You must have administrator access to perform this function.

ARGetListSchema

Retrieves a list of all the schemas on a specified server. You can specify a timestamp that limits the schemas returned to those modified after the designated time. Only schemas to which you have access are returned.

Field Operations

ARCreateField

Creates a new field in a specified schema. You must have administrator access to perform this function.

ARDeleteField

Deletes an existing field from the specified schema. In addition, this operation deletes (from the database) all of the data contained in the field. You must have administrator access to perform this function.

ARGetField

Retrieves information about a specified field in a particular schema. You must have administrator access to retrieve information about permissions.

ARSetField

Updates information for a specified field in a particular schema (for instance, whether a field is required or optional). You must have administrator access to perform this function.

ARGetListField

Retrieves a list of all the fields for a specified schema. You can specify a timestamp that limits the fields returned to those modified after the designated time.

1

Menu Operations

ARCreateCharMenu

Creates a new menu on the specified server. The menu can be a hierarchical list of items, a reference to a file, or a query to a schema. You must have administrator access to perform this function.

ARDeleteCharMenu

Deletes an existing menu from the specified server. You must have administrator access to perform this function.

ARGetCharMenu

Retrieves the definition of a menu on the specified server.

ARSetCharMenu

Updates information about a menu on the specified server. You must have administrator access to perform this function.

ARGetListCharMenu

Retrieves a list of all the menus on a specified server. You can specify a timestamp that limits the fields returned to those modified after the designated time.

ARExpandCharMenu

Expands the definition of the character menu to a list-style menu. Query references are resolved and file contents are loaded as defined by the menu structure.

Filter Operations

You must have administrator access to perform any of the following filter operations.

ARCreateFilter

Creates a new filter on the specified server. The filter is immediately activated, and remains activated until modified or deleted.

ARDeleteFilter

Deletes an existing filter from the specified server. The deleted filter is immediately removed and all processing associated with it is no longer performed.

ARGetFilter

Retrieves information about a filter on the specified server.

ARSetFilter

Updates an existing filter on the specified server. The updates are immediately activated and remain activated until modified or deleted.

ARGetListFilter

Retrieves a list of all the filters on a specified server. You can specify a timestamp that limits the filters returned to those modified after the designated time. Or, you can specify a particular schema to limit the filters returned to those that are linked to that schema.

Escalation Operations

You must have administrator access to perform any of the following escalation operations.

ARCreateEscalation

Creates a new escalation on the specified server. The escalation is immediately activated, and remains activated until modified or deleted.

ARDeleteEscalation

Deletes an existing escalation from the specified server. The deleted escalation is immediately removed and all processing associated with it is no longer performed.

ARGetEscalation

Retrieves information about a escalation on the specified server.

ARSetEscalation

Updates an existing escalation on the specified server. The updates are immediately activated and remain activated until modified or deleted.

ARGetListEscalation

Retrieves a list of all the escalations on a specified server. You can specify a timestamp that limits the escalations returned to those modified after the designated time. Or, you can specify a particular schema to limit the escalations returned to those that are linked to that schema.

Active Link Operations

ARCreateActiveLink

Creates an active link on the specified server. The active link is activated on a particular client the next time the client connects to the schema. You must have administrator access to create an active link.

ARDeleteActiveLink

Deletes an existing active link from the specified server. The deleted link will be removed from each client the next time a new copy of the schema is accessed by the client. You must have administrator access to delete an active link.

ARGetActiveLink

Retrieves information about an active link on the specified server. You must have access to an active link to retrieve it. You must have administrator access to retrieve information about permissions.

ARSetActiveLink

Updates an existing active link on the specified server. Each client will be informed of the updates the next time the client accesses the schema from the server. You must have administrator access to update an active link.

ARGetListActiveLink

Retrieves a list of all active links on a specified server. You can specify a timestamp that limits the active links returned to those modified after the designated time. Or, you can specify a particular schema to limit the active links returned to those that are linked to that schema. Only active links that are accessible to you are returned.

Administrator Extension Operations

Note – Administrator extensions are referred to as administrator *commands* in the AR System Administrator Tool for OSF/Motif and apply to the UNIX server only.

ARCreateAdminExtension

Creates a new administrator extension on the specified server. You must have administrator access to create an administrator extension.

ARDeleteAdminExtension

Deletes existing administrator extensions from the specified server. You must have administrator access to delete an administrator extension.

ARGetAdminExtension

Retrieves information about an administrator extension on the specified server. You must have access to an administrator extension to retrieve it. You must have administrator access to retrieve information about permissions.

ARSetAdminExtension

Updates information about an administrator extension on the specified server. You must have administrator access to update information about an administrator extension.

ARGetListAdminExtension

Retrieves a list of all the administrator extensions on a specified server. You can specify a timestamp that limits the extensions returned to those modified after the designated time. Only administrator extensions that are accessible to you are returned.

ARExecuteAdminExtension

Executes the process defined by an administrator extension on the specified server. You must have access to the administrator extension to execute it.

Miscellaneous Operations

ARInitialization

Initializes the program's interaction with the AR System. For some systems, this call performs no work, while for other systems it establishes an initial state for the system. You should always call this routine in case it is needed by the environment.

ARTermination

Terminates the program's interaction with the AR System. For all systems, it deregisters the user from being a current user on the system (otherwise they remain registered until the timeout interval). For some systems, this call

performs no additional work, while for other systems it performs some cleanup operations for the system. You should always call this routine in case it is needed by the environment.

ARVerifyUser

Checks whether or not the user is registered on the AR System. You use this routine to check the user against those registered on the specified server.

ARGetListServer

Retrieves a list of all the servers that are accessible from the current machine. It retrieves the list of servers by processing the AR directory file `/etc/ar`, retrieving all registered AR System servers.

ARExport

Exports one or more structure definitions from the AR System. Exportable structures are: schemas (including mail templates), menus, filters, escalations, active links, and administrator extensions. This allows the duplication of definitions from one server to another. You must have administrator access to export filters, escalations, administrator extensions, and full-detailed schema and active link definitions. Otherwise, if you have access (but not administrator access), you can export schema, menu, and active link definitions without permissions information.

ARGetListSQL

Retrieves a list of results for a specified SQL command on a specified server. It will retrieve values appropriate to the command issued and return rows in the database that match.

ARGetListGroup

Retrieves a list of all the groups on a specified server. You must have administrator access to perform this function. ARGetListGroup also allows retrieval of group information for a specific user. If for yourself, anyone can do it. If for someone else, must be Administrator.

ARGetListUser

Allows retrieval of a list of all registered or current users and information about their licenses (Admin only). Also allows retrieval of information about yourself (anyone).

ARImport

Imports one or more structure definitions (schemas, menus, filters, escalations, active links, and administrator extensions) to the AR System. This allows the duplication of definitions from one server to another. You must have administrator access to perform this function.

ARGetServerInfo

Retrieves information about the server. This information includes the version, the type of database, license information, OS and hardware environment, and configuration settings for the AR System server.

ARSetServerInfo

Updates information about the server environment. This information includes only the configuration settings subset of the information available in the `ARGetServerInfo` call.

ARGetFullTextInfo (UNIX only)

Retrieves information about Full Text configuration on the server. This includes its state, location, and other configuration settings.

ARSetFullTextInfo (UNIX only)

Allows setting Full Text configuration, such as state, location, and other settings.

ARGetServerStatistics

Allows retrieval of statistical information about the operation and performance of the AR System server.

ARDecodeStatusHistory

Takes a formatted status-history string returned from the server and decodes it into an array of name and time values.

ARDecodeDiary

Takes a formatted diary string returned from the server and decodes it into an array of name, time, and text values.

ARLoadARQualifierStruct

Takes a string containing a qualification and returns an `ARQualifierStruct`. It parses the string and builds an appropriate structure, allocating new space and building all necessary levels of the data structure.

1

Using the AR System API 2

This chapter provides a brief introduction to the AR System API and how it is used. The following subjects are covered:

- Libraries.
- Include files.
- Data structures.
- Special handling for field/keyword substitution.
- Routines to free allocated memory.
- Error handling.
- Notes about using Initialization, Termination, and GetList.
- Sample source — an API driver.

The AR System API provides a complete interface to the server. All AR clients, including clients that have been developed by Remedy Corporation, work exclusively through this API.

As described in Chapter 1, “Overview of the AR System API,” the API functions are organized into groups by the type of object they are working with (schemas, fields, etc.); each group supports five basic operations (Create, Delete, Get, Set, and GetList). All of the functions follow the same general guidelines for interaction. This chapter explains in detail how the API is organized, and explains the issues you must consider when using the API.

At the end of the chapter, there is a brief discussion of an API driver. The source for that driver is provided in the directory <*ar_install_dir*>/api/src/driver (for UNIX) and <*ar_install_dir*>\api\driver (for Windows NT). It provides examples of how to load all of the data structures and how to call each of the API routines in the product. There are a number of routines in this sample code (especially the print functions) that may be useful during your application development.

Libraries

There is one library associated with the AR System API:

`arapi.lib`	For Windows NT, the AR System library containing all the AR System API functions. This library is located in the <*ar_install_dir*>\api\lib directory.
`libar.a`	For UNIX, the AR System library containing all the AR System API functions. This library is located in the <*ar_install_dir*>/api/lib directory.

For Sun SPARCstations running SunOS the AR System library, `libar.a`, is built using the System V compiler (`xpg2`) using the System V message catalog facility. Using the catalog routines allows you to internationalize all messages returned by the system. If you do not have the System V compiler, you must link in the additional library, `libarcat.a`. This will provide some missing libraries to provide System V message catalog support.

Include Files

There are five include files associated with the AR System API. The following list contains a description of each file, as well as information on when the file should be included in your program:

`ar.h`	The main include file for the AR System. It contains all of the basic data structure definitions as well as all of the definitions for size limits and AR System constants. This file must be included whenever any AR System routine or structure is referenced in the file.

`arerrno.h`	The list of error codes for the AR System. It contains a definition for each of the error codes that can be generated by the AR System API or server. If you will be checking for specific error codes, you should include this file (always use the definition and not the error number itself in case the error number changes). If you are simply reporting returned errors or are not processing errors in a given file, you do not need this file.
`arextern.h`	The external declarations for all of the API functions. It contains an external declaration for each of the AR System API functions. The definitions are specified both with and without parameter prototypes so they can be used by a standard C, an ANSI C, or a C++ compiler. If you call any AR System API function in the file, it is recommended that you include this file. Although it is not required for the standard C compiler, both the ANSI C and C++ compilers require function prototypes.
`arfree.h`	The external declarations for all the AR System API free functions. It contains an external declaration for each of the AR System API functions that frees a data structure. The definitions are specified both with and without parameter prototypes so they can be used by a standard C, an ANSI C, or a C++ compiler. If you call any AR System API free function in the file, it is a good idea to include this file. Although it is not required for the standard C compiler, both the ANSI C and C++ compilers require function prototypes.
`arstruct.h`	The data and file structure definitions for the AR System. It contains definitions for core and other reserved field IDs, the set of special separator characters used within the database to format data entries, and the set of labels used whenever structure definitions are exported to files.

For UNIX, these include files are located in the <*ar_install_dir*>`/api/include` directory. For Windows NT, they are located in the <*ar_install_dir*>`\api\include` directory.

Data Structures

There are a number of data structures in the AR System, most of which are relatively straightforward. However, there are a few that are either a bit more involved or central to the use of the API. These data structures are described in detail in this section.

Note – Many of the data structures involve allocated memory. See the discussions that follow for information on freeing data structures and on who is responsible for freeing the data. It is important that you free space when it is no longer needed.

Before getting into any specific structures, a general note about lists is in order. There are many different places in the system where you are dealing with lists of names, IDs, or structures. In general, lists are handled as arrays in the system, not as linked lists. The basic list structure for a list of type XXX is defined as follows:

```
typedef struct {
       unsigned intnumItems;
       ARXXXStruct *ARXXX  List;
} ARXXXList;
```

The numItems field indicates the number of items that are on the list. This number can be 0, in which case the ARXXXList field is not used (ARXXXList is *generally* set to NULL but it does not have to be since numItems of 0 causes it to be ignored). If there is a single item, the ARXXXList field points to allocated space holding a single item of type ARXXXStruct. If there is more than one item, the ARXXXList field points to the start of an array of numItems ARXXXStruct items. A single block of memory is allocated for the set of ARXXXStruct items (not one block per item). If the ARXXXStruct items themselves point to allocated memory, the nested memory is allocated as needed separately from the array.

The following sections describe nine of the most important or complex structures. These include:

- ARControlStruct and ARStatusList, which are used in almost every call.
- ARValueStruct, ARFieldValueList, and ARQualifierStruct, which are used to interact with entries.
- ARFieldLimitStruct, which is used to manage fields.
- ARCharMenuStruct, which is used to manage menus.
- ARAssignStruct and ARFieldAssignList, which are used when dealing with filters, escalations, and active links.

ARControlStruct

```
typedef struct {
              long         cacheId;
              ARTimestamp  operationTime;
              ARNameType   user;
              ARNameType   password;
              char         language[AR_MAX_LANG_SIZE + 1];
              char         server[AR_MAX_SERVER_SIZE + 1];
} ARControlStruct;
```

The `ARControlStruct` structure is the first parameter in most of the AR System API functions. It contains information about you and the current environment in which you are operating. You must load information about yourself into this structure and transfer it with each call to the system. Since the API does not establish and maintain connections across the network, this information is necessary to verify permissions.

The fields of the `ARControlStruct` structure include:

`cacheId`	An ID that provides a key to the server about where information for the user is stored. The server creates and maintains this cache area. You should initialize the ID to 0 before the first call. It is important that you do not change this value or the AR System server will have to reload information about you for each call instead of using a local cache of information.
`operationTime`	A timestamp that indicates the date and time the operation occurred on the server. This value is assigned by the server on each API call.
`user`	Your user name in the AR System. This identifies you to the server so your permissions are retrieved and verified by the server.
`password`	The password for the specified user name.
`language`	An ASCII string containing the name of the language (using one of the language strings supported by the `setlocale` routine). To use the default language, specify either the empty string ("") or **C**. Error messages will be returned in the language specified (assuming there is a message catalog for that language).
`server`	The name of the server to contact for this operation. In both single- and multi-server environments, the server name is stored. Each call can be to a different server although it is likely that you will choose one server and perform most or all of your operations on that server. You can use the `ARGetListServer` routine to get a list of accessible servers.

ARStatusList

```
typedef struct {
             unsigned int      messageType;
             int               messageNum;
             char              *messageText;
} ARStatusStruct;

typedef struct {
             unsigned int      numItems;
             ARStatusStruct    *statusList;
} ARStatusList;
```

The `ARStatusList` structure is the last parameter of every AR System API function. It provides error and warning information about the operation that was performed. The structure itself is straightforward. It is simply a list (as described above) of `ARStatusStruct` items. The `ARStatusStruct` structure contains three fields:

`messageType`	A code for the type of message: `AR_RETURN_OK`: All is OK, just an informational note. `AR_RETURN_WARNING`: The operation completed successfully, but there is some condition that you may be interested in. `AR_RETURN_ERROR`: The operation failed. No action was performed.
`messageNum`	The numeric value for the message (can use the constants in `arerrno.h` to search for specific errors).
`messageText`	ASCII text message for the error. This message is up to `AR_MAX_MESSAGE_SIZE` bytes long and in the language specified by the language field of the `ARControlStruct`.

For more information on this structure, see the section “Error Handling, later in this chapter.

ARValueStruct

```
typedef struct {
        unsigned int  dataType;
        union {
                unsigned int  keyNum;
                long          intVal;
                double        realVal;
                char          *charVal;
                char          *diaryVal;
                unsigned long enumVal;
                ARTimestamp   timeVal;
                unsigned long maskVal;
        } u;
} ARValueStruct;
```

The ARValueStruct structure contains two pieces of information: the data type of the value and the value itself. Each data type has a specific format for the value it expects and a corresponding field:

AR_DATA_TYPE_NULL	A NULL (that is, no) value. This is not really a data type, as a value for any data type can be NULL. Note that a NULL value is *not* the same as an empty string or a 0 for a numeric value. **Note:** If you use an API call (such as ARGetEntry) on a field that returns null values, then the data type in the value structure changes to AR_DATA_TYPE_NULL. If you are concerned whether or not a value is null, you need to check the data type instead of the value itself.
AR_DATA_TYPE_KEYWORD	A numeric index (between 0 and AR_MAX_KEYWORD_USED as defined in the include file ar.h) indicating which keyword. This data type is used during assignment of values and to hold limits or default values. It will not be present in values returned from the database.
AR_DATA_TYPE_INTEGER	A 32-bit integer value.
AR_DATA_TYPE_REAL	A 64-bit floating-point value.
AR_DATA_TYPE_CHAR	A null-terminated character string. The space for the character string is allocated (and so must be freed). Specifying a data type of AR_DATA_TYPE_CHAR with a NULL pointer is equivalent to specifying a data type of AR_DATA_TYPE_NULL.

AR_DATA_TYPE_DIARY	A null-terminated character string. The space for the character string is allocated (and so must be freed). Specifying a data type of AR_DATA_TYPE_DIARY with a NULL pointer is equivalent to specifying a data type of AR_DATA_TYPE_NULL. **Note:** On input to a create or set operation, the diary data is simply the *new* diary text to be added to the value. Do not include the existing diary text or it will be added to the diary again. On output and on input to a merge operation, the string contains the entire contents of the diary value (including user and timestamps). Use the API call, ARDecodeDiary, to decode the string if needed.
AR_DATA_TYPE_ENUM	A numeric-enumerated (selection) value. An enumerated value is a 0-indexed value indicating the position of the choice in the list of enumerated values. If you want to interact with names instead of numbers, it is your responsibility to translate the names into numbers (and numbers into names) when interacting with this structure (see ARFieldLimitStruct).
AR_DATA_TYPE_TIME	A time and date stamp using a UNIX-style time (number of seconds since Jan. 1, 1970). You can use standard system functions to translate to and from a more readable format.
AR_DATA_TYPE_BITMASK	Reserved for future use.

ARFieldValueList

```
typedef struct {
              ARInternalId       fieldId;
              ARValueStruct      value;
} ARFieldValueStruct;

typedef struct {
              unsigned int       numItems;
              ARFieldValueStruct *fieldValueList;
} ARFieldValueList;
```

The ARFieldValueList structure is the main structure used in the Entry functions to retrieve and set values for the individual fields of an entry. Accordingly, you will be using this structure anytime you deal with entries in

the system. The structure is a list (see previous discussion) of ARFieldValueStruct items. Each ARFieldValueStruct item is structured as follows:

fieldId	The internal ID of a field. All interaction with the server regarding fields uses the field ID instead of the field name. This is because the field ID is a fixed number that is unique regardless of how users have customized the labels on their local view, and regardless of the language they are speaking.
value	An ARValueStruct item that holds the value of the associated field. Remember, if the field has or is to be assigned no value, use the AR_DATA_TYPE_NULL data type; otherwise, the data type of the value and of the field definition must be compatible (must match or be assigned to a type-compatible keyword).

ARQualifierStruct

Following is the *ARQualifierStruct* example.

```
typedef struct {
             unsigned long       enumVal;
             unsigned int        userOrTime;
} ARStatHistoryValue;

typedef struct {
             unsigned int                    tag;
          union {
                 ARInternalId             fieldId;
                 ARValueStruct            value;
                 struct ARArithOpStruct   *arithOp;
                 ARStatHistoryValue       statHistory;
          } u;
} ARFieldValueOrArithStruct;

typedef struct ARArithOpStruct {
          unsigned int                operation;
           ARFieldValueOrArithStruct   operandLeft;
           ARFieldValueOrArithStruct   operandRight;
} ARArithOpStruct;

typedef struct {
          unsigned int                operation;
          ARFieldValueOrArithStruct   operandLeft;
          ARFieldValueOrArithStruct   operandRight;
} ARRelOpStruct;

typedef struct ARQualifierStruct {
             unsigned int    operation;
             union {
                        struct {
                                struct ARQualifierStruct
   *operandLeft;
                                struct ARQualifierStruct
*operandRight;
                       } andor;
                         struct ARQualifierStruct        *not;
                         ARRelOpStruct
                   *relOp;
             } u;
} ARQualifierStruct;
```

Note – An API call, ARLoadARQualifierStruct, exists that takes a qualification string and converts it into the appropriate set of these structures. For details, see ARLoadARQualifierStruct in Chapter 5.

The ARQualifierStruct structure is one of the more involved structures in the system. It is used to specify qualification criteria to the ARGetListEntry and ARGetEntryStatistics calls and for the qualification in all the filter, escalation, and active link calls. It is important to be able to specify a complex sequence of conditions in a reasonable structure. The following discussion will detail how this structure is organized, including a discussion of the nested structures used to build a full qualification.

To start, you need to think about what a qualification is. It is a set of zero or more conditions that specify limits on which entries should be retrieved. The ARQualifierStruct structure (along with all its nested support structures) is used to define a qualification.

The ARQualifierStruct structure itself consists of an operation to be performed and then one or more pointers to structures that contain information for that operation. If there is no qualification, you can either specify a NULL for the entire structure *or* you can specify an operation of AR_COND_OP_NONE. If there is some qualification, you need to specify one of the other operations. AR_COND_OP_AND, AR_COND_OP_OR, and AR_COND_OP_NOT are used to indicate the three basic logical operations. The AND and OR operations require specification of a left and right operand while the NOT operation requires a single operand. Each of these operands is simply another ARQualifierStruct. AR_COND_OP_REL_OP indicates that the operation is not one of the logicals but a simple relational operation. The value associated with a relational operation is stored in the ARRelOpStruct structure.

The ARRelOpStruct structure defines a relational operation. The following six basic relational operators are supported:

- AR_REL_OP_EQUAL
- AR_REL_OP_GREATER
- AR_REL_OP_GREATER_EQUAL
- AR_REL_OP_LESS
- AR_REL_OP_LESS_EQUAL
- AR_REL_OP_NOT_EQUAL

A pattern-matching operator, AR_REL_OP_LIKE, is also supported. (See the discussion of the LIKE operator, Full Text Search capability, and the use of wild cards in the *Action Request System User's Guide*).

For each of these operations, the qualification has a left and right operand. The ARFieldValueOrArithStruct structure is used to define the value to be used in the operation. There are four major types of values:

AR_FIELD	Identify a field whose value is to be retrieved for the qualification.
AR_VALUE	Specify a constant value that is to be used in the qualification.
AR_ARITHMETIC	Use the ARArithOpStruct structure to define an arithmetic operation (see below) to be performed during the qualification.
AR_STAT_HISTORY	Identify a specific status history field whose value is to be retrieved for the qualification. A status history field is identified by supplying the enumerated value for the status of the desired field and a tag indicating whether the user or time information of that status is desired (AR_STAT_HISTORY_USER or AR_STAT_HISTORY_TIME).

Note – There are several additional branches to this structure that have been defined and not described here. They are reserved for future use and are not implemented at this time.

The ARArithOpStruct structure defines an arithmetic operation. The operations supported are the binary operations AR_ARITH_OP_ADD, AR_ARITH_OP_SUBTRACT, AR_ARITH_OP_MULTIPLY, AR_ARITH_OP_DIVIDE, and AR_ARITH_OP_MODULO along with the unary operation AR_ARITH_OP_NEGATE. For the binary operations, you must define the left and right operands using the ARFieldValueOrArithStruct structure. For the unary operation, ignore the operandLeft field and supply a single operand using operandRight.

ARFieldLimitStruct

```
typedef struct {
          long           rangeLow;
              long           rangeHigh;
} ARIntegerLimitsStruct;

typedef struct {
             double         rangeLow;
             double         rangeHigh;
             int            precision;
} ARRealLimitsStruct;

typedef struct {
             unsigned int   maxLength;
             unsigned int   menuStyle;
             unsigned int   qbeMatchOperation;
             ARNameType     charMenu;
             char           *pattern;
          unsigned int   FullTextOptions;
} ARCharLimitsStruct;

typedef struct {
          unsigned int   FullTextOptions;
} ARDiaryLimitsStruct;

typedef struct {
             unsigned int                dataType;
             union {
                       ARIntegerLimitsStruct intLimits;
                       ARRealLimitsStruct    realLimits;
                       ARCharLimitsStruct    charLimits;
                    ARDiaryLimitsStruct   diarylimits;
                       ARNameList            enumLimits;
          } u;
} ARFieldLimitStruct;
```

The ARFieldLimitStruct structure is used to define the limits on fields. During the create operation, you can specify no limits by supplying a NULL pointer for the limit parameter. During the set operation, a NULL pointer means the value is unchanged. So you will need to specify the structure with the dataType field set to AR_FIELD_LIMIT_NONE to reset to no limit (this can be used during the create operation too).

As noted, a data type of AR_FIELD_LIMIT_NONE is used to indicate that there are no limits defined for a field. If limits are defined, the dataType field will be set to the data type of the field. Note that the field will never be set for time field since there are no field limits that can be specified for this type. Each of the other types can have limits defined as follows:

AR_DATA_TYPE_INTEGER	Using the ARIntegerLimitsStruct structure, you can define a lower- and upper-range limit for the value.
AR_DATA_TYPE_REAL	Using the ARRealLimitsStruct structure, you can define a lower- and upper-range limit for the value. In addition, you can specify a precision setting to use when a value is displayed. This setting will limit the number of decimal places that are displayed for the value.
AR_DATA_TYPE_CHAR	Using the ARCharLimitsStruct structure, you can define a maximum length for the value. Specifying a maximum length of 0 indicates an unlimited length. See the discussion of the Length field property in Chapter 4 of the *Action Request System Administrator's Guide* for notes about the efficiency of long character string storage. In addition, you can specify whether a character menu is associated with the field, whether dropping a value from a menu (if attached) overwrites existing data or appends to it, the type of qualification to use by default for a QBE (query-by-example) query, a pattern that all values must match and a flag to indicate whether the field is Full Text indexed. Simply leave the charMenu or pattern fields blank to specify no menu or no pattern.
AR_DATA_TYPE_DIARY	Using the ARDiaryLimitStruct structure, you can define whether the field is Full Text indexed.
AR_DATA_TYPE_ENUM	Using the ARNameList structure, you define the list of one or more names that form the enumerated set for the field. You *must* specify a limit for an enumerated field since it is undefined without the set of enumerated values.

ARCharMenuList

```
typedef struct {
            ARNameType                      menuLabel;
            unsigned int                    menuType;
            union {
                      char                     *menuValue;
                      struct ARCharMenuStruct *childMenu;
            } u;
} ARCharMenuItemStruct;

typedef struct {
            ARNameType          schema;
            char                server[AR_MAX_SERVER_SIZE + 1];
            ARQualifierStruct   qualifier;
            ARInternalId        labelField;
            ARInternalId        valueField;
            ARBoolean           sortOnLabel;
} ARCharMenuQueryStruct;

typedef struct {
            unsigned int   fileLocation;
            char           *filename;
} ARCharMenuFileStruct;

typedef struct {
            unsigned int          numItems;
            ARCharMenuItemStruct  *charMenuList;
} ARCharMenuList;

typedef struct ARCharMenuStruct {
            unsigned int                    menuType;
            union {
                   ARCharMenuList           menuList;
                   ARCharMenuQueryStruct    menuQuery;
                   ARCharMenuFileStruct     menuFile;
                   ARCharMenuSQLStruct      menuSQL;
        } u;
} ARCharMenuStruct;
```

The ARCharMenuStruct structure is used when dealing with menus. A menu definition can be one of four types:

AR_CHAR_MENU_LIST	Labels and values are defined as part of the menu definition. A hierarchical menu definition is put into place as an integral part of the definition.
AR_CHAR_MENU_QUERY	A query to the same or another AR System schema on the same or a different server is used to build a menu dynamically. The definition includes which field to use as the label and which as the value.
AR_CHAR_MENU_FILE	A file contains the definition of the menu. The file can contain a flat or hierarchical file definition.
AR_CHAR_MENU_SQL	An SQL command on the same or a different server is used to build a menu dynamically. The definition includes which column to use as the label and which column as the value.

The ARCharMenuItemStruct structure holds information for a single item in a list type menu. It contains the following fields:

menuLabel	The label for the menu item. This is the visible label and not the text associated with the item.
menuType	The type of item at this position: AR_MENU_TYPE_VALUE: This item is a leaf in the menu and has an associated value. AR_MENU_TYPE_MENU: This item has a submenu so no value is associated with it.
menuValue	If AR_MENU_TYPE_VALUE, a pointer to the actual value that is associated with this menu item. It may or may not be the same as the label.
childMenu	If AR_MENU_TYPE_MENU, a pointer to an ARCharMenuStruct structure that contains the menu definition for the submenu.

ARAssignStruct

```
typedef struct {
            char                        server[AR_MAX_SERVER_SIZE
+ 1];
            ARNameType                  schema;
            ARQualifierStruct           qualifier;
            unsigned int                tag;
            union {
                        ARInternalId        fieldId;
                   ARStatHistoryValue  statHistory;
        } u;
} ARAssignFieldStruct;

typedef struct {
        char                         *serviceName;
        char                         *topic;
        char                         *item;
        unsigned int                 action;
            char                         *pathToProgram;
            char                         *command;
} ARDDEStruct;

typedef struct {
        char             server[AR_MAX_SERVER_SIZE + 1];
        char             *sqlCommand;
        unsigned int     valueIndex;
        unsigned int     noMatchOption;
        unsigned int     multiMatchOption;
} ARAssignSQLStruct;

typedef struct {
            unsigned int                            assignType;
            union {
                        ARValueStruct                       value;
                   ARAssignFieldStruct            *field;
                   char                           *process;
                   struct ARArithOpAssignStruct   *arithOp;
                   ARDDEStruct                    *dde;
                         ARAssignSQLStruct               *sql;
        } u;
} ARAssignStruct;

typedef struct ARArithOpAssignStruct {
            unsigned int     operation;
            ARAssignStruct   operandLeft;
            ARAssignStruct   operandRight;
} ARArithOpAssignStruct;

typedef struct ARFunctionAssignStruct {
            unsigned int     functionCode;
            unsigned int     numItems;
            ARAssignStruct   *parameterList;
} ARFunctionAssignStruct;
```

The ARAssignStruct structure contains two pieces of information: a tag detailing the type of assignment operation to perform and a structure with details for the assignment. The types of assignment and the details needed for each are as follows:

AR_ASSIGN_TYPE_VALUE	A simple constant value using the ARValueStruct structure. The value can be a constant value of the same type as the target field or a keyword value with a compatible type.
AR_ASSIGN_TYPE_FIELD	A cross-reference assignment to a value that is held in a field in the current or another entry in the AR System. The entry can be in the same or a different schema. You use the ARAssignFieldStruct structure to define the entry being referenced (refer to the ARAssignFieldStruct discussion on the following page).
AR_ASSIGN_TYPE_PROCESS	A string to be transferred to the operating system and executed in a shell. The value returned to stdout by the process executed will be assigned to the target field if the return code of the process is 0. If the return is not 0, it indicates an error return and the data in stdout is treated as error text and displayed to the user. (Not available for active links run from Windows or Macintosh.)
AR_ASSIGN_TYPE_ARITH	An arithmetic operation between several values is to be performed and the result assigned to the target. The operation is defined using the ARArithOpAssignStruct structure.
AR_ASSIGN_TYPE_FUNCTION	A function to be performed on one or more parameters and the result assigned to the target. The function is defined using the ARFunctionAssignStruct structure.
AR_ASSIGN_TYPE_DDE	A DDE request operation to be performed against another operation and the result assigned to the target. The operation is defined using the ARDDEStruct structure. (Only available for active links run from Windows.)
AR_ASSIGN_TYPE_SQL	A cross-reference assignment to a value that is returned using a SQL command. You use the ARAssignSQLStruct structure to define the data being referenced (refer to the ARAssignSQLStruct discussion).

The ARAssignFieldStruct structure is used to reference a field whose value is to be retrieved and assigned to the target field. It contains information about the server and schema that contains the desired field and an optional qualifier that selects an entry from that schema. Finally, it contains a tag indicating whether the desired value is a field value or a status history value and which field/status history is desired. To indicate that the value should be extracted from the currently displayed window and not from the database, specify '*' as the value for the schema and server.

Note – For filters and escalations, if no values match the qualification, a NULL value is assigned to the field; if more than one value is returned, the first return value is used. For active links, if no values match, an error is returned; if more than one value is returned, a selection list appears to allow the use to select the item desired.

The ARAssignSQLStruct structure is used to reference a database entry whose value is to be retrieved and assigned to the target field. It contains an SQL statement used to retrieve the database entry and an index of which value to use. It also specifies the action to take if there are no matches or more than one match.

The ARArithOpAssignStruct structure defines an arithmetic operation (much like the ARArithOpStruct defined above). You can specify any combination of operations using any of the types of value assignment defined as long as the resulting value is compatible with the target value.

The following operations are supported binary operations:

- AR_ARITH_OP_ADD
- AR_ARITH_OP_SUBTRACT
- AR_ARITH_OP_MULTIPLY
- AR_ARITH_OP_DIVIDE
- AR_ARITH_OP_MODULO

For these operations, you must define the left and right operands using the ARAssignStruct structure.

For the unary operation, AR_ARITH_OP_NEGATE, ignore the operandLeft field and supply a single operand using operandRight.

The ARFunctionOpAssignStruct structure defines a function to be performed and specifies the list of parameters to use while performing the function. There are over 20 functions defined. You can specify any arbitrary set of assign structure arguments for the parameters. You must define a number of parameters appropriate for the call being made. Automatic data type conversion will be performed for each parameter as necessary.

ARFieldAssignList

```
typedef struct {
           ARInternalId         fieldId;
           ARAssignStruct       assignment;
} ARFieldAssignStruct;

typedef struct {
            unsigned int        numItems;
            ARFieldAssignStruct *fieldAssignList;
} ARFieldAssignList;
```

The ARFieldAssignList structure is the main structure used in filter, escalation, and active link functions to set values for individual fields within a set fields action. The structure is a list (see previous description) of ARFieldAssignStruct items. Each ARFieldAssignStruct item is structured as follows:

fieldId	The internal ID of a field. All interaction with the server regarding fields uses the field ID instead of the field name. This is because the field ID is a fixed number that is unique regardless of how users have customized the labels on their local view, and regardless of the language they are speaking.
value	An ARAssignStruct item that holds the value of the associated field. Remember, if the field is specified, it must be assigned a value using one of the possible assignment types.

Special Handling for Field/Keyword Substitution

There are a number of cases where fields and/or keywords can be specified in a value with the intention of expanding the value with the contents of that field or keyword when the value is used. Some examples of these values include: notify text in the filter and escalation notify action, the message text in filter and active link message actions, the run process string in filter, escalation, and active link run process actions, and a macro parameter value in an active link macro action.

To handle these situations, place a parameter that represents the field or keyword directly into the character string. The string will be pre-processed with the parameter expanded. The syntax for field and keyword parameters are:

field id	`$id$`	where `id` is the field ID (for example, `$1$`).
keyword	`$-keynum$`	where `keynum` is the index of the keyword as defined in `ar.h` (for example, `$-3$`). Note the negative sign before the number. This is necessary to indicate a keyword value as distinguished from a field ID. For `$NULL$`, use the string `$--1$` (two minus signs followed by the number one).

When used through the `aruser` and `aradmin` programs, you are allowed to use the field and keyword names. This is because these programs translate the internal parameter format to and from names for easier viewing and manipulation by the user. However, since the names of the fields and of the keywords (in case of localization) can be different for different users, the parameters must key off of the IDs.

Routines to Free Allocated Memory

The API includes a set of functions to make it easy to free memory for all the AR System structures that allocate memory (see `arfree.h`). This includes any structures you may create and any that are returned from the AR System API calls. You simply pass a pointer to the structure to the appropriate free routine and it will recursively free all allocated memory within that structure.

Each of the free functions has two parameters. The first is a pointer to the structure whose memory is to be freed. The second is a boolean flag. If it is set to `TRUE`, you are stating that the memory used by the base structure is also allocated and should be freed with any nested memory. If it is set to `FALSE`, you are stating that the top level structure is *not* allocated and should *not* be freed

with the rest of the memory. If not freed, the fields of the top-level structure are initialized to an empty state so a second free on the structure will perform no action.

Note that all of the free functions are coded to handle “empty” structures. They will all no-op if a `NULL` pointer is specified for the structure. They will also work if the structure is a list structure containing 0 entries. You can safely call the free functions with any structure that has been initialized to an empty state regardless of whether any data has subsequently been assigned.

Responsibility for Freeing Allocated Memory

Both the input and output parameters to the AR System API functions involve allocated memory. It is important to remember to free that space to avoid a buildup of old allocated memory that is no longer used.

In simple terms, you (the caller of the API functions) are responsible for freeing *all* memory for both input and output parameters. The API function cannot tell if any of the input parameters are allocated (you could have statically allocated space on the stack and used pointers to that space) and cannot tell how long you want to retain any returned data. So, you have the responsibility of freeing any space when it is no longer needed.

If you receive a return from a function that is `AR_RETURN_ERROR`, `AR_RETURN_FATAL`, or `AR_RETURN_BAD_STATUS`, the API functions guarantee that the only structure that may have allocated memory is the `ARStatusList` parameter. All other output parameters will be initialized to an empty state. You can call the free functions on those empty parameters if you desire or if it is easier given how you have structured your code, but it is not required in an error case.

Error Handling

As in any system, the management of errors reported by the AR System API functions is important. Because all errors are reported in a common structure, `ARStatusList`, error processing can be handled in a generic routine for all the AR System API functions.

First, every API function has an `int` return, which indicates whether or not the operation was successful. The return is one of the following values:

AR_RETURN_OK	The operation was successfully performed. The status list may contain an informational note (in a few cases).
AR_RETURN_WARNING	The operation was successfully performed; however, something happened during the operation that the system wants you to be aware of. There will be one or more warning messages (and maybe some notes) in the status list.
AR_RETURN_ERROR	The operation failed and was *not* performed. There will be one or more error messages (and maybe some warnings and notes) in the status list.
AR_RETURN_FATAL	The operation failed and was *not* performed. In addition, an error was encountered while trying to prepare the status list with the error information. There *may* be one or more error messages (and maybe some warnings and notes) in the status list. Note that the status list may not be complete and may even be empty if the error is such that the entries cannot be added to the status list.
AR_RETURN_BAD_STATUS	The operation was *not* performed. The status parameter is bad so the operation was cancelled.

You can check the return value of the call to determine whether the operation occurred. It is safe to ignore any informational notes at any time. You can also ignore warnings, but it is better if you at least report them. You should never ignore errors since they resulted in the failure of the operation.

The errors in the status list are sorted so that errors are first, warnings next, and notes last. In addition, the most recent message of each category is first, with older messages in chronological order after that. So, the most recent and most serious messages are always listed first with the return of the function reflecting the severity of the first message in the list.

Remember that the ARStatusList structure uses allocated space for the message list and for the text of any message. It is important to call the FreeARStatusList function after any processing of the status list for every API call (even if they return AR_RETURN_OK since notes can be returned even if all is okay). You do not need to worry about whether there are any messages before calling the free routine since it will perform no action if there are no messages to free.

2

ARInitialization and ARTermination

The AR System API is designed to be able to run in a wide variety of environments. Each environment has a different set of operations that must be performed to set up and shut down the RPC and/or network environment for use by an application. Accordingly, the `ARInitialization` and `ARTermination` functions help deal with varying environments.

These functions perform the operations needed to establish the correct environment for the rest of the API functions. You should use `ARInitialization` as the first AR System API call in your program and `ARTermination` as the last. You can call `ARInitialization` at the beginning of the program and `ARTermination` at the end to make it easier to ensure the calls are made first and last, respectively.

In an environment where floating licenses are used, the `ARTermination` call is even more important. This call is used to disconnect from the server and to free the floating license token held by the user. If this call is not made, the token will not be freed and will be held until the defined timeout interval even though the user is no longer accessing the system.

Using the GetList Functions

The API has been structured around the idea of using the `ARxxxGetList` functions to get a list of 0 or more items matching a set of criteria, then using the IDs/names returned to call the `ARGetxxx`, `ARSetxxx`, and/or `ARDeletexxx` function to perform some operation on the items in the list. The `GetList` functions are all designed to be efficient at retrieving the values requested to provide quick response to your list requests.

The `ARGetListEntry` function uses the `ARQualifierStruct` structure (discussed previously) to specify conditions that all entries you are interested in must meet. The other `GetList` operations allow qualification of entries using a timestamp. That timestamp specifies that only items changed since the time indicated are returned. A timestamp of 0 is used to indicate all items. Generally, you will use a timestamp of 0 for an initial list and then the timestamp saved from the last retrieval (the time from the `ARControlStruct`) for future calls to just get a list of the items that have been modified.

Sample Source — An API Driver

The AR System provides a sample program that implements a prompt-driven interface to the API routines. The program contains a call to every API routine, illustrating how the input structures can be loaded and the output structures traversed. It is located in <*ar_install_dir*>`/api/src/driver` (in UNIX) and <*ar_install_dir*>`\api\driver` (in Windows NT).

The source is intended as example code to help you understand how to work with the AR System API. There are a number of routines that may be helpful to you while developing an application. A good example of this is the `print.c` file which contains a set of print routines for each of the data structures in the AR System. Using these routines, you can print the contents of structures before and after API calls to ensure they contain the data you expect.

This source can be compiled into a program called `driver` which provides an interactive, prompt-driven interface to the API calls. You can use the interface to see how various combinations of parameters would work and to get a feel for whether you are calling the routines correctly. When you run the driver, you should be aware of the following:

- The `ARControlStruct` structure is loaded with the **log** command and is loaded globally, not with each API call. You can change it if desired but it is not necessary to reset it for each call.
- Use the **h** command (for help) to display the menu of options again.

Other than this, you simply choose an API call to make and the driver will prompt you for all the parameters, make the call, and display the results.

Building an API Program in the Windows NT Environment

You can create an application that makes API calls to your AR System servers. To build your own applications, you need to link the API library. You can use the sample `driver.mak` file, found in the <*ar_install_dir*>`\api\driver` directory, as a template to create your own make file.

After you create a make file, run `nmake` from the command line, as in the following example:

```
nmake /f driver.mak
```

2

Overview of the Notification Subsystem API

3

The AR System supplies a set of API functions that allows access to the full functionality of the Notification Tool subsystem.

Note – Terminology: Be aware in this chapter that “NT” is sometimes used as an abbreviation for the AR System Notification Tool. When this guide specifically refers to the Windows NT operating system, it is described either as “the Windows NT operating system” or simply as “Windows NT.”

The Notification System API is divided into two sets of calls, NTxxxClient and NTxxxServer, which represent the division of the Notification subsystem into two processes:

- The *Notification client* (`ntclientd`) receives notifications and distributes them locally to processes that have registered interest. To receive notifications, you interact with this set of calls.
- The *Notification server* (`ntserverd`) accepts requests for notifications to be sent and delivers them to registered users. To send notifications, you interact with this set of calls.

The remainder of this chapter briefly describes each available API function. For more information on each operation, see the appropriate manual page in Chapter 5.

3

Notification Client Operations

This section contains a complete list of the Notification client operations, and information on the calls you can use to receive notifications.

Several of the operations are used to communicate between the `ntclientd` and `ntserverd` processes. Normally, you would not need to use these calls; however, it is possible to call them directly if you are replacing `ntclientd` with your own process. These calls are noted as follows: (*server -> client*)

NTCheckRegisteredClient

Checks with the Notification client to see if the specified user is registered. *(server -> client)*

NTDeregisterClient

Closes registration for the specified process with the Notification client.

NTInitializationClient

Initializes the program for interaction with the Notification client. For some systems, this call performs no work, while in others it establishes an initial state for the system. You should always call `NTInitializationClient` in case it is needed by the environment.

NTNotificationClient

Delivers a notification to the indicated client. *(server -> client)*

NTRegisterClient

Registers the process with the Notification client.

NTTerminationClient

Terminates the program's interaction with the Notification client. For some systems, this call performs no work, while in others it performs some cleanup operations for the system. You should always call `NTTerminationClient` in case it is needed by the environment.

To register to receive messages, use the following calls

`NTInitializationClient`	Set up the program for communications with the Notification client.
`NTRegisterClient`	Register with the Notification client, validate and register the user with each Notification server on the network. Read messages from pipe used for communications (specified in call).
`NTDeregisterClient`	Close registration for this Notification client.
`NTTerminationClient`	Perform cleanup operations and terminate interaction with the Notification client.

Notification Server Operations

This section contains a complete list of the Notification server operations and information on the calls you can use to send messages.

Several of the operations are used in communications between the `ntclientd` and `ntserverd` processes. Normally, you would not need to use these calls; however, it is possible for you to call them directly if you will be replacing `ntserverd` with your own process. These calls are noted as follows: *(client -> server*)

NTDeregisterServer

Closes registration for the specified process with the Notification server. *(client -> server)*

NTGetListServer

Retrieves a list of all the servers that are accessible from the current machine.

NTInitializationServer

Initializes the program for interaction with the Notification server. For some systems, this call performs no work, while in others it establishes an initial state for the system. You should always call this routine in case it is needed by the environment.

NTNotificationServer

Delivers a notification to the indicated server.

NTRegisterServer

Registers the process with the Notification server. *(client -> server)*

NTTerminationServer

Terminates the program's interaction with the Notification server. For some systems, this call performs no work, while in others it performs some cleanup operations for the system. You should always call this routine in case it is needed by the environment.

To send messages, use the following calls

`NTInitializationServer`	Set up the program for communications with the Notification server.
`NTNotificationServer`	Deliver message. One call for each message sent. The server will process the message to deliver it to the Notification client processes registered with the Notification server.
`NTTerminationServer`	Perform cleanup operations and terminate interaction with the Notification server.

Using the Notification Subsystem API 4

This chapter describes the structure of the Notification Tool subsystem API functions, and how to use the API functions to interact with the Notification client and server.

Note – Terminology Be aware in this chapter that “NT” is sometimes used as an abbreviation for the AR System Notification Tool. When this guide refers to the Windows NT operating system, it is described either as “the Windows NT operating system” or simply as “Windows NT.”

The topics covered in this chapter include:

- Libraries.
- Include files.
- Data structures.
- Routines to free allocated memory.
- Error handling.
- Notes about using Initialization and Termination.

The Notification System API provides a complete interface to the Notification client and server. All clients of the Notification subsystem, including clients that have been developed by Remedy Corporation, work exclusively through this API.

4

Libraries

There are two libraries associated with the Notification System API. Each of the libraries is described below for the UNIX and Windows NT platforms respectively.

For UNIX

The Notification System API libraries for UNIX are located in the <*ar_install_dir*>/api/lib directory:

libntc.a	The Notification client library containing all the Notification client API functions.
libnts.a	The Notification server library containing all the Notification server API functions.

For Sun SPARCstations running SunOS the AR System library, libar.a, is built using the System V compiler (xpg2) using the System V message catalog facility. Using the catalog routines allows you to internationalize all messages returned by the system. If you do not have the System V compiler, you must link in the additional library, libarcat.a. This will provide some missing libraries to provide System V message catalog support.

For Windows NT

The Notification System API libraries for Windows NT are located in the <*ar_install_dir*>\api\lib directory:

ntc.lib	The Notification client library containing all the Notification client API functions.
nts.lib	The Notification server library containing all the Notification server API functions.

Include Files

There are six include files associated with the Notification System API. The following list contains a description of each file, as well as information on when the file should be included in your program:

`nt.h` — The main include file for the Notification System subsystem. It contains all of the basic data structure definitions as well as all of the definitions for size limits and Notification constants.
This file must be included whenever any Notification routine or structure is referenced in the file.

`ntcextrn.h` — The external declarations for all of the Notification client API functions. It contains an external declaration for each of the Notification client API functions. The definitions are specified both with and without parameter prototypes so they can be used by a standard C, an ANSI C, and a C++ compiler.
If you call any Notification client API function in the file, it is a good idea to include this file. Although it is not required for the standard C compiler, both the ANSI C and C++ compilers require function prototypes.

`nterrno.h` — The list of error codes for the Notification System subsystem. It contains a definition for each of the error codes that can be generated by the Notification System subsystem.
If you will be checking for specific error codes, you should include this file (always use the define and not the error number itself in case the error number changes). If you are simply reporting returned errors or are not processing errors in a given file, you do not need this file.

`ntfree.h` — The external declarations for all the Notification System API free functions. It contains an external declaration for each of the Notification API functions that frees memory used by Notification data structures. The definitions are specified both with and without parameter prototypes so they can be used by a standard C, an ANSI C, and a C++ compiler.
If you call any Notification API free function in the file, it is a good idea to include this file. Although it is not required for the standard C compiler, both the ANSI C and C++ compilers require function prototypes.

`ntsextrn.h`	The external declarations for all of the Notification server API functions. It contains an external declaration for each of the Notification server API functions. The definitions are specified both with and without parameter prototypes so they can be used by a standard C, an ANSI C, and a C++ compiler. If you call any Notification server API function in the file, it is a good idea to include this file. Although it is not required for the standard C compiler, both the ANSI C and C++ compilers require function prototypes.
`ntcextrn.h`	The external declarations for all of the Notification client API functions. It contains an external declaration for each of the Notification client API functions. The definitions are specified both with and without parameter prototypes so they can be used by a standard C, an ANSI C, and a C++ compiler. If you call any Notification client API function in the file, it is a good idea to include this file. Although it is not required for the standard C compiler, both the ANSI C and C++ compilers require function prototypes.

For UNIX, these include files are located in the <*ar_install_dir*>`/api/include` directory. For Windows NT, they are located in the <*ar_install_dir*>`\api\include` directory.

Data Structures

There are few data structures in the Notification subsystem. Only one significant structure is described in detail in this section — `NTStatusList`.

Note – Some of the data structures involve allocated memory. See the following discussions on freeing data structures and on who is responsible for freeing the data. It is important that you free space when it is no longer needed.

Before getting into any specific structures, a general note about lists is in order. There are many different places in the system where you are dealing with lists of names, IDs, or structures. In general, lists are handled as arrays in the system, not as linked lists. The basic list structure for a list of type XXX is defined as follows:

```
typedef struct {
unsigned intnumItems;
NTXXXStruct    *NTXXXList;
} NTXXXList;
```

The numItems field indicates the number of items that are on the list. This number can be 0 in which case the NTXXXList field is not used (NTXXXList is *generally* set to NULL but it does not have to be since numItems of 0 causes it to be ignored). If there is a single item, the NTXXXList field points to allocated space holding a single item of type NTXXXStruct. If there is more than one item, the NTXXXList field points to the start of an array of numItems NTXXXStruct items. A single block of memory is allocated for the set of NTXXXStruct items (not one block per item). If the NTXXXStruct items themselves point to allocated memory, the nested memory is allocated as needed separately from the array.

NTStatusList

```
typedef struct {
             unsigned int      messageType;
             int               messageNum;
             char              *messageText;
} NTStatusStruct;

typedef struct {
             unsigned int      numItems;
          NTStatusStruct    *statusList;
} NTStatusList;
```

The NTStatusList structure is the last parameter of every Notification System API function. It provides error and warning information about the operation that was performed. The structure itself is straightforward. It is simply a list (as described) of NTStatusStruct items. The NTStatusStruct structure contains three fields:

messageType	A code for the type of message: NT_RETURN_OK: All is OK, just an informational note. NT_RETURN_WARNING: The operation completed successfully, but there is some condition that you may be interested in. NT_RETURN_ERROR: The operation failed. No action was performed.
messageNum	The numeric value for the message (can use the constants in nterrno.h to search for specific errors).
messageText	ASCII text message for the error. This message is up to NT_MAX_MESSAGE_SIZE bytes long.

For more information on this structure, see the section, “Error Handling, later in this chapter.

Routines to Free Allocated Memory

The API includes a set of functions to make it easy to free memory for all the Notification structures that allocate memory (see ntfree.h). This includes any structures you may create and any that are returned from the Notification System API calls. You simply pass a pointer to the structure to the appropriate free routine and it will recursively free all allocated memory within that structure.

Each of the free functions has two parameters. The first is a pointer to the structure whose memory is to be freed. The second is a Boolean flag. If it is set to TRUE, you are stating that the memory used by the base structure is also allocated and should be freed with any nested memory. If it is set to FALSE, you are stating that the top level structure is *not* allocated and should *not* be freed with the rest of the memory. If not freed, the fields of the top-level structure are initialized to an empty state so a second free on the structure will perform no action.

Note that all of the free functions are coded to handle “empty” structures. They will all no-op if a `NULL` pointer is specified for the structure. They will also work if the structure is a list structure containing 0 entries. You can safely call the free functions with any structure that has been initialized to an empty state regardless of whether any data has subsequently been assigned.

Responsibility for Freeing Allocated Memory

Both the input and output parameters to the Notification System API functions involve allocated memory. It is important to remember to free that space to avoid a buildup of allocated memory that is no longer used.

In simple terms, you (the caller of the API functions) are responsible for freeing *all* memory for both input and output parameters. The API function cannot tell if any of the input parameters are allocated (you could have statically allocated space on the stack and used pointers to that space) and cannot tell how long you want to retain any returned data. So, you have the responsibility of freeing any space when it is no longer needed.

If you receive a return from a function that is `NT_RETURN_ERROR`, `NT_RETURN_FATAL`, or `NT_RETURN_BAD_STATUS`, the API functions guarantee that the only structure that may have allocated memory is the `NTStatusList` parameter. All other output parameters will be initialized to an empty state. You can call the free functions on those empty parameters if you desire or if it is easier given how you have structured your code, but it is not required in an error case.

Error Handling

As in any system, the management of errors reported by the Notification System API functions is important. Because all errors are reported in a common structure, `NTStatusList`, error processing can be handled in a generic routine for all the Notification System API functions.

First, every API function has an int return, which indicates whether or not the operation was successful. The return is one of the following values:

`NT_RETURN_OK`	The operation was successfully performed. The status list may contain an informational note (in a few cases).

NT_RETURN_WARNING	The operation was successfully performed; however, something happened during the operation that the system wants you to be aware of. There will be one or more warning messages (and maybe some notes) in the status list.
NT_RETURN_ERROR	The operation failed and was *not* performed. There will be one or more error messages (and maybe some warnings and notes) in the status list.
NT_RETURN_FATAL	The operation failed and was *not* performed. In addition, an error was encountered while trying to prepare the status list with the error information. There *may* be one or more error messages (and maybe some warnings and notes) in the status list. Note that the status list may not be complete and may even be empty if the error is such that the entries cannot be added to the status list.
NT_RETURN_BAD_STATUS	The operation was *not* performed. The status parameter is bad so the operation was cancelled.

You can check the return value of the call to determine whether the operation occurred. It is safe to ignore any informational notes at any time. You can also ignore warnings, but it is better if you at least report them. You should never ignore errors since they resulted in the failure of the operation.

The errors in the status list are sorted so that errors are first, warnings next, and notes last. In addition, the most recent message of each category is first, with older messages in chronological order after that. So, the most recent and most serious messages are always listed first with the return of the function reflecting the severity of the first message in the list.

Remember that the NTStatusList structure uses allocated space for the message list and for the text of any message. It is important to call the FreeNTStatusList function after any processing of the status list for every API call (even if they return NT_RETURN_OK since notes can be returned even if all is okay). You do not need to worry about whether there are any messages before calling the free routine since it will perform no action if there are no messages to free.

NTInitialization Client, NTTermination Client, NTInitialization Server, and NTTermination Server

The Notification System API is designed to be able to run in a wide variety of environments. Each environment has a different set of operations that must be performed to set up and shut down the RPC and/or network environment for use by an application.

Accordingly, the `NTInitializationClient/NTInitializationServer` and `NTTerminationClient/NTTerminationServer` functions help deal with varying environments.

These functions perform the operations needed to establish the correct environment for the rest of the API functions. You should call `NTInitializationClient/NTInitializationServer` as the first Notification System API call in your program and `NTTerminationClient/NTTerminationServer` as the last.

You can call `NTInitializationClient/NTInitializationServer` at the beginning of the program and `NTTerminationClient/NTTerminationServer` at the end to make it easier to ensure the calls are made first and last, respectively.

4

AR System and Notification Subsystem Manual Pages 5

This Chapter contains the UNIX-style manual pages for the programming interfaces.

Note – Functions which apply only to the UNIX or Windows NT environments are noted in their respective sections.

The chapter is organized first by manual section:

- User Commands (beginning on page 56).
- C Library Functions (beginning on page 87).
- File Formats (beginning on page 231).

and then alphabetically within each section.

5

User Commands

addsnm

NAME

addsnm – add new User Commands to one or more SunNet Manager schema files

SYNOPSIS

addsnm

DESCRIPTION

addsnm is a program that helps you to add User Commands to the SunNet Manager schema files. By default, it will connect to the elements.schema file. You can change to any SunNet Manager schema file you want.

Whatever schema file you choose, including the default, you must have write access to the file you will be updating. If you do not have write access, you will receive an error when you try and save your changes to the file.

You can load a single User Command onto a single glyph class, a single command onto multiple glyph classes, or multiple commands onto a single or multiple glyph classes.

This process only updates the schema files. It is your responsibility to restart SunNet Manager (using the -i option) to recompile using the new schema files.

SEE ALSO

snm (1)

aradmin

NAME

aradmin – interactive Administrator interface to AR System

SYNOPSIS

aradmin [-a | -e | -f | -m | -s | -t] [-x server]

DESCRIPTION

aradmin is the Administrator interface to the AR System. It allows you to manage the AR System server. Only an AR System Administrator user can use this tool. You can create and manage schemas, filters, active links, menus, escalations, and administrator extensions.

When the program runs, it stores cached and personalized configuration information in a directory on the local machine. By default, that directory is **$HOME/arHome** where $HOME is your home directory. If you want the local information stored in another location, you must set the **ARHOME** environment variable to the directory you want to use. You must have write access to the directory you choose.

OPTIONS

The following options to **aradmin** may appear in any order on the command line:

-a Place the system into active link mode at startup.

-e Place the system into administrator extensions mode at startup.

-f Place the system into filter mode at startup.

-m Place the system into character menu mode at startup.

-s Place the system into schema mode at startup.

-t Place the system into escalation mode at startup.

-x Specify the name of a server to connect to. This option may be included more than once to connect to multiple servers.

By default, the list of servers the tool will connect to is defined in the directory file **/etc/ar**. If the **-x** option is not specified, this file will be used to determine where to connect. If this option is specified one or more times, the servers specified in these options will be used.

ENVIRONMENT

ARDATE Defines the date format to be used by the program. By default, the date format is determined by the format associated with the language identified by the LANG environment variable. If the default format is not sufficient, you can set this variable. The value is a string of operators as defined by the strftime(3) library call. NOTE: There are some combinations of formats that will display successfully, but cannot be translated for input.

ARHOME Defines the location of the local home environment for the user. It is the directory where the config file is stored and where the default arcmd directory is located. If not specified, this directory defaults to **$HOME/arHome**.

ARPATH Defines a search path of directories that may contain macro definitions. The value is a set of one or more directories separated by colons. The directory $ARHOME/arcmds is always assumed to be on the path so all directories defined here are in addition to this directory.

FILES

/etc/ar

$ARHOME/config

SEE ALSO

arserverd (1), **aruser** (1), **notifier** (1)

5

aradmin

NAME

aradmin – interactive Administrator interface to AR System

SYNOPSIS

aradmin [-a | -e | -f | -m | -s | -t] [-x server]

DESCRIPTION

aradmin is the Administrator interface to the AR System. It allows you to manage the AR System server. Only an AR System Administrator user can use this tool. You can create and manage schemas, filters, active links, menus, escalations, and administrator extensions.

When the program runs, it stores cached and personalized configuration information in a directory on the local machine. By default, that directory is **$HOME/arHome** where $HOME is your home directory. If you want the local information stored in another location, you must set the **ARHOME** environment variable to the directory you want to use. You must have write access to the directory you choose.

OPTIONS

The following options to **aradmin** may appear in any order on the command line:

-a Place the system into active link mode at startup.

-e Place the system into administrator extensions mode at startup.

-f Place the system into filter mode at startup.

-m Place the system into character menu mode at startup.

-s Place the system into schema mode at startup.

-t Place the system into escalation mode at startup.

-x Specify the name of a server to connect to. This option may be included more than once to connect to multiple servers.

By default, the list of servers the tool will connect to is defined in the directory file **/etc/ar**. If the **-x** option is not specified, this file will be used to determine where to connect. If this option is specified one or more times, the servers specified in these options will be used.

ENVIRONMENT

ARDATE Defines the date format to be used by the program. By default, the date format is determined by the format associated with the language identified by the LANG environment variable. If the default format is not sufficient, you can set this variable. The value is a string of operators as defined by the strftime(3) library call. NOTE: There are some combinations of formats that will display successfully, but cannot be translated for input.

ARHOME Defines the location of the local home environment for the user. It is the directory where the config file is stored and where the default arcmd directory is located. If not specified, this directory defaults to **$HOME/arHome**.

ARPATH Defines a search path of directories that may contain macro definitions. The value is a set of one or more directories separated by colons. The directory $ARHOME/arcmds is always assumed to be on the path so all directories defined here are in addition to this directory.

FILES

/etc/ar

$ARHOME/config

SEE ALSO

arserverd (1), **aruser** (1), **notifier** (1)

5

arascii

NAME

arascii – interactive interface to the AR System

SYNOPSIS

arascii [-s | -q] [-I] [-n] [-d directory] [-x server]
[{ -e | -i } macroName [-p param=value ...]]

DESCRIPTION

arascii is the interface to the AR System for dumb terminals. It allows you to submit new entries to the AR System and to query and manage existing entries.

To submit an entry, open a Submit window. You simply fill in the data on the screen that represents the problem/request you are submitting.

You can specify qualification criteria to select existing entries that match a given set of conditions. You can then proceed to view and/or change the entries selected. Note that the entries you have access to and what data for each entry you can access are determined by the permissions you have been given in the system.

When the program runs, it stores cached and personalized configuration information in a directory on the local machine. By default, that directory is **$HOME/arHome** where $HOME is your home directory. If you want the local information stored in another location, you must set the **ARHOME** environment variable to the directory you want to use. You must have write access to the directory you choose.

To allow access to the on-line help subsystem, you must set the environment variable ARHELP to the help directory where the AR system is installed.

OPTIONS

The following options to **arascii** may appear in any order on the command line:

-d The directory that the macro indicated by the **-e** or **-i** parameter is located in. This parameter is optional. If it is not specified, the macro is assumed to be in the default location for the user running the command (**$ARHOME/arcmds**).

-e Run the indicated macro at system startup and exit the program when the macro has been completed. This option is good for running the tool to perform some specific operation (for example, generate a report) but not come up interactively. This is especially useful when running the system from a batch script.

-i Run the indicated macro at system startup and the program remains active when the macro has been completed. This option is good for setting the system into an initial state at startup.

-p Specify a value for a parameter. There may be more than one **-p** option in a command line. If the macro specified (using the **-e** or **-i** options) has a parameter, a value can be supplied by naming that parameter and assigning a value. If either the parameter name or value includes a space or other special character that is interpreted by the command line, the parameter must be enclosed in quotes to stop the interpretation of the special characters.

-q This option performs no operation. It is supplied for backward compatibility.

-s Open a submit window on the initial schema at startup. If there is no initial schema, this option does nothing.

-x Specify the name of a server to connect to. This option may be included more than once to connect to multiple servers.

By default, the list of servers the tool will connect to is defined in the directory file **/etc/ar**. If this option is not specified, this file will be used to determine where to connect. If this option is specified one or more times, the servers specified in these options will be used.

ENVIRONMENT

ARDATE	Defines the date format to be used by the program. By default, the date format is determined by the format associated with the language identified by the LANG environment variable. If the default format is not sufficient, you can set this variable. The value is a string of operators as defined by the strftime(3) library call. NOTE: There are some combinations of formats that will display successfully, but cannot be translated for input.
ARHELP	Defines the directory that contains the help files for the AR System. If this variable is not defined, on-line help will not be accessible to this program.
ARHOME	Defines the location of the local home environment for the user. It is the directory where the config file is stored and where the default arcmd directory is located. If not specified, this directory defaults to **$HOME/arHome**.
ARPATH	Defines a search path of directories that may contain macro, custom report, or user command definitions. The value is a set of one or more directories separated by colons. The directory **$ARHOME/arcmds** is always assumed to be on the path so all directories defined here are in addition to this directory.
ARRPC	Defines the specific RPC socket to communicate with during the run of the program. If there is no AR System server running on the identified socket, an error will be returned and the program will not run. In general, this variable is not used.
LPDEST	Defines the default printer for the user. If a report to printer is issued and no printer is specified, the default printer is used. If this variable is set, it defines the printer to use. If not set, the system default printer is used.
NT_SPAWN_DELAY	Defines the timeout interval after which the system will stop waiting for an acknowledgment from a tool we are trying to share. The default is 30 seconds. It can be adjusted if needed, although there is rarely a need to adjust this setting.

EXAMPLES

arascii -s &

Start arascii and open a submit window on the initial schema.

arascii -iTestQuery -p parm = 10 &

Start arascii, execute the macro named TestQuery, substituting the value of 10 for the parameter named parm. The program will remain running after the macro is executed.

arascii -x fred &

Start arascii specifying the machine fred as the machine running the AR System server process. The directory file **/etc/ar** is not accessed.

arascii -p “which user=Fred” &

To set the parameter “which user” to the value Fred, use -p “which user=Fred”.

FILES

/etc/ar
$ARHOME/config
$ARHOME/*.ard
$ARHOME/*.arf
$ARHOME/*.arv
$ARHOME/arcmds/*.arc
$ARHOME/arcmds/*.arq
$ARHOME/arcmds/*.arr

SEE ALSO

aradmin (1), **arserverd** (1), **notifier** (1)

arcache

NAME

arcache – update an entry in one or more user/group caches in the AR System

SYNOPSIS

arcache {-U | -G} {a | d} [-d] -e entryId [-n name] [-s server]
[-g groupList] [-ld flashboardLicense] [-lf fulltextLicense]
[-lw writeLicense] [-m mailAddress] [-p password]
[-x notifyMech]
[-i groupId] [-t groupType]

DESCRIPTION

arcache is the interface allowing you to update a single user/group entry in the access control cache for one or more AR System servers. You specify the operation you want to perform and the information about the item to update. The program will send that update to all the appropriate target servers to update their caches with the new information.

This program is only used in a multi-server environment where there is a desire to have a centralized access control scheme. All updates from a user/group schema to the local access control cache are performed automatically.

This program is generally run using a filter in the AR System. A pair of filters can be defined which execute on Submit and Modify to the User and Group schemas. These filters would allow the system to perform an appropriate update operation on all servers whenever the schemas that contain the user/group information are updated.

OPTIONS

There is a set of common options that can be used with either users or groups and a set of options specific to one or the other. Each option is identified below according to which type of option it is.

The following options to **arcache** may appear in any order on the command line:

-d Set the system into debugging mode. This mode prints messages to stdout that detail the progress of the operations it is performing. Debugging mode should be used only to find problems with how the **arcache** process is running.

-e Identifies the entry ID of the corresponding entry in the User or Group schema. This is the key for the entry and must be supplied for both users and groups.

-g A list of groups defining the user's permissions in the system. This list consists of a set of one or more group IDs separated by semicolons. This field is used only when adding/updating users.

For example, for a new user who was an administrator, the group list value would be "1;". If the user had Customize capability and was a member of the Staff group (which had an ID of 43), the value would be "2; 43;".

-G Identifies this operation as an operation on the group cache. This tag must be followed by the type of operation to be performed:

a - Add a new or update an existing group

d - Delete an existing group

This option is mutually exclusive with the -U option.

-i The ID for the group. This field is used only when adding/updating groups.

-ld The flashboard license type (0 - none, 1 - fixed, or 2 - floating) to be issued to this user. This field is used only when adding/updating users.

If no flashboard license type is specified, it will default to 0 (none).

-lf The full text license type (0 - none, 1 - fixed, or 2 - floating) to be issued to this user. This field is used only when adding/updating users.

If no full text license type is specified, it will default to 0 (none).

-lw The write license type (0 - read, 1 - fixed, or 2 - floating) to be issued to this user. This field is used only when adding/updating users.

If no write license type is specified, it will default to 0 (read).

-m The email address for the user. The address is used by default when a message is to be sent to the user. This field is used only when adding/updating users.

-n The name of the user/group. This field is required for add operations and is recommended (but not required) for delete operations. This field is used with both users and groups.

-p The password for the user. This field is used only when adding/updating users.

-s The name of a single server. Ordinarily, the program will update the entry in ALL servers in the system. It finds all servers by reading the **/etc/ar** file (in UNIX) or **<ar_config_dir>\ar** (in the Windows NT server) and contacting all AR System servers it identifies. With this option, you can identify a single specific server to be updated with the information. This field can be used with both users and groups.

-t The type of the group (1 - view only or 2 - view/change). This field is used only when adding/updating groups.

-U Identifies this operation as an operation on the user cache. This tag must be followed by the type of operation to be performed:

a - Add a new or update an existing user

d - Delete an existing user

This option is mutually exclusive with the -G option.

-x The default notify mechanism for the user. The notify mechanism is used when a notification is delivered to the user via the default notify method for that user. This field is used only when adding/updating users.

If no notify mechanism is specified, it will default to 1 (notifier).

EXAMPLES

arcache -Ua -e000000000000104 -n "Fred Johnson" -m "fredj@remedy.com" -x 1

Add a new user to all server caches. The new user is Fred Johnson with no password, an email address of fredj@remedy.com, and a default notify mechanism of 1 (via the notifier, see ar.h). The entry ID of the entry in the User schema is 000000000000104. Fred will be added to the cache as a user who is a member of no access group.

FILES

/etc/ar (UNIX)

<*ar_config_dir*>\ar (Windows NT)

SEE ALSO

arreload (1), **arserverd** (1)

arimport

NAME

arimport – interactive interface to import data into the AR System

SYNOPSIS

arimport [[-x server] ...] [-d directory] [-m mappingName] [-b]

DESCRIPTION

arimport is the data import interface to the AR System. It allows you to import data from a file into the AR System. This file could have been generated by **aruser** or some other application.

To import a data file, you should first open a schema and open a file. The file could be in AR Export, Comma Separated Values (CSV), or plain text (ASCII) format. Once the schema and file have been opened, mappings need to be defined for the schema fields, then the file can be imported to the schema. The tool can automatically define mappings by matching field ids and field names, but you can also map each field individually. The field mapping can be any text string, an import file field, a keyword, or any combination of the above.

Once the mappings are defined, you can save the mappings. Once saved, the mapping can be loaded at a later time and used again. The mappings can be saved in any directory, but the default is $ARHOME/arcmds.

When the program runs, it stores personalized configuration information in a directory on the local machine. By default, that directory is **$HOME/arHome** where $HOME is your home directory. If you want the local information stored in another location, you must set the **ARHOME** environment variable to the directory you want to use. You must have write access to the directory you choose.

All error messages will be logged in the import log file, whose name can be changed from the Preferences window.

OPTIONS

The following options to **arimport** may appear in any order on the command line:

-b Load the indicated mapping at tool startup and exit the tool when the import of the file indicated in the mapping has been completed. The tool does not display any windows when this option is used.

-d The directory that the mapping indicated by the **-m** parameter is located in. This parameter is optional. If it is not specified, the mapping is assumed to be in the default location for the user running the command ($ARHOME/arcmds).

-m Load the indicated mapping at system startup. This option is good for loading an initial mapping.

-x Specify the name of a server to connect to. This option may be included more than once to connect to multiple servers.

By default, the list of servers the tool will connect to is defined in the directory file **/etc/ar**. If this option is not specified, this file will be used to determine where to connect. If this option is specified one or more times, the servers specified in these options will be used.

ENVIRONMENT

ARDATE	Defines the date format to be used by the program. By default, the date format is determined by the format associated with the language identified by the LANG environment variable. If the default format is not sufficient, you can set this variable. The value is a string of operators as defined by the strftime(3) library call. NOTE: There are some combinations of formats that will display successfully, but cannot be translated for input.
ARHELP	Defines the directory that contains the help files for the AR System. If this variable is not defined, on-line help will not be accessible to this program.
ARHOME	Defines the location of the local home environment for the user. It is the directory where the config file is stored and where the default arcmd directory is located. If not specified, this directory defaults to **$HOME/arHome**.
ARPATH	Defines a search path of directories that may contain mapping definitions. The value is a set of one or more directories separated by colons. The directory **$ARHOME/arcmds** is always assumed to be on the path so all directories defined here are in addition to this directory.
ARRPC	Defines the specific RPC socket to communicate with during the run of the program. If there is no AR System server running on the identified socket, an error will be returned and the program will not run. In general, this variable is not used.

EXAMPLES

arimport -m "DB Mapping" &

Start arimport and load the mapping named “DB Mapping”. The program will remain running after the mapping is loaded.

arimport -m “DB Mapping” -d /usr/mappings -b

Start arimport and load the mapping named “DB Mapping” from the directory /usr/mappings, then import the data from the import file specified in the mapping. The program will exit after the file is imported.

arimport -x fred &

Start arimport specifying the machine fred as the machine running the AR System server process. The directory file is not accessed.

FILES

/etc/ar $ARHOME/config

$ARHOME/arcmds/*.arm

SEE ALSO

aruser(1), **aradmin**(1), **arserverd**(1)

armaild

NAME

armaild – AR System mail daemon

SYNOPSIS

armaild [-d] [-f filename] [-n number-of-intervals]

DESCRIPTION

armaild supplies the mail interface to the AR System. It accepts mail messages containing new ARs and creates the new entries in the AR System. It also accepts messages containing queries for information that exists in the database.

Before the mail daemon can be used, you must first set up a mail address for messages to be mailed to the AR System. By default, this address is **ARSystem**; however, you can use any address you desire by supplying a value for the **Address:** setting in the configuration file.

The **armaild** process will watch that mailbox and when a new entry arrives, retrieve the message and process its contents. If the message has a valid format, either a new entry is submitted to the system or a query is run against the system. If a submit is requested and the entry is successfully submitted, a message containing the submitted text and the ID of the newly created entry are returned. If there is an error during submit, the submitted text and all errors generated are returned. If a query is requested, a list of matching entries is returned. If the message format is not valid, the message is rejected and error messages along with the original message are returned.

OPTIONS

The following options to **armaild** may appear in any order on the command line:

-d Set the system into debugging mode. This mode prints messages to stdout that detail the progress of the operations it is performing. Debugging mode should be used only to find problems with how the **armaild** process is running.

(For UNIX) Messages are printed to stdout.

(For NT) Messages are printed to stdout if the -m option is specified or to the file armaild.log in the db directory under the AR System installation directory.

-f Process the indicated file for command options to the **armaild** process. The various options that can be defined are discussed below.

If not specified, no configuration file is associated with this run and all the system defaults apply.

-m (NT only) This option acts as a flag to control what occurs with the debug output. If specified, debug output is directed to the current window. If not specified, debug output is directed to a file.

-n The number of polling intervals this daemon will run. This setting allows you to limit the amount of time the daemon will run.

If not specified, the system will run until terminated.

Configuration file settings:

Address: The mail address to be used as the address of the mailbox to watch for messages to the AR System. This address does not have to be associated to a user on the system, just with a mail address. You can specify any address you want, but it should be used strictly for the AR System.

Default: ARSystem

Default-Password: The AR System password to use if there is no "Password:" specified in the submitted message. If there is a "Password:" line in the submitted message, it overrides this setting.

Default: no password

Default-Schema: The schema to submit to or query on if there is no "Schema:" specified in the submitted message. If there is a "Schema:" line in the submitted message, it overrides this setting.

Default: none, the schema must be specified in the message

Default-Server: The server to submit to if there is no "Server:" specified in the submitted message. If there is a "Server:" line in the submitted message, it overrides this setting.

Default: the same machine as the one running the **armaild** process

Default-User: The AR System login user to use if there is no "Login:" specified in the message. If there is a "Login:" line in the message, it overrides this setting.

Default: Mailer daemon

Include-Original-On-Failure: A flag indicating whether to include the full text of the original message in a reply to a failed submission. Legal values are T and F.

Default: T

Include-Original-On-Success: A flag indicating whether to include the full text of the original message in a reply to a successful submission. Legal values are T and F.

Default: T

Poll-Interval: The number of seconds to wait between polls to the mailbox to check for new messages. The minimum interval is 5 seconds and any time shorter than 5 will default to 5.

Default: 300

Reply-Failure: The email address to use to send replies to failed submissions. This definition is used to redirect all replies for failed submissions to a third party instead of sending the response to the sender of the message.

(For UNIX) Set the address to /dev/null (or to an address directed to /dev/null in the mail aliases file with a line like nobody: /dev/null) to suppress sending of a message on failure.

(For Windows NT) Set the address to discard to suppress sending of a message on failure.

Default: Send the reply to the sender of the message. Use the "Reply-to:" field of the header, then the "From:" field, and finally the "From" header to try and find the user to respond to.

Reply-Success: The email address to use to send replies to successful submissions. This definition is used to redirect all replies for successful submissions to a third party instead of sending the response to the sender of the message.

(For UNIX) Set the address to /dev/null (or to an address directed to /dev/null in the mail aliases file with a line like nobody: /dev/null) to suppress sending of a message on success.

(For Windows NT) Set the address to discard to suppress sending of a message on success.

Default: Send the reply to the sender of the message. Use the “Reply-to:” field of the header, then the “From:” field, and finally the “From” header to try and find the user to respond to.

Required-Schema: The only schema for which messages will be accepted. If there is a “Schema:” line in the message, it must contain this schema name or the message will be rejected. If there is no “Schema:” line, the setting of the **Default-Schema** is used to determine if there is a default or if “Schema:” is required. If there is a default, it must match this value.

Default: there is no required schema setting

Required-Server: The only server for which messages will be accepted. If there is a “Server:” line in the message, it must contain this server name or the message will be rejected. If there is no “Server:” line, the setting of the **Default-Server** is used to determine if there is a default or if “Server:” is required. If there is a default, it must match this value.

Default: there is no required server setting

NOTES

For backward compatibility with earlier releases, the **armaild** process will also accept the address and poll interval as positional parameters with no option code. For example, **armaild ARSystem 60** is equivalent to running with a configuration file containing settings for **Address:** and **Poll-Interval:**. The old positional style of specifying options is present only for backward compatibility to allow existing registrations to continue operating as they have been. It should not be used for any new registrations of the daemon.

ENVIRONMENT

ARDATE Defines the date format to be used by the program. By default, the date format is determined by the format associated with the language identified by the LANG environment variable. If the default format is not sufficient, you can set this variable. The value is a string of operators as defined by the strftime(3) library call. NOTE: There are some combinations of formats that will display successfully, but cannot be translated for input.

ARRPC Defines the specific RPC socket to communicate with during the run of the program. If there is no AR System server running on the identified socket, an error will be returned and the program will not run. In general, this variable is not used.

MAIL Defines the path to the mail directory. In general, the default value is sufficient, but if it is not correct, this setting will allow you to define where the directory is located. The default for SGI, HP, Solaris, and Motorola is /usr/mail, for NCR is /var/mail, and for SunOS and IBM is /var/spool/mail.

SEE ALSO

arserverd (1), **aruser** (1)

arnvd

NAME

arnvd – IBM NetView/6000 event stream to AR System translation daemon

SYNOPSIS

arnvd [-d] [-p password]

DESCRIPTION

arnvd is the daemon that takes SNMP traps from the IBM NetView/6000 event stream and translates the traps that are of interest into AR System entries. It reads translation requests from the file **nv.ar**. Every trap received by IBM NetView/6000 is forwarded to this daemon for processing. When the trap is received, it is checked against all the translations specified. If it matches one, an AR System entry is generated according to the translation. If the trap does not match any of the requested translations, it is ignored.

The entries that are created will be submitted by the user *IBM NetView/6000* with the password specified by the **-p** option. Field values will be assigned as specified in the translation.

OPTIONS

The following options to **arnvd** may appear in any order on the command line:

-d Run the program in debug mode. For each trap received, messages will be printed to stdout to display the progress toward mapping the trap into an entry in the AR System. This mode is especially useful if a specific type of trap is not being mapped and you expect it to be.

-p Password for the *IBM NetView/6000* user. If there is no password defined for this user or if the user is not registered with the AR System, this option must be omitted.

EXAMPLES

arnvd -p fred

Run the IBM NetView/6000 translation daemon where the password for the *IBM NetView/6000* user is fred.

FILES

/etc/ar

/etc/ar.conf

<*ar_install_dir*>/db/nv.ar

SEE ALSO

arserverd (1), **arnvui** (1), **nvw** (1)

arnvui

NAME

arnvui – define translations of IBM NetView/6000 traps to AR System entries

SYNOPSIS

arnvui [-p password]

DESCRIPTION

arnvui is the interface allowing you to specify translations from the IBM NetView/6000 event stream to the AR System. At startup, the tool will open the **nv.ar** file and display all the existing translations. You can delete or modify existing translations or create entirely new ones. Remember that the translation list is processed serially so the order of the translations is important (especially when wild cards are used).

When the changes are saved, the **arnvd** process will pick up the new definitions when the next trap from IBM NetView/6000 arrives. You do not need to take any further action to have your new translations take affect.

OPTIONS

The following option to **arnvui** may appear on the command line:

-p Password for the *IBM NetView/6000* user. If there is no password defined for this user or if the user is not registered with the AR System, this option must be omitted.

EXAMPLES

arnvui -p fred

Run the process to specify translations where the password for the *IBM NetView/6000* user is fred.

FILES

/etc/ar.conf

<*ar_install_dir*>/db/nv.ar

SEE ALSO

arserverd (1), **arnvd** (1), **nvw** (1)

arovd

NAME

arovd – HP OpenView event stream to AR System translation daemon

5

SYNOPSIS

arovd [-d] [-p password]

DESCRIPTION

arovd is the daemon that takes SNMP traps from the HP OpenView event stream and translates the traps that are of interest into AR System entries. It reads translation requests from the file **ov.ar**. Every trap received by HP OpenView is forwarded to this daemon for processing. When the trap is received, it is checked against all the translations specified. If it matches one, an AR System entry is generated according to the translation. If the trap does not match any of the requested translations, it is ignored.

It is important that this daemon is run AFTER SunNet manager is started. This is because the daemon registers with the SunNet Manager process. The SunNet Manager process must be active first or no events will be passed.

The entries that are created will be submitted by the *HP OpenView* user with the password specified by the **-p** option. Field values will be assigned as specified in the translation.

OPTIONS

The following options to **arovd** may appear in any order on the command line:

-d Run the program in debug mode. For each trap received, messages will be printed to stdout to display the progress toward mapping the trap into an entry in the AR System. This mode is especially useful if a specific type of trap is not being mapped and you expect it to be.

-p Password for the *HP OpenView* user. If there is no password defined for this user or if the user is not registered with the AR System, this option must be omitted.

EXAMPLES

arovd -p fred

Run the HP OpenView translation daemon where the password for the *HP OpenView* user is fred.

FILES

/etc/ar

/etc/ar.conf

<*ar_install_dir*>/db/ov.ar

SEE ALSO

arserverd (1), **arovui** (1), **ovw** (1)

arovui

NAME

arovui – define translations of HP OpenView traps to AR System entries

SYNOPSIS

arovui [-p password]

DESCRIPTION

arovui is the interface allowing you to specify translations from the HP OpenView event stream to the AR System. At startup, the tool will open the **ov.ar** file and display all the existing translations. You can delete or modify existing translations or create entirely new ones. Remember that the translation list is processed serially so the order of the translations is important (especially when wild cards are used).

When the changes are saved, the **arovd** process will pick up the new definitions when the next trap from HP OpenView arrives. You do not need to take any further action to have your new translations take affect.

OPTIONS

The following option to **arovui** may appear on the command line:

-p Password for the *HP OpenView* user. If there is no password defined for this user or if the user is not registered with the AR System, this option must be omitted.

EXAMPLES

arovui -p fred

Run the process to specify translations where the password for the *HP OpenView* user is fred.

FILES

/etc/ar.conf

<*ar_install_dir*>/db/ov.ar

SEE ALSO

arserverd (1), **arovd** (1), **ovw** (1)

5

arreload

NAME

arreload – reload all the user/group cache entries from a given schema

SYNOPSIS

arreload { -u | -g } schema -a adminUser [-d] [-f] [-p adminPassword] [-s server]

DESCRIPTION

arreload is the interface allowing you to reload the user/group caches from a given user/group schema to one or more AR System servers. The process will delete all the existing cached entries for the selected schema and load all the information from the schema into the cache. This is useful for bringing all the servers into sync.

Note that this process must be run on the same machine as the AR System server that contains the schema to be reloaded.

In a multi-server environment, this program is generally run as part of a periodic maintenance operation for the AR System. Setting up a job to run on a weekly or monthly basis that does a reload of the user/group caches helps insure that all the caches remain in sync with all the latest information. In a single-server environment, this program is generally unused (except to recover from disk crashes).

OPTIONS

The following options to **arreload** may appear in any order on the command line:

-a The user name of a user with Administrator access to the schema to be reloaded.

-d Set the system into debugging mode. This mode prints messages to stdout that detail the progress of the operations it is performing.

-f Flush the user or group cache definition of entries from ALL servers before reloading the cache with the specified definitions. This option is especially useful when you are changing or renaming the machine that is running the AR System.

-g Identifies this operation as a group reload and supplies the name of the schema that is to be reloaded.

-p The password for the Administrator user identified with the **-a** option. If there is a password for the Administrator identified (which is recommended), you must specify this option with the appropriate password.

-s The name of a single server. Ordinarily, the program will update the entry in ALL servers in the system. It finds all servers by reading the **/etc/ar** file (in UNIX) or **<ar_config_dir>\ar** (in Windows NT) and contacting all AR System servers it identifies. With this option, you can identify a single specific server to be updated with the information.

-u Identifies this operation as a user reload and supplies the name of the schema that is to be reloaded.

EXAMPLES

arreload -a Admin -p password -u User

Reload the user cache of all servers to remove any existing definitions from the user schema on this server and add entries for all the current users defined in the User schema. The administrator user is Admin with a password of password to allow access to the schema to get the information to be reloaded.

FILES

/etc/ar (UNIX)

<*ar_config_dir*>\ar (Windows NT)

SEE ALSO

arcache (1), **arserverd** (1)

arserverd

NAME

arserverd – AR System server daemon

SYNOPSIS

arserverd [-r rpcSocket]

DESCRIPTION

arserverd is the main server daemon for the AR System. This program is the core of the AR System. It handles all interaction between the clients and the database.

Although this program must be run before any access to the AR System is possible, it is generally launched and managed by the AR System controller daemon, **arservtcd**. Private servers can be started manually if they are licensed. If the process is accidentally (or purposely) shut down, it can simply be restarted at any time.

arserverd will reread all its configuration files and reset all cached information about structures in the AR System whenever it receives a **SIGHUP** signal and read only the configuration file if it receives a **SIGUSR1** signal. These signals would generally be used only when there was some manual repair or restore operation performed. If done accidentally, no damage will be done and there is no affect on any users currently accessing the AR System.

OPTIONS

The following options to **arserverd** may appear in any order on the command line:

-r Set a specific RPC socket number for this program. This option is used only if the system is running in Multi-Process mode. It identifies the specific instance of the **arserverd** program that is to be run.

ENVIRONMENT

ARDATE Defines the date format to be used by the program. By default, the date format is determined by the format associated with the language identified by the LANG environment variable. If the default format is not sufficient, you can set this variable. The value is a string of operators as defined by the strftime(3) library call. NOTE: There are some combinations of formats that will display successfully, but cannot be translated for input.

ARRPC Defines the specific RPC socket to open for this program. This variable is overridden by the use of the -r command line argument.

FILES

/etc/ar (UNIX)

/etc/ar.conf (UNIX)

/etc/remedy.lic (UNIX)

<ar_config_dir>\ar (Windows NT)

<ar_config_dir>\remedy.lic (Windows NT)

<ar_config_dir>\ar.cfg (Windows NT)

SEE ALSO

aradmin (1), **arservdsd(1)**, **arservftd(1)**, **arservtcd(1)**, **aruser** (1), **ntserverd** (1)

arservdsd

NAME

arservdsd – AR System Distributed Server Option daemon

SYNOPSIS

arservdsd [-d] [-r rpcSocket]

DESCRIPTION

arservdsd is the server daemon for the AR System Distributed Server Option. This program is responsible for all the distributed server operations performed by the system. It works as a client to the **arserverd** program to read and write data.

Although this program must be run before any distributed server operations are possible, it is generally launched and managed by the AR System controller daemon, **arservtcd**. If the process is accidentally (or purposely) shut down, it can simply be restarted at any time. arserverd will reread all its configuration files and reset all cached information about structures in the AR System whenever it receives a SIGHUP signal, read only the configuration file if it receives a SIGUSR1 signal, and recheck its list of pending distributed operations whenever it receives a SIGUSR2 signal. These signals would generally be used only when there was some manual repair or restore operation performed. If done accidentally, no damage will be done and there is no affect on any users currently accessing the AR System.

OPTIONS

The following options to **arservdsd** may appear in any order on the commandline:

-d Activate the debug trace mode for the distributed server daemon. This is an override option that allows the debug trace mode to be turned on and displayed to stdout. It will override the setting for the debug trace mode configured through the **aradmin** tool.

-r Set a specific RPC socket number for this program to use when communicating with the **arserverd** process. This option can be used only if the system is running in Multi-Process mode. An instance of the **arserverd** process must be running on this socket on ALL machines in the distributed environment.

ENVIRONMENT

ARRPC Defines the specific RPC socket to use for this program. This variable is overridden by the use of the **-r** command line argument.

FILES

/etc/ar

/etc/ar.conf

/etc/remedy.lic

SEE ALSO

aradmin (1), **arserverd**(1), **arservtcd**(1)

arservftd

NAME

arservftd – AR System Full Text indexer daemon

SYNOPSIS

arservftd

DESCRIPTION

arservftd is the indexer process for the Full Text search capability of the AR System. It is responsible for performing the management and update of the Full Text indexes in the system.

This program is launched by the **arservtcd** process. It is responsible for performing all the indexing operations needed for the Full Text search capability of the system.

FILES

/etc/ar.conf

/usr/ar/db/arftp.lst

/usr/ar/db/arft.log

/usr/ar/db/arftext.log

SEE ALSO

arserverd(1), **arservtcd** (1)

5

arservtcd

NAME

arservtcd - AR System controller daemon

SYNOPSIS

arservtcd

DESCRIPTION

arservtcd is the controller daemon for the AR System. This program launches and manages the one or more instances of the **arserverd** process that are running in the system. In addition, if the Distributed Server Option is licensed, the program will launch and manage the **arservdsd** process.

This program is generally the only AR System process that the user must run. It will launch instances of the server processes as it has been configured. Generally, the program is registered in the system startup file so that the program is started on system startup. If it is accidentally (or purposely) shut down, it can simply be restarted at any time.

arservtcd will reread all its configuration files and reset all child processes whenever it receives a **SIGHUP** or **SIGUSR1** signal. These signals would generally be used only when there was some manual repair or restore operation performed. If done accidentally, no damage will be done and there is no affect on any users currently accessing the AR System.

FILES

/etc/ar.conf

/etc/remedy.lic

SEE ALSO

aradmin (1), **aruser** (1), **arservdsd** (1), **arserverd** (1)

arsnmd

NAME

arsnmd – SunNet Manager event stream to AR System translation daemon

SYNOPSIS

arsnmd [-d] [-p password]

DESCRIPTION

arsnmd is the daemon that takes events from the SunNet Manager event stream and translates events (events are either SunNet Manager threshold events or SNMP traps) that are of interest into AR System entries. It reads translation requests from the file **snm.ar**. Every event received by SunNet Manager is forwarded to this daemon for processing. When the event is received, it is checked against all the translations specified. If it matches one, an AR System entry is generated according to the translation. If the event does not match any of the requested translations, it is ignored.

It is important that this daemon is run *after* SunNet Manager is started. This is because the daemon registers with the SunNet Manager process. The SunNet Manager process must be active first or no events will be passed.

The entries that are created will be submitted by the *SunNet Manager* user with the password specified by the **-p** option. Field values will be assigned as specified in the translation.

OPTIONS

The following options to **arsnmd** may appear in any order on the command line:

-d Run the program in debug mode. For each event/trap received, messages will be printed to stdout to display the progress toward mapping the event/trap into an entry in the AR System. This mode is especially useful if a specific type of trap/event is not being mapped and you expect it to be.

-p Password for the *SunNet Manager* user. If there is no password defined for this user or if the user is not registered with the AR System, this option must be omitted.

EXAMPLES

arsnmd -p fred

Run the SNM translation daemon where the password for the *SunNet Manager* user is fred.

FILES

/etc/ar

/etc/ar.conf

<*ar_install_dir*>/db/snm.ar

SEE ALSO

arserverd (1), **arsnmui** (1), **snm** (1)

arsnmui

NAME

arsnmui – define translations of SunNet Manager events to AR System entries

SYNOPSIS

arsnmui [-p password]

DESCRIPTION

arsnmui is the interface allowing you to specify translations from the SunNet Manager event stream to the AR System. At startup, the tool will open the **snm.ar** file and display all the existing translations. You can delete or modify existing translations or create entirely new ones. Remember that the translation list is processed serially so the order of the translations is important (especially when wild cards are used).

When the changes are saved, the **arsnmd** process will pick up the new definitions when the next event from SunNet Manager arrives. You do not need to take any further action to have your new translations take affect.

OPTIONS

The following option to **arsnmui** may appear on the command line:

-p Password for the *SunNet Manager* user. If there is no password defined for this user or if the user is not registered with the AR System, this option must be omitted.

EXAMPLES

arsnmui -p fred

Run the process to specify translations where the password for the *SunNet Manager* user is fred.

FILES

/etc/ar.conf

<*ar_install_dir*>/db/snm.ar

SEE ALSO

arserverd (1), **arsnmd** (1), **snm** (1)

aruser

NAME

aruser – interactive interface to the AR System

SYNOPSIS

aruser [-s | -q] [-I] [-n] [-d directory] [-x server]
[{ -e | -i } macroName [-p param=value ...]]

DESCRIPTION

aruser is the main interface to the AR System. It allows you to submit new entries to the AR System and to query and manage existing entries.

To submit an entry, open a Submit window. You simply fill in the data on the screen that represents the problem/request you are submitting. Remember that **bold font** labels represent required fields for which you must supply a value, *italic font* labels represent fields that are automatically managed by the system and so are read-only on this screen, and Normal font labels represent optional fields for which you can supply a value or not.

You can specify qualification criteria to select existing entries that match a given set of conditions. You can then proceed to view and/or change the entries selected. Note that the entries you have access to and what data for each entry you can access are determined by the permissions you have been given in the system.

When the program runs, it stores cached and personalized configuration information in a directory on the local machine. By default, that directory is **$HOME/arHome** where $HOME is your home directory. If you want the local information stored in another location, you must set the **ARHOME** environment variable to the directory you want to use. You must have write access to the directory you choose.

OPTIONS

The following options to **aruser** may appear in any order on the command line:

-d The directory that the macro indicated by the **-e** or **-i** parameter is located in. This parameter is optional. If it is not specified, the macro is assumed to be in the default location for the user running the command ($ARHOME/arcmds).

-e Run the indicated macro at system startup and exit the program when the macro has been completed. This option is good for running the tool to perform some specific operation (for example, generate a report) but not come up interactively. This is especially useful when running the system from a batch script.

-i Run the indicated macro at system startup and the program remains active when the macro has been completed. This option is good for setting the system into an initial state at startup.

-I Iconize the program at startup.

-n Do not share an existing instance of the **aruser** program even if it is possible. Run this instance as an independent program.

By default, the system will share a running instance of the program when run with the **-i** option. This makes the system run more efficiently by running the macro in an existing instance of the tool and not paying the overhead penalty for an independent instance. This option allows you to override that default operation and force an independent instance of the tool to be run.

In general, you want to take advantage of the sharing of existing instances as it is both faster and uses the resources on the workstation more efficiently. This option should be used as an exception case when and if needed.

-p Specify a value for a parameter. There may be more than one **-p** option in a command line. If the macro specified (using the **-e** or **-i** options) has a parameter, a value can be supplied by naming that parameter and assigning a value. If either the parameter name or value includes a space or other special character that is interpreted by the command line, the parameter must be enclosed in quotes to stop the interpretation of the special characters.

-q This option performs no operation. It is supplied for backward compatibility.

-s Open a submit window on the initial schema at startup. If there is no initial schema, this option does nothing.

-x Specify the name of a server to connect to. This option may be included more than once to connect to multiple servers.

By default, the list of servers the tool will connect to is defined in the directory file **/etc/ar**. If this option is not specified, this file will be used to determine where to connect. If this option is specified one or more times, the servers specified in these options will be used.

ENVIRONMENT

ARDATE Defines the date format to be used by the program. By default, the date format is determined by the format associated with the language identified by the LANG environment variable. If the default format is not sufficient, you can set this variable. The value is a string of operators as defined by the strftime(3) library call. NOTE: There are some combinations of formats that will display successfully, but cannot be translated for input.

ARHELP Defines the directory that contains the help files for the AR System. If this variable is not defined, on-line help will not be accessible to this program.

ARHOME	Defines the location of the local home environment for the user. It is the directory where the config file is stored and where the default arcmd directory is located. If not specified, this directory defaults to **$HOME/arHome**.
ARPATH	Defines a search path of directories that may contain macro, custom report, or user command definitions. The value is a set of one or more directories separated by colons. The directory **$ARHOME/arcmds** is always assumed to be on the path so all directories defined here are in addition to this directory.
ARRPC	Defines the specific RPC socket to communicate with during the run of the program. If there is no AR System server running on the identified socket, an error will be returned and the program will not run. In general, this variable is not used.
LPDEST	Defines the default printer for the user. If a report to printer is issued and no printer is specified, the default printer is used. If this variable is set, it defines the printer to use. If not set, the system default printer is used.
NT_SPAWN_DELAY	Defines the timeout interval after which the system will stop waiting for an acknowledgement from a tool we are trying to share. The default is 30 seconds. It can be adjusted if needed, although there is rarely a need to adjust this setting.

EXAMPLES

aruser -s &

Start aruser and open a submit window on the initial schema.

aruser -iTestQuery -p parm = 10 &

Start aruser, execute the macro named TestQuery, substituting the value of 10 for the parameter named parm. The program will remain running after the macro is executed.

aruser -x fred &

Start aruser specifying the machine fred as the machine running the AR System server process. The directory file **/etc/ar** is not accessed.

aruser -p “which user=Fred” &

To set the parameter “which user” to the value Fred, use -p “which user=Fred”.

FILES

/etc/ar

$ARHOME/config

$ARHOME/*.ard

$ARHOME/*.arf

$ARHOME/*.arv

$ARHOME/arcmds/*.arc

$ARHOME/arcmds/*.arq

$ARHOME/arcmds/*.arr

SEE ALSO

aradmin (1), **arserverd** (1), **notifier** (1)

license

NAME

license – Remedy License Tool

SYNOPSIS

license

DESCRIPTION

license is a tool that is used to install and maintain licenses for the software from Remedy Corporation. The License Tool allows you to examine and alter existing licenses as well as generate new ones.

The **License** Tool must be run on the system that is being licensed. If you wish, you can run this tool via remote X on some other workstation if it is more convenient. To do this, you should rlogin to the system that is being licensed and then run the License Tool with the *-display local_host:0.0* runtime switch to specify your local display.

Once the **License** Tool has been invoked, you can examine existing licenses by selecting one of the entries in the license table or create a new license by pressing the “New License” button. After filling in the fields on the window (such as selecting the proper feature or license type), you will need to call your software reseller to obtain a key to enable the license.

After you have obtained the key and entered it into the “License Key” field, you press the “Apply” button to validate and save the license. If an error is detected, a notice box will appear warning you that the key is not valid. Otherwise, a message appears at the bottom of the window stating the key is valid and the license is added to the system license file **/etc/remedy.lic**.

FILES

/etc/remedy.lic

SEE ALSO

arserverd (1)

5

notifier

NAME

notifier – “desktop beeper” notification tool

SYNOPSIS

notifier

DESCRIPTION

notifier is the interface to the Notification System. It allows you to register your presence on a given machine and configure parameters about how the system notifies you of new items.

When the program runs, it stores cached and personalized configuration information in a directory on the local machine. By default, that directory is **$HOME/arHome** where $HOME is your home directory. If you want the local information stored in another location, you must set the **NTHOME** environment variable to the directory you want to use. You must have write access to the directory you choose.

ENVIRONMENT

NTHELP	Defines the directory that contains the help files for the AR System. If this variable is not defined, the system will use the setting of the **ARHELP** variable. If neither is set, on-line help will not be accessible to this program.
NTHOME	Defines the location of the local home environment for the user. It is the directory where the config file is stored and where the default arcmd directory is located. If not specified, this directory defaults to $HOME/arHome.

FILES

/etc/ar

$NTHOME/config

SEE ALSO

aradmin (1), **aruser** (1), **ntclientd** (1), **ntserverd** (1)

ntclientd

NAME

ntclientd – client daemon for the Notification System

SYNOPSIS

ntclientd [-d] [-p num-pipes] [-s] [-x server]

DESCRIPTION

ntclientd is the daemon used by the Notification System that runs on the local machine with the notification client. It registers users when one or more notifiers are started and listens for and distributes notifications that are received from the server to the appropriate client(s) on that machine. This daemon is required to allow multiple notifier clients to be running on a single machine. Since only one process can register for rpc calls, there must be a process who is responsible for listening for rpc calls which are then distributed to the appropriate display.

This daemon is automatically launched when the **notifier** program is run.

OPTIONS

The following options to **ntclientd** may appear in any order on the command line:

-d Set the system into debugging mode. This mode prints messages to stdout that detail the progress of the operations it is performing.

-p Specify the maximum number of pipes the system configuration allows to be open by a single process. By default, this value is 11. If you will be having more than 11 processes connected to this daemon, you must reconfigure the system kernel to allow more processes. Do not use this option unless you have reconfigured the system to allow more pipes per process. If this number is more than configured, the notification system can stop functioning and need to be reset to allow communication to resume.

Default: 11

This option applies only to the SunOS operating system.

-s Suppress the message issued if a second copy of this daemon is run. Only one instance of this daemon can be running on a system at a given time. If a second copy is run, an error is issued that indicates there is already a running process. This flag allows the suppression of this message. This is most useful when this program is launched by other programs as a first step in connecting to the notification system.

This option does not allow a second instance of the program to run. It simply suppresses the error message and causes the program to exit quietly.

-x Specify the name of a server to connect to. This option may be included more than once to connect to multiple servers.

By default, the list of servers the tool will connect to is defined in the directory file **/etc/ar**. If this option is not specified, this file will be used to determine where to connect. If this option is specified one or more times, the servers specified in these options will be used.

FILES

/etc/ar

/usr/tmp/ntclient.lck

SEE ALSO

notifier (1), **ntserverd** (1)

ntserverd

NAME

ntserverd – server daemon for the Notification System

SYNOPSIS

ntserverd [-d]

DESCRIPTION

ntserverd is the server daemon for the Notification System. It can run on any machine in the system although it is likely that it will run on the same machine as an AR System server. It accepts registration of users and will deliver any notification message it receives to the target user if they are registered, wherever they are registered.

Notifications for a user that is registered in one or more locations are delivered immediately to that user. If the user is not currently registered, the notifications are held in the file **/usr/spool/remedy/ntserver.log**. When a user registers an active connection, all pending notifications are delivered to that user. The notification log is periodically purged of all entries over 30 days old to prevent overflow of the notification log with obsolete notifications.

To survive system problems, the daemon records all current registrations in a file named **/usr/spool/remedy/ntusers**. This way, if the **ntserverd** process is shut down or the system fails and has to restart, the program can read a list of all the users it last had registered from this file. This allows the system to gracefully recover and maintain “connection” to registered users.

OPTIONS

The following option to **ntserverd** may appear on the command line:

-d Set the system into debugging mode. This mode prints messages to stdout that detail the progress of the operations it is performing. Debugging mode should be used only to find problems with how the **ntserverd** process is running.

FILES

/usr/spool/remedy/ntserver.log

/usr/spool/remedy/ntusers

/usr/tmp/ntserver.lck

SEE ALSO

arserverd (1), **notifier** (1), **ntclientd** (1)

C Library Functions

ARCreateActiveLink

NAME

ARCreateActiveLink – create a new active link in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARCreateActiveLink (control, name, order, schema, groupList, executeMask, field, displayList, enable,
        query, actionList, helpText, owner, changeDiary, status)
    ARControlStruct          *control;
    ARNameType               name;
    unsigned int             order;
    ARNameType               schema;
    ARInternalIdList         *groupList;
    unsigned int             executeMask;
    ARInternalId             *field;
    ARDisplayList            *displayList;
    unsigned int             enable;
    ARQualifierStruct        *query;
    ARActiveLinkActionList   *actionList;
    char                     *helpText;
    ARNameType               owner;
    char                     *changeDiary;
    ARStatusList             *status;
```

DESCRIPTION

ARCreateActiveLink will create a new active link with the indicated name on the specified server. The active link will be added to the server immediately and will be returned to users who request information about active links. Since the operation of active links is on clients accessing the server, the new definition will not be available on individual clients until the client reloads configuration from the server (by reconnecting to the schema).

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the active link to create. The names of all active links on a given server must be unique.
order	The order of the new active link. The active link order is a code between 0 and 1000, inclusive. It allows the ordering of active links so that active links with lower orders are executed before active links with higher orders. So, the active link order allows you to specify the order in which active links will be processed and lets you insure that given active links will be performed in the order you desire.
schema	The name of the schema the active link is linked to. The active link *must* be tied to a single, specific schema. This schema must currently exist on the server.
groupList	A list of 0 or more groups identifying the groups which will have access to this active link. Users can perform an active link if they are a member of a group that has access to the active link. Defining a groupList with 0 items will define an active link that can be accessed only by users with Administrator capability. Defining a groupList that contains the "Public" group (group id 0) will define an active link that can be accessed by all users.
executeMask	A bit mask of the conditions under which this active link will be executed. This field consists of one or more of the following values OR'ed together: **AR_EXECUTE_ON_BUTTON**, **AR_EXECUTE_ON_RETURN**, **AR_EXECUTE_ON_SUBMIT**, **AR_EXECUTE_ON_MODIFY**, **AR_EXECUTE_ON_DISPLAY**, **AR_EXECUTE_ON_MENU_CHOICE**, and **AR_EXECUTE_ON_SET_DEFAULT**.
field	The id of the field the active link is tied to if the execute mask includes the AR_EXECUTE_ON_RETURN or AR_EXECUTE_ON_MENU_CHOICE flag. If the flag is included, a value must be supplied for this field. Otherwise, any value supplied is ignored.
displayList	Information about the button and its position if the execute mask includes the AR_EXECUTE_ON_BUTTON flag. If the flag is included, a value is expected for this field.
enable	A flag with a setting of 0 to indicate that this active link is to be marked as disabled so it will not be executed or 1 to indicate that the link is active and available for use. An active link that is disabled will not be visible to the end user and will not fire.
query	A qualification that is to be performed when the active link is executed. It will allow the conditional execution of the active link. If the condition specified by the query is met, the active link executes; otherwise, it is not performed (as if there were no active link). If there is no qualification, specify NULL or assign a value of AR_COND_OP_NONE to this value.

actionList The set of one or more actions to take when the active link is executed. Every active link MUST have at least one action and may have as many as AR_MAX_ACTIONS actions.

helpText The help text that is to be associated with the active link. The help text can be of any length.

If no help text is to be assigned, a NULL pointer should be supplied for this parameter.

owner The owner for the active link.

If NULL is specified for this parameter, the owner will default to the user performing the operation.

changeDiary The initial change diary that is to be associated with the active link. The change diary text can be of any length. When saved, the time of the change and the user who made the change are added to the change diary (called time- and user-stamping).

If no change diary is to be assigned, a NULL pointer should be supplied for this parameter.

RETURN VALUES

status A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK:** Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARDeleteActiveLink (3), **ARGetActiveLink** (3), **ARGetListActiveLink** (3), **ARSetActiveLink** (3), **FreeARActiveLinkActionList** (3), **FreeARDisplayList** (3), **FreeARInternalIdList** (3), **FreeARQualifierStruct** (3), **FreeARStatusList** (3)

5

ARCreateAdminExtension

NAME

ARCreateAdminExtension – create a new administrator extension in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARCreateAdminExtension (control, name, groupList, command, helpText, owner, changeDiary, status)
    ARControlStruct     *control;
    ARNameType          name;
    ARInternalIdList    *groupList;
    char                *command;
    char                *helpText;
    ARNameType          owner;
    char                *changeDiary;
    ARStatusList        *status;
```

DESCRIPTION

ARCreateAdminExtension will create a new administrator extension with the indicated name on the specified server.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the extension to create. The names of all extensions on a given server must be unique.
groupList	A list of 0 or more groups identifying the groups which will have access to this extension. Users can perform an extension if they are a member of a group that has access to the extension. Defining a groupList with 0 items will define an administrator extension that can be accessed only by users with Administrator capability. Defining a groupList that contains the “Public” group (group id 0) will define an administrator extension that can be accessed by all users.
command	The command that forms the administrator extension. There must be a command value specified.
helpText	The help text that is to be associated with the extension. The help text can be of any length. If no help text is to be assigned, a NULL pointer should be supplied for this parameter.

owner — The owner for the extension.

If NULL is specified for this parameter, the owner will default to the user performing the operation.

changeDiary — The initial change diary that is to be associated with the extension. The change diary text can be of any length. When saved, the time of the change and the user who made the change are added to the change diary (called time- and user-stamping).

If no change diary is to be assigned, a NULL pointer should be supplied for this parameter.

RETURN VALUES

status — A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARDeleteAdminExtension (3), **ARExecuteAdminExtension** (3), **ARGetAdminExtension** (3), **ARGetListAdminExtension** (3), **ARSetAdminExtension** (3), **FreeARInternalIdList** (3), **FreeARStatusList** (3)

5

ARCreateCharMenu

NAME

ARCreateCharMenu – create a new character menu in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARCreateCharMenu (control, name, refreshCode, menuDefn, helpText, owner, changeDiary, status)
      ARControlStruct          *control;
      ARNameType               name;
      unsigned int             refreshCode;
      ARCharMenuStruct         *menuDefn;
      char                     *helpText;
      ARNameType               owner;
      char                     *changeDiary;
      ARStatusList             *status;
```

DESCRIPTION

ARCreateCharMenu will create a new character menu with the indicated name on the specified server.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the character menu to create. The names of all character menus on a given server must be unique.
refreshCode	Code indicating when the menu should be refreshed. This code allows you to balance performance and the frequency at which the menu is checked for consistency with the server. The refresh code is one of **AR_MENU_REFRESH_CONNECT, AR_MENU_REFRESH_OPEN,** or **AR_MENU_REFRESH_INTERVAL.**
menuDefn	The definition of the character menu.
helpText	The help text that is to be associated with the character menu. The help text can be of any length. If no help text is to be assigned, a NULL pointer should be supplied for this parameter.
owner	The owner for the character menu.

If NULL is specified for this parameter, the owner will default to the user performing the operation.

changeDiary

The initial change diary that is to be associated with the character menu. When saved, the time of the change and the user who made the change are added to the change diary (called time- and user-stamping).

If no change diary is to be assigned, a NULL pointer should be supplied for this parameter.

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARDeleteCharMenu (3), **ARExpandCharMenu** (3), **ARGetCharMenu** (3), **ARGetListCharMenu** (3), **ARSetCharMenu** (3), **FreeARCharMenuStruct** (3), **FreeARStatusList** (3)

5

ARCreateEntry

NAME

ARCreateEntry – create a new entry in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARCreateEntry (control, schema, fieldList, entryId, status)
      ARControlStruct      *control;
      ARNameType           schema;
      ARFieldValueList     *fieldList;
      AREntryIdType        entryId;
      ARStatusList         *status;
```

DESCRIPTION

ARCreateEntry will add a new entry to the specified schema. The user can specify any number of fields and associated values. The system will check permissions for each field and report errors if a field does not exist or if the user does not have access. If any one of the fields is in error, the entire create operation is rejected and no change is made to the database.

Access to entries is controlled through the security scheme of the AR System. The user identified in the control record in combination with the create mode specified for the field is used to determine which fields can be updated by the user. If a value is provided for a field to which the user does not have write access, an error will be reported on that field and the operation will be cancelled. Each value is checked to make sure it can be updated with an error returned if not writable.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
schema	Identifies the schema to which the new entry is to be added.
fieldList	A list of field/value pairs for all the fields to be set for the new entry. The fields can be in any order in the list. Any nonexistent or inaccessible field will result in an error return. The datatype of the value must match the datatype of the field (or be NULL). All required fields that do not have a default value MUST be assigned a value. To assign no value for a field, assign the value NULL (AR_DATA_TYPE_NULL) as the value. If the field is a required field in the system, you cannot assign it a NULL value. Note that this use overrides any configured default value for the field.

RETURN VALUES

entryId	This is the unique identifier for the new entry. The system will create a new entry ID to uniquely identify this entry in all future operations.

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK:** Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARDeleteEntry (3), **ARGetEntry** (3), **ARGetEntryStatistics** (3), **ARGetListEntry** (3), **ARMergeEntry** (3), **ARSetEntry** (3), **FreeARFieldValueList** (3), **FreeARStatusList** (3)

5

ARCreateEscalation

NAME

ARCreateEscalation – create a new escalation in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARCreateEscalation (control, name, escalationTm, schema, enable, query, actionList, helpText, owner,
        changeDiary, status)
    ARControlStruct          *control;
    ARNameType               name;
    AREscalationTmStruct     *escalationTm;
    ARNameType               schema;
    unsigned int             enable;
    ARQualifierStruct        *query;
    ARFilterActionList       *actionList;
    char                     *helpText;
    ARNameType               owner;
    char                     *changeDiary;
    ARStatusList             *status;
```

DESCRIPTION

ARCreateEscalation will create a new escalation with the indicated name on the specified server. The escalation condition will be checked regularly depending on time structure defined when it is enabled.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the escalation to create. The names of all escalations on a given server must be unique.
escalationTm	The checking time of the new escalation. It defines a datetime or frequency of AR System checks for the escalation condition. Escalation time has two formats, one is in seconds as a time interval between checks, the other is a datetime mask specifying on what day of the month or week and at what hour and minute of the day the AR System checks the escalation condition.
schema	The name of the schema the escalation is linked to. The escalation MUST be tied to a single, specific schema. This schema must currently exist on the server.

enable A flag with a setting of 0 to indicate that this escalation is to be marked as disabled so it will not be executed or 1 to indicate that the escalation is active and will be checked on the specified time interval. An escalation that is disabled will not perform its condition checks and will not fire.

query A qualification that is used to search the specified schema. Any records that match the qualification will have the escalation action performed on them.

If there is no qualifying condition, specify NULL or assign a value of AR_COND_OP_NONE to this value.

actionList The set of one or more actions to take when the escalation conditions are met. Every escalation MUST have at least one action and may have as many as AR_MAX_ACTIONS actions.

helpText The help text that is to be associated with the escalation. The help text can be of any length.

If no help text is to be assigned, a NULL pointer should be supplied for this parameter.

owner The owner for the escalation.

If NULL is specified for this parameter, the owner will default to the user performing the operation.

changeDiary The initial change diary that is to be associated with the escalation. The change diary text can be of any length. When saved, the time of the change and the user who made the change are added to the change diary (called time- and user-stamping).

If no change diary is to be assigned, a NULL pointer should be supplied for this parameter.

RETURN VALUES

status A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING:** Warning during process but operation completed successfully
- **AR_RETURN_ERROR:** Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARDeleteEscalation (3), **ARGetEscalation** (3), **ARGetListEscalation** (3), **ARSetEscalation** (3), **FreeARFilterActionList** (3), **FreeARQualifierStruct** (3), **FreeARStatusList** (3)

ARCreateField

NAME

ARCreateField – create a new field in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARCreateField (control, schema, fieldId, reservedIdOK, dataType, option, createMode, defaultVal,
                   permissions, limit, displayList, helpText, owner, changeDiary, status)
    ARControlStruct      *control;
    ARNameType           schema;
    ARInternalId         *fieldId;
    ARBoolean            reservedIdOK;
    unsigned int         dataType;
    unsigned int         option;
    unsigned int         createMode;
    ARValueStruct        *defaultVal;
    ARPermissionList     *permissions;
    ARFieldLimitStruct   *limit;
    ARDisplayList        *displayList;
    char                 *helpText;
    ARNameType           owner;
    char                 *changeDiary;
    ARStatusList         *status;
```

DESCRIPTION

ARCreateField will create a new field in the indicated schema on the specified server.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
schema	The name of the schema in which the field to create is to be located. The field must be tied to a single, specific schema. This schema must currently exist on the server.

fieldId	The internal ID of the new field within the schema. Remember that the IDs for all fields must be unique within a single schema. The system has a core field range (0 to AR_MAX_CORE_FIELD_ID), a reserved ID range (AR_MAX_CORE_FIELD_ID + 1 to AR_MAX_RESERVED_FIELD_ID), and an open range (AR_MAX_RESERVED_FIELD_ID + 1 and up). Ids in the core range are allocated for the core fields. No ID can be created in the core field range. IDs in the reserved range represent fields that have a registered definition with Remedy Corporation and should be used only for fields that match the rules and criteria for the registered definition. Ids in the open range are available for any use desired by the user. If an ID of 0 is specified for this field, the AR System server will find the next available ID and assign it to the new field. The new ID assigned will be returned in this parameter.
reservedIdOK	A flag indicating whether it is OK to create a field in the reserved ID range. If the ID is in the reserved range, the setting of this flag indicates the action to be taken. If the flag is TRUE, the field will be created. If it is FALSE, the field will NOT be created and an error will be reported. This is to allow an extra warning before a reserved field is created that the field is expected to match the registered definition for the field.
option	The option flag indicating whether the field is a required or optional field. The value is one of **AR_FIELD_OPTION_REQUIRED, AR_FIELD_OPTION_OPTIONAL,** or **AR_FIELD_OPTION_SYSTEM**.
createMode	A flag indicating whether the field is open or protected at create time. An open field is one that any user, whether registered or not with the system, can set during the Submit operation. A protected field is one the user must have been given specific access to in order to set the field during the Submit operation. The value is one of **AR_FIELD_OPEN_AT_CREATE** or **AR_FIELD_PROTECTED_AT_CREATE**.
defaultVal	The value used by the system if a new entry is submitted without a value for this field. If a value is specified, its datatype must match the datatype of the field. Specify NULL for this parameter or a value of AR_DEFAULT_VALUE_NONE to indicate that there is no default value for this field.
permissions	The permissions that have been assigned to this field. The information details which groups have access to the field and what access those groups have. If you assign a blank permissions set, you will receive a warning, AR_WARN_ADMIN_ONLY_ACCESS, indicating that the field can be accessed only by the administrator. The field will be created as specified.
limit	The limits that you want assigned for this field. If limits are assigned, the datatype of the limits must match the datatype of the field. Specify NULL for this parameter or a value of AR_FIELD_LIMIT_NONE to indicate that there are no limits for the field.

displayList The display list is an array of entries, each of which describes how the field should be displayed on the screen. Each of these items has an associated character tag to identify it. This tag can be specified during Export operations to get only the definition for a given tag. Each entry contains the tag, the name for the field, the position of the field, and the type of control to use on the screen.

helpText The help text that is to be associated with the field.

If you do not want any helpText, specify NULL for this parameter.

owner The owner for the field.

If NULL is specified for this parameter, the owner will default to the user performing the operation.

changeDiary The initial change diary that is to be associated with the field. The change diary text can be of any length. When saved, the time of the change and the user who made the change are added to the change diary (called time- and user-stamping).

If no change diary is to be assigned, a NULL pointer should be supplied for this parameter.

RETURN VALUES

fieldID The internal ID of the field within the schema.

status A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL:** Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARDeleteField (3), **ARDisplayList** (3), **ARGetField** (3), **ARGetListField** (3), **ARSetField** (3), **FreeARFieldLimitStruct** (3), **FreeARPermissionList** (3), **FreeARStatusList** (3), **FreeARValueStruct** (3)

ARCreateFilter

NAME

ARCreateFilter – create a new filter in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARCreateFilter (control, name, order, schema, opSet, enable, query, actionList, helpText, owner,
                    changeDiary, status)

    ARControlStruct          *control;
    ARNameType               name;
    unsigned int             order;
    ARNameType               schema;
    unsigned int             opSet;
    unsigned int             enable;
    ARQualifierStruct        *query;
    ARFilterActionList       *actionList;
    char                     *helpText;
    ARNameType               owner;
    char                     *changeDiary;
    ARStatusList             *status;
```

DESCRIPTION

ARCreateFilter will create a new filter with the indicated name on the specified server. The filter will take effect immediately and will remain in effect until changed or deleted.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the filter to create. The names of all filters on a given server must be unique.
order	The order of the new filter. The filter order is a code between 0 and 1000, inclusive. It allows the ordering of filters so that filters with lower orders are executed before filters with higher orders. So, the filter order allows you to specify the order in which filters will be processed and lets you insure that given filters will be performed in the order you desire.
schema	The name of the schema the filter is linked to. The filter *must* be tied to a single, specific schema. This schema must currently exist on the server.

opSet — A bit mask of the operations for which this filter applies. This field consists of one or more of the following values OR'ed together. **AR_OPERATION_GET**, **AR_OPERATION_SET**, **AR_OPERATION_CREATE**, **AR_OPERATION_DELETE**, and **AR_OPERATION_MERGE**.

enable — A flag with a setting of 0 to indicate that this filter is to be marked as disabled so it will not be executed or 1 to indicate that the filter is active and available for use. A filter that is disabled will not be checked for match during any operation and will not fire.

query — A qualification that is used to test the operation/record being accessed. The operation/values of fields for the record must match this qualification to trigger the filter.

If there is no qualifying condition, specify NULL or assign a value of AR_COND_OP_NONE to this value.

actionList — The set of one or more actions to take when the filter conditions are met. Every filter MUST have at least one action and may have as many as AR_MAX_ACTIONS actions.

helpText — The help text that is to be associated with the filter. The help text can be of any length.

If no help text is to be assigned, a NULL pointer should be supplied for this parameter.

owner — The owner for the filter.

If NULL is specified for this parameter, the owner will default to the user performing the operation.

changeDiary — The initial change diary that is to be associated with the filter. The change diary text can be of any length. When saved, the time of the change and the user who made the change are added to the change diary (called time- and user-stamping).

If no change diary is to be assigned, a NULL pointer should be supplied for this parameter.

RETURN VALUES

status — A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error

- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARDeleteFilter (3), **ARGetFilter** (3), **ARGetListFilter** (3), **ARSetFilter** (3), **FreeARFilterActionList** (3), **FreeARQualifierStruct** (3), **FreeARStatusList** (3)

ARCreateSchema

NAME

ARCreateSchema – create a new schema in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARCreateSchema (control, name, groupList, adminGroupList, getListFields, indexList, helpText, owner,
                    changeDiary, status)
      ARControlStruct          *control;
      ARNameType               name;
      ARInternalIdList         *groupList;
      ARInternalIdList         *adminGroupList;
      AREntryListFieldList     *getListFields;
      ARIndexList              *indexList;
      char                     *helpText;
      ARNameType               owner;
      char                     *changeDiary;
      ARStatusList             *status;
```

DESCRIPTION

ARCreateSchema will create a new schema with the indicated name on the specified server. The schema created will contain the “core” fields.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the schema to create. The names of all schemas on a given server must be unique.

groupList
The groupList definition for the schema. This list of 0 or more groups defines the list of groups whose users are allowed to access this schema.

Defining a groupList with 0 items will define a schema that can be accessed only by users with Administrator capability. Defining a groupList that contains the "Public" group (group id 0) will define a schema that can be accessed by all users.

adminGroupList
The list of groups whose users have potential Sub-Administrator access to this schema. This is the list of groups whose members will have sub-administration rights to the schema and its associated filters, escalations, and active links if the user is also a member of the SubAdministrator group.

Defining an adminGroupList with 0 items will define a schema that can be administered only by users with Administrator capability. Defining an adminGroupList that contains the "Public" group (group id 0) will define a schema that can be administered by all users who are members of the SubAdministrator group.

getListFields
The getListFields definition for the schema. This list of 0 or more fields and formatting information defines the description that will be returned with the ARGetListEntry call. The maximum size of the fields and separators must be less than or equal to AR_MAX_SDESC_SIZE.

Specifying NULL for this parameter or defining a getListFields definition with 0 items specifies that the description will be derived from the Short-Description core field (field id is 8).

indexList
The indexList definition for the schema. This list of 0 or more indexes defines the indexes created in the database on the schema. Indexes can be specified on a single or on multiple columns. You cannot index any diary field or character field with a maximum length over 255 bytes.

helpText
The help text that is to be associated with the schema. The help text can be of any length.

If no help text is to be assigned, a NULL pointer should be supplied for this parameter.

owner
The owner for the schema.

If NULL is specified for this parameter, the owner will default to the user performing the operation.

changeDiary
The initial change diary that is to be associated with the schema. The change diary text can be of any length. When saved, the time of the change and the user who made the change are added to the change diary (called time- and user-stamping).

If no change diary is to be assigned, a NULL pointer should be supplied for this parameter.

RETURN VALUES

status
A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateField (3), **ARDeleteSchema** (3), **ARGetSchema** (3), **ARGetListSchema** (3), **ARSetField** (3), **ARSetSchema** (3), **FreeAREntryListFieldList** (3), **FreeARIndexList** (3), **FreeARInternalIdList** (3), **FreeARStatusList** (3)

ARDecodeDiary

NAME

ARDecodeDiary – decode a formatted diary string into an array

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARDecodeDiary (diaryString, diaryList, status)
    char                    *diaryString;
    ARDiaryList             *diaryList;
    ARStatusList            *status;
```

DESCRIPTION

ARDecodeDiary takes a formatted diary string as returned for a diary field from the **ARGetEntry** call or for the change diary record from any structure and decodes it. The resulting timestamp, user name, and value sets are returned in an array.

INPUT ARGUMENTS

diaryString	The formatted diary value that is returned by **ARGetEntry** or one of the structure get calls.

RETURN VALUES

diaryList	Array structure that holds the decoded diary value.
status	A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later. The return of the function itself is an integer indicating the success of the call. The return will be one of the following values: n **AR_RETURN_OK**: Operation successful n **AR_RETURN_WARNING**: Warning during process but operation completed successfully n **AR_RETURN_ERROR**: Operation failed due to error n **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information n **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARDecodeStatusHistory (3), **ARGetActiveLink** (3), **ARGetAdminExtension** (3), **ARGetCharMenu** (3), **ARGetEntry** (3), **ARGetEscalation** (3), **ARGetField** (3), **ARGetFilter** (3), **ARGetSchema** (3), **FreeARDiaryList** (3), **FreeARStatusList** (3)

ARDecodeStatusHistory

NAME

ARDecodeStatusHistory – decode a formatted status history string into an array

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARDecodeStatusHistory (statHistString, statHistList, status)
        char                    *statHistString;
        ARStatusHistoryList     *statHistList;
        ARStatusList            *status;
```

DESCRIPTION

ARDecodeStatusHistory takes a formatted status history string as returned for the field AR_CORE_STATUS_HISTORY from the **ARGetEntry** call and decodes it. The resulting timestamp and user name values for each status are returned in an array.

INPUT ARGUMENTS

statHistString	The formatted status history value that is returned by the **ARGetEntry** call.

RETURN VALUES

statHistList	Array structure that holds the decoded status history value.
status	A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later. The return of the function itself is an integer indicating the success of the call. The return will be one of the following values: n **AR_RETURN_OK**: Operation successful n **AR_RETURN_WARNING**: Warning during process but operation completed successfully n **AR_RETURN_ERROR**: Operation failed due to error n **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information n **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARDecodeDiary (3), **ARGetEntry** (3), **FreeARStatusList** (3), **FreeARStatusHistoryList** (3)

ARDeleteActiveLink

NAME

ARDeleteActiveLink – delete an existing active link from the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARDeleteActiveLink (control, name, status)
    ARControlStruct          *control;
    ARNameType               name;
    ARStatusList             *status;
```

DESCRIPTION

ARDeleteActiveLink will delete an existing active link with the indicated name from the specified server. The deleted active link is immediately removed and will no longer be returned to users who request information about active links.

Since the operation of active links is on clients accessing the server, the active link will still be available on individual clients until the client reloads the configuration from the server.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the active link to delete.

RETURN VALUES

status	A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later. The return of the function itself is an integer indicating the success of the call. The return will be one of the following values: n **AR_RETURN_OK**: Operation successful n **AR_RETURN_WARNING**: Warning during process but operation completed successfully n **AR_RETURN_ERROR**: Operation failed due to error n **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information n **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateActiveLink (3), **ARDeleteSchema** (3), **ARGetActiveLink** (3), **ARGetListActiveLink** (3), **ARSetActiveLink** (3), **FreeARStatusList** (3)

ARDeleteAdminExtension

NAME

ARDeleteAdminExtension – delete an existing administrator extension from the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARDeleteAdminExtension (control, name, status)
    ARControlStruct     *control;
    ARNameType          name;
    ARStatusList        *status;
```

DESCRIPTION

ARDeleteAdminExtension will delete an existing administrator extension with the indicated name from the specified server.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control — The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

name — The name of the extension to delete.

RETURN VALUES

status — A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING:** Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateAdminExtension (3), **ARExecuteAdminExtension** (3), **ARGetAdminExtension** (3), **ARGetListAdminExtension** (3), **ARSetAdminExtension** (3), **FreeARStatusList** (3)

ARDeleteCharMenu

NAME

ARDeleteCharMenu – delete an existing character menu from the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARDeleteCharMenu (control, name, status)
    ARControlStruct          *control;
    ARNameType               name;
    ARStatusList             *status;
```

DESCRIPTION

ARDeleteCharMenu will delete an existing character menu with the indicated name from the specified server. The deleted character menu is immediately removed and will no longer be returned to users who request information about character menus. Since the operation of character menus is on clients accessing the server, the character menu will still be available on individual clients until the client refreshes the character menu definition (controlled by the refresh code).

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the character menu to delete.

RETURN VALUES

status	A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later. The return of the function itself is an integer indicating the success of the call. The return will be one of the following values: n **AR_RETURN_OK**: Operation successful

- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateCharMenu (3), **ARExpandCharMenu** (3), **ARGetCharMenu** (3), **ARGetListCharMenu** (3), **ARSetCharMenu** (3), **FreeARStatusList** (3)

ARDeleteEntry

NAME

ARDeleteEntry – delete an entry from the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARDeleteEntry (control, schema, entryId, status)
    ARControlStruct     *control;
    ARNameType          schema;
    AREntryIdType       entryId;
    ARStatusList        *status;
```

DESCRIPTION

ARDeleteEntry will add a delete an entry from the specified schema. This operation can only be performed by users who have Administrator access to the schema.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
schema	Identifies the schema from which the entry is to be deleted.
entryId	Identifies the specific entry within the schema.

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateEntry (3), **ARGetEntry** (3), **ARGetEntryStatistics** (3), **ARGetListEntry** (3), **ARMergeEntry** (3), **ARSetEntry** (3), **FreeARStatusList** (3)

ARDeleteEscalation

NAME

ARDeleteEscalation – delete an existing escalation from the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARDeleteEscalation (control, name, status)
    ARControlStruct          *control;
    ARNameType               name;
    ARStatusList             *status;
```

DESCRIPTION

ARDeleteEscalation will delete an existing escalation with the indicated name from the specified server. The deleted escalation is immediately removed and all processing associated with it is no longer performed.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control — The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

name — The name of the escalation to delete.

RETURN VALUES

status — A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateEscalation (3), **ARDeleteSchema** (3), **ARGetEscalation** (3), **ARGetListEscalation** (3), **ARSetEscalation** (3), **FreeARStatusList** (3)

5

ARDeleteField

NAME

ARDeleteField – delete an existing field from an AR System schema

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARDeleteField (control, schema, fieldId, deleteData, status)
    ARControlStruct     *control;
    ARNameType          schema;
    ARInternalId        fieldId;
    ARBoolean           deleteData;
    ARStatusList        *status;
```

DESCRIPTION

ARDeleteField will delete an existing field with the indicated id from the specified schema on the specified server. The delete operation will delete the field along with any data contained within that field (see the deleteData flag below).

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
schema	The name of the schema containing the field to be deleted.
fieldId	The id of the specific field to delete.
deleteData	A flag controlling the operation of the delete. Before the delete operation is performed, the system checks to see if there are any data entries for the field. If not, the delete operation is performed. If there is data, this flag is checked. If it is TRUE, the field is deleted; otherwise, the delete operation is NOT performed and a notice is returned to the user.

RETURN VALUES

status	A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later. The return of the function itself is an integer indicating the success of the call. The return will be one of the following values: n **AR_RETURN_OK**: Operation successful

- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateField (3), **ARDeleteSchema** (3), **ARGetField** (3), **ARGetListField** (3), **ARSetField** (3), **FreeARStatusList** (3)

ARDeleteFilter

NAME

ARDeleteFilter – delete an existing filter from the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARDeleteFilter (control, name, status)
    ARControlStruct          *control;
    ARNameType               name;
    ARStatusList             *status;
```

DESCRIPTION

ARDeleteFilter will delete an existing filter with the indicated name from the specified server. The deleted filter is immediately removed and all processing associated with it is no longer performed.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the filter to delete.

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateFilter (3), **ARDeleteSchema** (3), **ARGetFilter** (3), **ARGetListFilter** (3), **ARSetFilter** (3), **FreeARStatusList** (3)

ARDeleteSchema

NAME

ARDeleteSchema – delete an existing schema from the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARDeleteSchema (control, name, deleteEntries, status)
    ARControlStruct     *control;
    ARNameType          name;
    ARBoolean           deleteEntries;
    ARStatusList        *status;
```

DESCRIPTION

ARDeleteSchema will delete an existing schema with the indicated name from the specified server. The delete operation will delete the schema, all associated fields, and any active links, filters, and escalations that are linked to this schema. In addition, the delete operation removes any data contents for this schema from the server (see the deleteEntries flag below).

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the schema to delete.
deleteEntries	A flag controlling the operation of the delete. Before the delete operation is performed, the system checks to see if there are any data entries for the schema. If not, the delete operation is performed. If there are entries, this flag is checked. If it is TRUE, the schema is deleted; otherwise, the delete operation is NOT performed and a notice is returned to the user.

RETURN VALUES

status	A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later. The return of the function itself is an integer indicating the success of the call. The return will be one of the following values: n **AR_RETURN_OK**: Operation successful n **AR_RETURN_WARNING**: Warning during process but operation completed successfully n **AR_RETURN_ERROR**: Operation failed due to error n **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information n **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateSchema (3), **ARGetSchema** (3), **ARGetListSchema** (3), **ARSetSchema** (3), **FreeARStatusList** (3)

5

ARExecuteAdminExtension

NAME

ARExecuteAdminExtension – execute an administrator extension on the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARExecuteAdminExtension (control, name, status)
    ARControlStruct          *control;
    ARNameType               name;
    ARStatusList             *status;
```

DESCRIPTION

ARExecuteAdminExtension will perform the administrator extension with the indicated name on the specified server.

Execute permission on administrator extensions is controlled through the security scheme of the AR System. The user identified in the control record is used to determine which extensions are accessible to the requestor. Only extensions that are accessible to the user can be executed.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the extension to execute.

RETURN VALUES

status	A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateAdminExtension (3), **ARDeleteAdminExtension** (3), **ARGetAdminExtension** (3), **ARGetListAdminExtension** (3), **ARSetAdminExtension** (3), **FreeARStatusList** (3)

ARExpandCharMenu

NAME

ARExpandCharMenu – expand the passed menu definition by resolving all query and file references

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARExpandCharMenu (control, menuDefn, menu, status)
        ARControlStruct         *control;
        ARCharMenuStruct        *menuDefn;
        ARCharMenuStruct        *menu;
        ARStatusList            *status;
```

DESCRIPTION

ARExpandCharMenu takes the specified menu definition and expands any query and file references within the menu. The resulting menu contains only items using the “list” style of menu.

This operation is available to all users. If there is a query style menu that must be expanded, the query operation is controlled by the access rights of the user.

INPUT ARGUMENTS

control — The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

menuDefn The definition of the character menu that is to be expanded.

RETURN VALUES

menu The expanded menu definition. The expanded definition will use only the “list” branch of the structure.

status A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateCharMenu (3), **ARDeleteCharMenu** (3), **ARGetCharMenu** (3), **ARGetListCharMenu** (3), **ARSetCharMenu** (3), **FreeARCharMenuStruct** (3), **FreeARStatusList** (3)

ARExport

NAME

ARExport – export existing schema, mail template, filter, escalation, active link, menu, and admin extension definitions

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARExport (control, structItems, displayTag, exportBuf, status)
    ARControlStruct      *control;
    ARStructItemList     *structItems;
    ARNameType           displayTag;
    char                 **exportBuf;
    ARStatusList         *status;
```

DESCRIPTION

ARExport will export one or more structure definitions (schemas, escalations, filters, active links, character menus, and administrator extensions) from the AR System. This allows the copying of definitions from one server to another.

The export of filters, escalations, administrator extensions, and full detailed schema and active link definitions can be performed only by users who have Administrator capabilities within the AR System. Other users can export schema, active link, and admin extension definitions without sensitive information like permissions if they have permission to access the structure.

INPUT ARGUMENTS

control

The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

structItems

An array of the items to export. The type and name of the items to export are identified. Each item to be exported must be indicated.

The codes for the types of exports allowed are as follows:

- **AR_STRUCT_ITEM_SCHEMA**: Full schema definition
- **AR_STRUCT_ITEM_SCHEMA_DEFN**: Partial schema definition containing field database definition
- **AR_STRUCT_ITEM_SCHEMA_VIEW**: Partial schema definition containing field display definition
- **AR_STRUCT_ITEM_SCHEMA_MAIL**: Mail template format of schema definition
- **AR_STRUCT_ITEM_FILTER**: Filter definition

- **AR_STRUCT_ITEM_ACTIVE_LINK**: Active link definition
- **AR_STRUCT_ITEM_ADMIN_EXT**: Administrator extension definition
- **AR_STRUCT_ITEM_CHAR_MENU**: Character menu definition
- **AR_STRUCT_ITEM_ESCALATION**: Escalation definition

displayTag Tag that identifies which specific display struct view to export when exporting a schema using AR_STRUCT_ITEM_SCHEMA_VIEW or an active link definition. If there is a display that matches the tag, only that display structure is exported. If there is not one that matches, the display that is listed first in the displayList is exported. If an empty string or NULL is specified, only the first item in the displayList is exported for each field and active link.

RETURN VALUES

exportBuf malloced buffer containing the text of the items exported. The buffer will contain information for all of the items requested (or at least for all the items without an error).

Note that if a schema is to be later imported, the full definition format of the schema must be exported. The other formats are used for caching of definition and/or view information and are not the complete definitions needed.

status A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARImport (3), **FreeARStatusList** (3), **FreeARStructItemList** (3)

ARGetActiveLink

NAME

ARGetActiveLink – retrieve information about an active link in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetActiveLink (control, name, order, schema, groupList, executeMask, field, displayList, enable, query,
                     actionList, helpText, timestamp, owner, lastChanged, changeDiary, status)
    ARControlStruct          *control;
    ARNameType               name;
    unsigned int             *order
    ARNameType               schema;
    ARInternalIdList         *groupList;
    unsigned int             *executeMask;
    ARInternalId             *field;
    ARDisplayList            *displayList;
    unsigned int             *enable;
    ARQualifierStruct        *query;
    ARActiveLinkActionList   *actionList;
    char                     **helpText;
    ARTimestamp              *timestamp;
    ARNameType               owner;
    ARNameType               lastChanged;
    char                     **changeDiary;
    ARStatusList             *status;
```

DESCRIPTION

ARGetActiveLink will retrieve information about an active link with the indicated name on the specified server.

Access to active link information is restricted to users who have been granted access to the active link. If you are granted permission to the active link, you can see all the data except groupList information which is limited to users with Administrator capability. If you are not granted permission, you cannot retrieve any information about the active link.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the active link to retrieve.

RETURN VALUES

order	The order of the active link. The active link order is a code between 0 and 1000, inclusive. It allows the ordering of active links so that active links with lower orders are executed before active links with higher orders. So, the active link order allows you to specify the order in which active links will be processed and lets you insure that given active links will be performed in the order you desire. If you do not want the order returned, specify NULL for this parameter.
schema	The name of the schema the active link is tied to. If you do not want the schema returned, specify NULL for this parameter.
groupList	A list of 0 or more groups identifying groups that have permission to access this active link. Only a user with Administrator capability can retrieve the groupList information. Any other user will receive an empty list along with a warning, AR_WARN_PERM_REQUIRES_ADMIN, indicating that no groupList information was returned because the user did not have Administrator capability. If you do not want the group list returned, specify NULL for this parameter.
executeMask	A bit mask of the conditions under which this active link will be executed. This field consists of one or more of the following values OR'ed together: **AR_EXECUTE_ON_BUTTON**, **AR_EXECUTE_ON_RETURN**, **AR_EXECUTE_ON_SUBMIT**, **AR_EXECUTE_ON_MODIFY**, **AR_EXECUTE_ON_DISPLAY**, **AR_EXECUTE_ON_MENU_CHOICE**, and **AR_EXECUTE_ON_SET_DEFAULT**. If you do not want the execute mask returned, specify NULL for this parameter.
field	The ID of the field the active link is tied to if the execute mask includes the AR_EXECUTE_ON_RETURN or AR_EXECUTE_ON_MENU_CHOICE flag. If not included, the return for this field is a 0. If you do not want the field returned, specify NULL for this parameter.
displayList	The display information for the button to be displayed on the user's view if the execute mask includes the AR_EXECUTE_ON_BUTTON flag. If not included, the label is set to blank and the positions to 0s. If you do not want the display information returned, specify NULL for this parameter.
enable	A flag set to 0 if this active link is disabled and to 1 if it is enabled. If you do not want the enable flag returned, specify NULL for this parameter.
query	A qualification that is to be performed when the active link is executed. It will allow conditional execution of active links or a different operation to be performed depending on the conditions on the window where the active link is being executed. If there is no qualifying condition, a value of AR_COND_OP_NONE is returned for this value. If you do not want the query returned, specify NULL for this parameter.

actionList
The set of one or more actions to take when the active link is executed. Every active link MUST have at least one action and can have up to AR_MAX_ACTIONS.

If you do not want the action list returned, specify NULL for this parameter.

helpText
The help text that is associated with the active link.

If you do not want the help text returned, specify NULL for this parameter. This is useful when you will not be using the help text information as you are calling this routine to verify the existence of an active link. By not retrieving the help text, the operation is more efficient and takes less time and space.

timestamp
The timestamp indicating when this active link was last changed.

If you do not want the timestamp returned, specify NULL for this parameter.

owner
The user who is defined as the owner of this active link definition.

If you do not want the owner returned, specify NULL for this parameter.

lastChanged
The user who last changed this active link definition in some way.

If you do not want the lastChanged user returned, specify NULL for this parameter.

changeDiary
The change diary that is associated with the active link. Space is allocated for the text and a pointer to that space returned. If there is no change diary for the active link, a NULL pointer is returned.

The text returned is a formatted diary string. Use the ARDecodeDiary call to decode this string into an array of timestamp, user name, and text string pieces.

If you do not want the change diary returned, specify NULL for this parameter. This is useful when you will not be using the change diary information as you are calling this routine to verify the existence of an active link. By not retrieving the change diary, the operation is more efficient and takes less time and space.

status
A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateActiveLink (3), **ARDecodeDiary** (3), **ARDeleteActiveLink** (3), **ARGetListActiveLink** (3), **ARSetActiveLink** (3), **FreeARActiveLinkActionList** (3), **FreeARDisplayList** (3), **FreeARInternalIdList** (3), **FreeARQualifierStruct** (3), **FreeARStatusList** (3)

ARGetAdminExtension

NAME

ARGetAdminExtension – retrieve information about an administrator extension in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetAdminExtension (control, name, groupList, command, helpText, timestamp, owner, lastChanged,
        changeDiary, status)
    ARControlStruct     *control;
    ARNameType          name;
    ARInternalIdList    *groupList;
    char                *command;
    char                **helpText;
    ARTimestamp         *timestamp;
    ARNameType          owner;
    ARNameType          lastChanged;
    char                **changeDiary;
    ARStatusList        *status;
```

DESCRIPTION

ARGetAdminExtension will retrieve information about the administrator extension indicated by name on the specified server.

Access to administrator extension information is restricted to users who have been granted access to the administrator extension. If you are granted permission to the administrator extension, you can see all the data except groupList information which is limited to users with Administrator capability. If you are not granted permission, you cannot retrieve any information about the administrator extension.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the extension about which information is being requested.

RETURN VALUES

groupList	A list of 0 or more groups identifying groups that have permission to access this extension. Only a user with Administrator capability can retrieve the groupList information. Any other user will receive an empty list along with a warning, AR_WARN_PERM_REQUIRES_ADMIN, indicating that no groupList information was returned because the user did not have Administrator capability. If you do not want the group list returned, specify NULL for this parameter.
command	The command that represents the extension. It can be anything from a simple shell command to running a process with a variety of parameters. If you do not want the command returned, specify NULL for this parameter.
helpText	The help text that is associated with the extension. Space is allocated for the text and a pointer to that space returned. If there is no help text for the extension, a NULL pointer is returned. If you do not want the help text returned, specify NULL for this parameter. This is useful when you will not be using the help text information as you are calling this routine to verify the existence of a extension. By not retrieving the help text, the operation is more efficient and takes less time and space.
timestamp	Timestamp indicating when this extension was last changed. If you do not want the timestamp returned, specify NULL for this parameter.
owner	The user who is defined as the owner of this extension definition. If you do not want the owner returned, specify NULL for this parameter.
lastChanged	The user who last changed this extension definition in some way. If you do not want the lastChanged user returned, specify NULL for this parameter.
changeDiary	The change diary that is associated with the extension. Space is allocated for the text and a pointer to that space returned. If there is no change diary for the extension, a NULL pointer is returned. The text returned is a formatted diary string. Use the ARDecodeDiary call to decode this string into an array of timestamp, user name, and text string pieces. If you do not want the change diary returned, specify NULL for this parameter. This is useful when you will not be using the change diary information as you are calling this routine to verify the existence of an extension. By not retrieving the change diary, the operation is more efficient and takes less time and space.
status	A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later. The return of the function itself is an integer indicating the success of the call. The return will be one of the following values: n **AR_RETURN_OK**: Operation successful

- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateAdminExtension (3), **ARDecodeDiary** (3), **ARDeleteAdminExtension** (3), **ARExecuteAdminExtension** (3), **ARGetListAdminExtension** (3), **ARSetAdminExtension** (3), **FreeARInternalIdList** (3), **FreeARStatusList** (3)

ARGetCharMenu

NAME

ARGetCharMenu – retrieve information about a character menu definition in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetCharMenu (control, name, refreshCode, menuDefn, helpText, timestamp, owner, lastChanged,
      changeDiary, status)
ARControlStruct          *control;
ARNameType               name;
unsigned int             *refreshCode;
ARCharMenuStruct         *menuDefn;
char                     **helpText;
ARTimestamp              *timestamp;
ARNameType               owner;
ARNameType               lastChanged;
char                     **changeDiary;
ARStatusList             *status;
```

DESCRIPTION

ARGetCharMenu will retrieve information about the character menu indicated by name on the specified server.

Access to character menu information is available to all users.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the character menu to retrieve.

RETURN VALUES

refreshCode	The code indicating the refresh frequency for the character menu when it is being used. If you do not want the refresh code returned, specify NULL for this parameter.
menuDefn	The definition of the character menu. If you do not want the menu definition returned, specify NULL for this parameter.
helpText	The help text that is associated with the character menu. If you do not want the help text returned, specify NULL for this parameter. This is useful when you will not be using the help text information as you are calling this routine to verify the existence of a character menu. By not retrieving the help text, the operation is more efficient and takes less time and space.
timestamp	The timestamp indicating when this character menu was last changed. If you do not want the timestamp returned, specify NULL for this parameter.
owner	The user who is defined as the owner of this character menu definition. If you do not want the owner returned, specify NULL for this parameter.
lastChanged	The user who last changed this character menu definition in some way. If you do not want the lastChanged user returned, specify NULL for this parameter.
changeDiary	The change diary that is associated with the character menu. Space is allocated for the text and a pointer to that space returned. If there is no change diary for the character menu, a NULL pointer is returned. The text returned is a formatted diary string. Use the ARDecodeDiary call to decode this string into an array of timestamp, user name, and text string pieces. If you do not want the change diary returned, specify NULL for this parameter. This is useful when you will not be using the change diary information as you are calling this routine to verify the existence of a character menu. By not retrieving the change diary, the operation is more efficient and takes less time and space.
status	A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later. The return of the function itself is an integer indicating the success of the call. The return will be one of the following values: n **AR_RETURN_OK**: Operation successful

- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateCharMenu (3), **ARDecodeDiary** (3), **ARDeleteCharMenu** (3), **ARExpandCharMenu** (3), **ARGetListCharMenu** (3), **ARSetCharMenu** (3), **FreeARCharMenuStruct** (3), **FreeARStatusList** (3)

ARGetEntry

NAME

ARGetEntry – retrieve details of an entry in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetEntry (control, schema, entryId, idList, fieldList, status)
    ARControlStruct      *control;
    ARNameType           schema;
    AREntryIdType        entryId;
    ARInternalIdList     *idList;
    ARFieldValueList     *fieldList;
    ARStatusList         *status;
```

DESCRIPTION

ARGetEntry will retrieve information about the entry indicated by the entry ID in the specified schema. You can request the values for a specific list of fields, for all fields that are accessible, or to no fields to just verify the existence of an entry.

Access to entries is controlled through the security scheme of the AR System. The user identified in the control record is used to determine which fields are accessible to the requestor. If a value is requested for a field that the user does not have access to, no value will be returned for that field in the result and a *warning* message will be returned in the status list. Each value is checked for accessibility, with either the value or an error returned.

INPUT ARGUMENTS

control — The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

schema Identifies the schema that contains the desired entry.

entryId Identifies the specific entry within the schema whose values are to be retrieved.

idList A list of field IDs identifying the fields for which values are desired. If you want only a few specific fields, it is more efficient to specify just those fields rather than retrieve all values and then just use the ones you want.

To retrieve ALL fields (subject to security constraints), specify a NULL pointer for this parameter or specify this list with the numItems field set to 0.

To perform a quick check of existence, specify a NULL pointer for this parameter and NULL for the **fieldList** parameter.

RETURN VALUES

fieldList A list of field/value pairs for all the fields requested. The items in the list will be in the same order as the IDs listed in the idList (if a specific list was given). For any field ID that doesn't exist in the schema or for which you do not have access, the field ID at the corresponding position will be 0 and the value NULL. Appropriate errors/warnings will be included in the status return.

To check existence of the item without retrieving any fields, pass a NULL pointer for this parameter.

status A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateEntry (3), **ARDecodeDiary** (3), **ARDecodeStatusHistory** (3), **ARDeleteEntry** (3), **ARGetEntryStatistics** (3), **ARGetListEntry** (3), **ARSetEntry** (3), **FreeARInternalIdList** (3), **FreeARFieldValueList** (3), **FreeARStatusList** (3)

5

ARGetEntryStatistics

NAME

ARGetEntryStatistics – compute a statistic on data meeting a specified condition

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetEntryStatistics (control, schema, qualifier, target, statistic, results, status)
    ARControlStruct              *control;
    ARNameType                   schema;
    ARQualifierStruct            *qualifier;
    ARFieldValueOrArithStruct    *target;
    unsigned int                 statistic;
    ARValueList                  *results;
    ARStatusList                 *status;
```

DESCRIPTION

ARGetEntryStatistics will compute a statistical result on data matching a specified set of criteria. It will retrieve a statistic computed across all of all the entries in the schema that meet the specified criteria and are accessible by the user.

The access control scheme of the system controls which entries can be searched. If you have access to the entry ID field of an entry, the entry is included in your search set and will be included by this call. If you don't, the item is not in the search set and will never be included by this call.

INPUT ARGUMENTS

control — The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

schema — Identifies the schema that is to be searched.

qualifier — A structure identifying the condition(s) to be searched on. Any field that is accessible to the user can be qualified. Qualifications include conditional operations (AND, OR, NOT), relational operations (=, !=, >, >=, <, <=, and a pattern matching LIKE operator), and arithmetic operations on numeric types. The condition can apply to a single field or to multiple fields.

You must have read access to all the fields being qualified. If a field does not exist or you do not have read access to a field, an error is returned and the operation is not performed.

To select ALL entries, supply either a NULL or a structure with the operation field set to AR_COND_OP_NONE.

target — A structure identifying the arithmetic operation to be used for computing the statistic. The structure can be as simple as a single field or as complex as desired within the arithmetic operations supported by the system.

You must have read access to all the fields on which you are computing the statistic result. If you do not have read access to a field, the operation will be rejected.

If the operation being performed is a COUNT operation, this field is optional. You can omit it by setting the tag field to 0. For all other operations, this field must be supplied to identify what field(s) you are computing the statistic on.

statistic — The statistic to perform. It must be from the set **AR_STAT_OP_COUNT, AR_STAT_OP_SUM, AR_STAT_OP_AVERAGE, AR_STAT_OP_MINIMUM,** and **AR_STAT_OP_MAXIMUM.**

RETURN VALUES

results — The result from the statistic operation.

status — A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateEntry (3), **ARDeleteEntry** (3), **ARGetEntry** (3), **ARGetListEntry (3), ARLoadARQualifierStruct (3), ARMergeEntry** (3), **ARSetEntry** (3), **FreeARFieldValueOrArithStruct** (3), **FreeARQualifierStruct** (3), **FreeARStatusList** (3), **FreeARValueList** (3)

5

ARGetEscalation

NAME

ARGetEscalation – retrieve information about an escalation in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetEscalation (control, name, escalationTm, schema, enable, query, actionList, helpText, timestamp,
        owner, lastChanged, changeDiary, status)
    ARControlStruct          *control;
    ARNameType               name;
    AREscalationTmStruct     *escalationTm;
    ARNameType               schema;
    unsigned int             *enable;
    ARQualifierStruct        *query;
    ARFilterActionList       *actionList;
    char                     **helpText;
    ARTimestamp              *timestamp;
    ARNameType               owner;
    ARNameType               lastChanged;
    char                     **changeDiary;
    ARStatusList             *status;
```

DESCRIPTION

ARGetEscalation will retrieve information about an escalation with the indicated name on the specified server.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the escalation to retrieve.

RETURN VALUES

escalationTm	The checking time of the escalation. It defines a datetime or frequency of AR System checks for escalation condition. Escalation time has two formats, one is in seconds as a time interval between checks, the other is a datetime mask specifying on what day of the month or week at what hour and minute of the day the AR System checks the escalation condition. If you do not want the escalationTm returned, specify NULL for this parameter.

schema	The name of the schema the escalation is tied to. If you do not want the schema returned, specify NULL for this parameter.
enable	A flag set to 0 if this escalation is disabled and to 1 if it is enabled. If you do not want the enable flag returned, specify NULL for this parameter.
query	A qualification that is used to search the specified schema. Any records that match the qualification will have the escalation action performed on them. If there is no qualifying condition, a value of AR_COND_OP_NONE is returned for this value. If you do not want the query returned, specify NULL for this parameter.
actionList	The set of one or more actions to take when the escalation conditions are met. Every escalation MUST have at least one action and can have up to AR_MAX_ACTIONS. If you do not want the action list returned, specify NULL for this parameter.
helpText	The help text that is associated with the escalation. If you do not want the help text returned, specify NULL for this parameter. This is useful when you will not be using the help text information as you are calling this routine to verify the existence of a escalation. By not retrieving the help text, the operation is more efficient and takes less time and space.
timestamp	The timestamp indicating when this escalation was last changed. If you do not want the timestamp returned, specify NULL for this parameter.
owner	The user who is defined as the owner of this escalation definition. If you do not want the owner returned, specify NULL for this parameter.
lastChanged	The user who last changed this escalation definition in some way. If you do not want the lastChanged user returned, specify NULL for this parameter.
changeDiary	The change diary that is associated with the escalation. Space is allocated for the text and a pointer to that space returned. If there is no change diary for the escalation, a NULL pointer is returned. The text returned is a formatted diary string. Use the ARDecodeDiary call to decode this string into an array of timestamp, user name, and text string pieces. If you do not want the change diary returned, specify NULL for this parameter. This is useful when you will not be using the change diary information as you are calling this routine to verify the existence of a escalation. By not retrieving the change diary, the operation is more efficient and takes less time and space.
status	A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later. The return of the function itself is an integer indicating the success of the call. The return will be one of the following values: n **AR_RETURN_OK**: Operation successful

- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateEscalation (3), **ARDecodeDiary** (3), **ARDeleteEscalation** (3), **ARGetListEscalation** (3), **ARSetEscalation** (3), **FreeARQualifierStruct** (3), **FreeARFilterActionList** (3), **FreeARStatusList** (3)

ARGetField

NAME

ARGetField – retrieve information about a field in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetField (control, schema, fieldId, dataType, option, createMode, defaultVal, permissions, limit,
                displayList, helpText, timestamp, owner, lastChanged, changeDiary, status)

ARControlStruct      *control;
ARNameType           schema;
ARInternalId         fieldId;
unsigned int         *dataType;
unsigned int         *option;
unsigned int         *createMode;
ARValueStruct        *defaultVal;
ARPermissionList     *permissions;
ARFieldLimitStruct   *limit;
ARDisplayList        *displayList;
char                 **helpText;
ARTimestamp          *timestamp;
ARNameType           owner;
ARNameType           lastChanged;
char                 **changeDiary;
ARStatusList         *status;
```

DESCRIPTION

ARGetField will retrieve information about the specified field in the indicated schema on the specified server. Information specifically about the individual field is retrieved.

Access to field information is restricted to users who have been granted access to the schema which holds the field. If you are granted permission to the schema, you can see all the data except for the permission information which is limited to users with Administrator capability. If you are not granted permission, you cannot retrieve any information about the field.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
schema	The name of the schema containing the field about which information is being requested.
fieldId	The internal ID of the specific field within the schema to get information about.

RETURN VALUES

dataType	The datatype of the field. The code (as defined in ar.h) indicating the base datatype of the field. If you do not want the datatype returned, specify NULL for this parameter.
option	The option flag indicating whether the field is a required or optional field. If you do not want the option flag returned, specify NULL for this parameter.
createMode	A flag indicating whether the field is open or protected at create time. An open field is one any user, whether having write access or not, can set during the Submit operation. A protected field is one the user must have been given specific write access to in order to set the field during the Submit operation. If you do not want the create mode returned, specify NULL for this parameter.
defaultVal	The value to use for this field if a new entry is submitted without a value for this field. If there is no default value for the field, the value returned will have a tag of AR_DEFAULT_VALUE_NONE. If you do not want the default value returned, specify NULL for this parameter.
permissions	The permissions that have been assigned to this field. The information details which groups have access to the field and what access those groups have. Only a user with Administrator capability can retrieve the permission information. Any other user will receive an empty list along with a warning, AR_WARN_PERM_REQUIRES_ADMIN, indicating that no permission information was returned because the user did not have Administrator capability. If you do not want the permissions returned, specify NULL for this parameter.
limit	The limits which have been assigned for this field. If you do not want the limits returned, specify NULL for this parameter.

displayList

The display list is an array of entries, each of which describes how the field should be displayed on the screen. Each of these items has an associated character tag to identify it. This tag can be specified during Export operations to get only the definition for a given tag. Each entry contains the tag, the name for the field, the position of the field, and the type of control to use on the screen.

If you do not want the displayList returned, specify NULL for this parameter.

helpText

The help text that is associated with the field. Space is allocated for the text and a pointer to that space returned. If there is no help text for the schema, a NULL pointer is returned.

If you do not want the helpText returned, specify NULL for this parameter. This is useful when you will not be using the helpText information as you are calling this routine to verify the existence of a schema. By not retrieving the helpText, the operation is more efficient and takes less time and space.

timestamp

Timestamp indicating when this field was last changed.

If you do not want the timestamp returned, specify NULL for this parameter.

owner

The user who is defined as the owner of this field definition.

If you do not want the owner returned, specify NULL for this parameter.

lastChanged

The user who last changed this field definition in some way.

If you do not want the lastChanged user returned, specify NULL for this parameter.

changeDiary

The change diary that is associated with the field. Space is allocated for the text and a pointer to that space returned. If there is no change diary for the field, a NULL pointer is returned.

The text returned is a formatted diary string. Use the ARDecodeDiary call to decode this string into an array of timestamp, user name, and text string pieces.

If you do not want the change diary returned, specify NULL for this parameter. This is useful when you will not be using the change diary information as you are calling this routine to verify the existence of a field. By not retrieving the change diary, the operation is more efficient and takes less time and space.

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information

n **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateField (3), **ARDecodeDiary** (3), **ARDeleteField** (3), **ARGetSchema** (3), **ARGetListField** (3), **ARSetField** (3), **FreeARDisplayList** (3), **FreeARFieldLimitStruct** (3), **FreeARPermissionList** (3), **FreeARStatusList** (3), **FreeARValueStruct** (3)

ARGetFilter

NAME

ARGetFilter – retrieve information about a filter in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetFilter (control, name, order, schema, opSet, enable, query, actionList, helpText, timestamp, owner,
      lastChanged, changeDiary, status)
    ARControlStruct      *control;
    ARNameType           name;
    unsigned int         *order;
    ARNameType           schema;
    unsigned int         *opSet;
    unsigned int         *enable;
    ARQualifierStruct    *query;
    ARFilterActionList   *actionList;
    char                 **helpText;
    ARTimestamp          *timestamp;
    ARNameType           owner;
    ARNameType           lastChanged;
    char                 **changeDiary;
    ARStatusList         *status;
```

DESCRIPTION

ARGetFilter will retrieve information about a filter with the indicated name on the specified server.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the filter to retrieve.

RETURN VALUES

order	The order of the filter. The filter order is a code between 0 and 1000, inclusive. It allows the ordering of filters so that filters with lower orders are executed before filters with higher orders. So, the filter order allows you to specify the order in which filters will be processed and lets you insure that given filters will be performed in the order you desire. If you do not want the order returned, specify NULL for this parameter.
schema	The name of the schema the filter is tied to. If you do not want the schema returned, specify NULL for this parameter.
opSet	A bit mask of the operations this filter applies to. This field consists of one or more of the following values OR'ed together: **AR_OPERATION_GET**, **AR_OPERATION_SET**, **AR_OPERATION_CREATE**, **AR_OPERATION_DELETE**, and **AR_OPERATION_MERGE**. If you do not want the operation set returned, specify NULL for this parameter.
enable	A flag set to 0 if this filter is disabled and to 1 if it is enabled. If you do not want the enable flag returned, specify NULL for this parameter.
query	A qualification that is used to test the operation/record being accessed. The operation/values of fields for the record must match this qualification to trigger the filter. If there is no qualifying condition, a value of AR_COND_OP_NONE is returned for this value. If you do not want the query returned, specify NULL for this parameter.
actionList	The set of one or more actions to take when the filter conditions are met. Every filter MUST have at least one action and can have up to AR_MAX_ACTIONS. If you do not want the action list returned, specify NULL for this parameter.
helpText	The help text that is associated with the filter. If you do not want the help text returned, specify NULL for this parameter. This is useful when you will not be using the help text information as you are calling this routine to verify the existence of a filter. By not retrieving the help text, the operation is more efficient and takes less time and space.
timestamp	The timestamp indicating when this filter was last changed. If you do not want the timestamp returned, specify NULL for this parameter.
owner	The user who is defined as the owner of this filter definition. If you do not want the owner returned, specify NULL for this parameter.
lastChanged	The user who last changed this filter definition in some way. If you do not want the lastChanged user returned, specify NULL for this parameter.

changeDiary

The change diary that is associated with the filter. Space is allocated for the text and a pointer to that space returned. If there is no change diary for the filter, a NULL pointer is returned.

The text returned is a formatted diary string. Use the ARDecodeDiary call to decode this string into an array of timestamp, user name, and text string pieces.

If you do not want the change diary returned, specify NULL for this parameter. This is useful when you will not be using the change diary information as you are calling this routine to verify the existence of a filter. By not retrieving the change diary, the operation is more efficient and takes less time and space.

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateFilter (3), **ARDecodeDiary** (3), **ARDeleteFilter** (3), **ARGetListFilter** (3), **ARSetFilter** (3), **FreeARFilterActionList** (3), **FreeARQualifierStruct** (3), **FreeARStatusList** (3)

5

ARGetFullTextInfo

NAME

ARGetFullTextInfo – get full text information from an AR System server

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"

int ARGetFullTextInfo (control, requestList, fullTextInfo, status)
        ARControlStruct              *control;
        ARFullTextInfoRequestList    *requestList;
        ARFullTextInfoList           *fullTextInfo;
        ARStatusList                 *status;
```

DESCRIPTION

ARGetFullTextInfo retrieves one or more pieces of information about the AR System server full text environment.

Access to this information is available to all users.

INPUT ARGUMENTS

control — The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

requestList — This is a list of one or more codes indicating the information that is desired. The various codes are discussed below under the FullTextInfo parameter.

RETURN VALUES

fullTextInfo — The information retrieved from the server. If there is an error retrieving any piece of information, its value will be NULL (have datatype set to AR_DATA_TYPE_NULL).

Following are the various codes that can be requested:

- **AR_FULLTEXTINFO_COLLECTION_DIR**: A character string containing the directory in which the full text collection information is stored.
- **AR_FULLTEXTINFO_STOPWORD**: A structure containing the currently defined words to ignore (stopwords) for the full text collection.
- **AR_FULLTEXTINFO_CASE_SENSITIVE_SRCH**: An integer indicating whether the search will be performed in a case-sensitive (AR_CASE_SENSITIVE_SEARCH) or case-insensitive (AR_CASE_INSENSITIVE_SEARCH) manner.

- **AR_FULLTEXTINFO_STATE:** An integer indicating whether full text support is Active (AR_FULLTEXT_STATE_ON) or inactive (AR_FULLTEXT_STATE_OFF).

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARGetServerInfo (3), **ARSetFullTextInfo** (3), **ARSetServerInfo** (3), **FreeARFullTextInfoList** (3), **FreeARFullTextInfoRequestList** (3), **FreeARStatusList** (3)

5

ARGetListActiveLink

NAME

ARGetListActiveLink – retrieve a list of active links on a server

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetListActiveLink (control, schema, changedSince, nameList, status)
    ARControlStruct     *control;
    ARNameType          schema,
    ARTimestamp         changedSince;
    ARNameList          *nameList;
    ARStatusList        *status;
```

DESCRIPTION

ARGetListActiveLink retrieves a list of all the active links on a given server. You can specify a timestamp that limits the active links retrieved to those changed after the time specified or a schema name that limits the list of active links to those related to a given schema. These are useful for getting a list of things that are different since the last time you accessed the server or that were tied to a specific server.

Access to active links is controlled through the security scheme of the AR System. The user identified in the control record is used to determine which active links are accessible to the requestor. Only active links that are accessible to the user are returned.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
schema	The name of a schema to limit active links retrieved to only those linked to the specified schema. If this parameter is NULL or an empty string, there is no schema qualification so all active links will be retrieved. If specified, only those linked to the specified schema are retrieved.
changedSince	Timestamp used to limit active links returned to those that have been changed after this time. To specify retrieval of ALL active links (subject to qualification by the schema as noted above), set the timestamp to 0.

RETURN VALUES

nameList	A list of all the active links, matching the time and schema criteria, to which the user has access. If no accessible active links match the criteria, the routine will return success but this list will contain 0 names.

status A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateActiveLink (3), **ARDeleteActiveLink** (3), **ARDeleteSchema** (3), **ARGetActiveLink** (3), **ARSetActiveLink** (3), **FreeARNameList** (3), **FreeARStatusList** (3)

ARGetListAdminExtension

NAME

ARGetListAdminExtension – retrieve a list of administrator extensions on a server

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetListAdminExtension (control, changedSince, nameList, status)
    ARControlStruct     *control;
    ARTimestamp         changedSince;
    ARNameList          *nameList;
    ARStatusList        *status;
```

DESCRIPTION

ARGetListAdminExtension retrieves a list of all the administrator extensions on a given server. You can specify a timestamp that limits the extensions retrieved to those changed after the time specified. This is useful for getting a list of things that are different since the last time you accessed the server.

Access to administrator extensions is controlled through the security scheme of the AR System. The user identified in the control record is used to determine which extensions are accessible to the requestor. Only extensions that are accessible to the user are returned.

INPUT ARGUMENTS

control — The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

changedSince — Timestamp used to limit extensions returned to those that have been changed after this time. To specify retrieval of ALL extensions, set the timestamp to 0.

RETURN VALUES

nameList — A list of all the extensions matching the time criteria to which the user has access.

If no accessible extensions match the criteria, the routine will return success but this list will contain 0 names.

status — A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateAdminExtension (3), **ARDeleteAdminExtension** (3), **ARExecuteAdminExtension** (3), **ARGetAdminExtension** (3), **ARSetAdminExtension** (3), **FreeARNameList** (3), **FreeARStatusList** (3)

ARGetListCharMenu

NAME

ARGetListCharMenu – retrieve a list of character menus on a server

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetListCharMenu (control, changedSince, nameList, status)
    ARControlStruct     *control;
    ARTimestamp         changedSince;
    ARNameList          *nameList;
    ARStatusList        *status;
```

DESCRIPTION

ARGetListCharMenu retrieves a list of all the character menus on a given server. You can specify a timestamp that limits the character menus retrieved to those changed after the time specified. This is useful for getting a list of things that are different since the last time you accessed the server.

Access to character menu information is available to all users.

INPUT ARGUMENTS

control — The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

changedSince — Timestamp used to limit character menus returned to those that have been changed after this time. To specify retrieval of ALL character menus set the timestamp to 0.

RETURN VALUES

nameList — A list of all the character menus matching the time criteria.

If no character menus match the criteria, the routine will return success but this list will contain 0 names.

status — A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully

- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateCharMenu (3), **ARDeleteCharMenu** (3), **ARExpandCharMenu** (3), **ARGetCharMenu** (3), **ARSetCharMenu** (3), **FreeARNameList** (3), **FreeARStatusList** (3)

ARGetListEntry

NAME

ARGetListEntry – retrieve a list of entries meeting a specified condition

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetListEntry (control, schema, qualifier, sortList, maxRetrieve, entryList, numMatches, status)
      ARControlStruct          *control;
      ARNameType               schema;
      ARQualifierStruct        *qualifier;
      ARSortList               *sortList;
      unsigned int             maxRetrieve;
      AREntryListList          *entryList;
      unsigned int             *numMatches;
      ARStatusList             *status;
```

DESCRIPTION

ARGetListEntry is a high performance database search. It will retrieve the entry ID and list description of all the entries in the schema that meet the specified criteria and are accessible by the user.

The access control scheme of the system controls which entries can be searched. If you have access to the entry ID field of an entry, the entry is included in your search set and can be selected by this call. If you don't, the item is not in the search set and will never be returned by this call.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
schema	Identifies the schema that is to be searched.

qualifier

A structure identifying the condition(s) to be searched on. Any field that is accessible to the user can be qualified. Qualifications include conditional operations (AND, OR, NOT), relational operations (=, !=, >, >=, <, <=, and a pattern matching LIKE operator), and arithmetic operations on numeric types. The condition can apply to a single field or to multiple fields.

You must have read access to all the fields being qualified. If a field does not exist or you do not have read access to a field, an error is returned and the operation is not performed.

To select ALL entries, supply either a NULL or a structure with the operation field set to AR_COND_OP_NONE.

sortList

A list of fields to sort the resulting entries by. You can specify any number of fields and specify whether to sort in ascending or descending order.

You must have read access to all the fields on which you are sorting. If you do not have read access to a field, the operation will be rejected.

If sorting is not important, you can specify either a NULL or set the numItems field to 0. By default, the entries are sorted by entry ID.

maxRetrieve

The maximum number of entries to retrieve. This setting allows you to specify an upper boundary on the number of entries that this call will retrieve. This helps to limit calls to the database that access too many entries due to insufficient qualifying criteria.

Specify a value of AR_NO_MAX_LIST_RETRIEVE to specify unlimited retrieval of *all* possible matches.

RETURN VALUES

entryList

A list of all the entries matching the specified criteria. Each item in the list will supply the entry ID and the list description for the matching entries. If you do not have access to fields defined as part of the list description, those fields will be omitted. If you do not have access to any of the fields, the list description will be an empty string.

If no entries match the criteria, the routine will return success but this list will contain 0 entries.

numMatches

This field will be loaded with the total number of matches found for the qualification criteria. If the number that matches is less than the value for **maxRetrieve**, this number will be equal to the number of items returned. Otherwise, it will be equal to the total number that would have been returned had they all been returned.

This parameter can be set to NULL to ignore the total count. If set to NULL, the database can abandon the search for matches as soon as it hits the maximum limit specified by **maxRetrieve**. This provides for better performance with queries that match more than what is desired. With this parameter, the search must continue in order to complete the count of potential matches. There is a gain from a reduction of data returned, but the search costs are unchanged.

status A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR:** Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateEntry (3), **ARDeleteEntry** (3), **ARGetEntry** (3), **ARGetEntryStatistics (3), ARLoadARQualifierStruct (3), ARMergeEntry** (3), **ARSetEntry** (3), **FreeAREntryList** (3), **FreeARQualifierStruct** (3), **FreeARSortList** (3), **FreeARStatusList** (3)

ARGetListEscalation

NAME

ARGetListEscalation – retrieve a list of escalations on a server

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetListEscalation (control, schema, changedSince, nameList, status)
    ARControlStruct     *control;
    ARNameType          schema;
    ARTimestamp         changedSince;
    ARNameList          *nameList;
    ARStatusList        *status;
```

DESCRIPTION

ARGetListEscalation retrieves a list of all the escalations on a given server. You can specify a timestamp that limits the escalations retrieved to those to be fired before the time specified and/or a schema name that limits the list of escalations to those related to a given schema. These are useful for getting a list of things that are different since the last time you accessed the server and/or that were tied to a specific server.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
schema	The name of a schema to limit escalations retrieved to only those linked to the specified schema. If this parameter is NULL or an empty string, there is no schema qualification so all escalations will be retrieved. If specified, only those linked to the specified schema are retrieved.
changedSince	Timestamp used to limit escalations returned to those that are to be fired before this time. To specify retrieval of *all* escalations, set the timestamp to 0.

RETURN VALUES

nameList	A list of all the escalations matching the time and schema criteria. If no escalations match the criteria, the routine will return success but this list will contain 0 names.
status	A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later. The return of the function itself is an integer indicating the success of the call. The return will be one of the following values: n **AR_RETURN_OK**: Operation successful n **AR_RETURN_WARNING**: Warning during process but operation completed successfully n **AR_RETURN_ERROR**: Operation failed due to error n **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information n **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateEscalation (3), **ARDeleteEscalation** (3), **ARDeleteSchema** (3), **ARGetEscalation** (3), **ARSetEscalation** (3), **FreeARNameList** (3), **FreeARStatusList** (3)

5

ARGetListField

NAME

ARGetListField – retrieve a list of fields for a given schema

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetListField (control, schema, changedSince, idList, status)
    ARControlStruct          *control;
    ARNameType               schema;
    ARTimestamp              changedSince;
    ARInternalIdList         *idList;
    ARStatusList             *status;
```

DESCRIPTION

ARGetListField retrieves a list of all the fields for a given schema. You can specify a timestamp that limits the fields retrieved to those changed after the time specified. This is useful for getting a list of things that are different since the last time you accessed the schema.

Access to field lists is available to all users who have permission to access the schema.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
schema	The name of the schema from which to get a list of fields.
changedSince	Timestamp used to limit fields returned to those that have been changed after this time. To specify retrieval of *all* fields, set the timestamp to 0.

RETURN VALUES

idList	A list of the internal IDs of all the fields matching the time criteria. If no fields match the criteria, the routine will return success but this list will contain 0 IDs.
status	A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later. The return of the function itself is an integer indicating the success of the call. The return will be one of the following values: n **AR_RETURN_OK**: Operation successful

- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateField (3), **ARDeleteField** (3), **ARGetField** (3), **ARSetField** (3), **FreeARInternalIDList** (3), **FreeARStatusList** (3)

ARGetListFilter

NAME

ARGetListFilter – retrieve a list of filters on a server

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetListFilter (control, schema, changedSince, nameList, status)
    ARControlStruct     *control;
    ARNameType          schema;
    ARTimestamp         changedSince;
    ARNameList          *nameList;
    ARStatusList        *status;
```

DESCRIPTION

ARGetListFilter retrieves a list of all the filters on a given server. You can specify a timestamp that limits the filters retrieved to those changed after the time specified and/or a schema name that limits the list of filters to those related to a given schema. These are useful for getting a list of things that are different since the last time you accessed the server and/or that were tied to a specific server.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control — The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

schema — The name of a schema to limit filters retrieved to only those linked to the specified schema. If this parameter is NULL or an empty string, there is no schema qualification so all filters will be retrieved. If specified, only those linked to the specified schema are retrieved.

changedSince — Timestamp used to limit filters returned to those that have been changed after this time. To specify retrieval of *all* filters, set the timestamp to 0.

RETURN VALUES

nameList — A list of all the filters matching the time and schema criteria.

If no filters match the criteria, the routine will return success but this list will contain 0 names.

status — A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateFilter (3), **ARDeleteFilter** (3), **ARDeleteSchema** (3), **ARGetFilter** (3), **ARSetFilter** (3), **FreeARNameList** (3), **FreeARStatusList** (3)

ARGetListGroup

NAME

ARGetListGroup – retrieve a list of groups from the access control cache

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetListGroup (control, userName, groupList, status)
    ARControlStruct     *control;
    ARNameType          *userName;
    ARGroupInfoList     *groupList;
    ARStatusList        *status;
```

DESCRIPTION

ARGetListGroup retrieves a list of all the groups known on a given server.

This operation can be performed by any user to get information about the list of groups on the server or for a list of groups for yourself. To get information about groups for another user, you must have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control — The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

userName — The name of a specific user of the system. If specified, a list of groups that the specified user is a member of is returned. If not specified (set to NULL), a list of all groups defined on the server is returned.

RETURN VALUES

groupList — A list of all the groups the user is a member of or, if no user is specified, that are known on the server. The information returned consists of a list of the unique group IDs along with the one or more names associated with each ID.

status — A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARGetListUser (3), **FreeARGroupInfoList** (3), **FreeARStatusList** (3)

ARGetListSchema

NAME

ARGetListSchema – retrieve a list of schemas on a server

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetListSchema (control, changedSince, nameList, status)
        ARControlStruct         *control;
        ARTimestamp             changedSince;
        ARNameList              *nameList;
        ARStatusList            *status;
```

DESCRIPTION

ARGetListSchema retrieves a list of all the schemas on a given server. You can specify a timestamp that limits the schemas retrieved to those changed after the time specified. This is useful for getting a list of things that are different since the last time you accessed the server.

Access to schema lists is available to all users. The list of schemas returned will be limited to those schemas to which you have access.

INPUT ARGUMENTS

control — The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

changedSince — Timestamp used to limit schemas returned to those that have been changed after this time. To specify retrieval of *all* schemas, set the timestamp to 0.

RETURN VALUES

nameList — A list of all the schemas matching the time criteria to which you have access.

If no accessible schemas match the criteria, the routine will return success but this list will contain 0 names.

status — A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateSchema (3), **ARDeleteSchema** (3), **ARGetSchema** (3), **ARSetSchema** (3), **FreeARNameList** (3), **FreeARStatusList** (3)

5

ARGetListServer

NAME

ARGetListServer – retrieve a list of servers accessible from the current machine

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"

int ARGetListServer (serverList, status)
        ARServerNameList        *serverList;
        ARStatusList            *status;
```

DESCRIPTION

ARGetListServer retrieves a list of all the servers that are accessible from the current machine. It gets the list of servers by processing the AR directory file `/etc/ar` (in UNIX) or `<`*`ar_config_dir`*`>\ar` (in the Windows NT server), retrieving all registered AR System servers.

The `/etc/ar` file may be under NIS control. If it is, the NIS map is consulted instead of any local `/etc/ar` file.

Access to the server list is available to all users.

RETURN VALUES

serverList — A list of all the servers that are registered.

If no servers are registered, the routine will return success but this list will contain 0 names.

status — A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

FreeARServerNameList (3), **FreeARStatusList** (3)

ARGetListSQL

NAME

ARGetListSQL – retrieve a list of results for a specified SQL command

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetListSQL(control, sqlCommand, maxRetrieve, valueListList, numMatches,
      status)
    ARControlStruct          *control;
    char                     *sqlCommand;
    unsigned int             maxRetrieve;
    ARValueListList          *valueListList;
    unsigned int             *numMatches;
    ARStatusList             *status;
```

DESCRIPTION

ARGetListSQL allows you to issue an SQL command directly to the underlying SQL database. It will retrieve values appropriate to the command issued and will return the 0 or more rows in the database that match. If this operation is attempted against a flat file database, 0 matches and a warning will be returned.

The SQL command is issued by the AR System server process so the permissions of the AR System server are used to control what level of access is available to items in the database.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
sqlCommand	The SQL command to be issued. You can issue any SQL command, but all the syntax rules that are appropriate to the underlying database must be followed. You must have access to perform the requested operation. Remember, you are running as the AR System server so that user must have permissions to perform the operation being requested.

maxRetrieve

The maximum number of rows to retrieve. This setting allows you to specify an upper boundary on the number of rows that this call will retrieve. This helps to limit calls to the database that access too many rows due to insufficient qualifying criteria.

Specify a value of AR_NO_MAX_LIST_RETRIEVE to specify unlimited retrieval of ALL possible matches.

RETURN VALUES

valueListList

A list of all the columns for all the rows that were selected by the command. Each item in this list is a list of the values that were retrieved for a particular row selected from the database.

If no rows match the criteria, the routine will return success but this list will contain 0 entries.

numMatches

This field will be loaded with the total number of matches found for the qualification criteria. If the number that matches is less than the value for **maxRetrieve**, this number will be equal to the number of items returned. Otherwise, it will be equal to the total number that would have been returned had they all been returned.

This parameter can be set to NULL to ignore the total count. If set to NULL, the database can abandon the search for matches as soon as it hits the maximum limit specified by **maxRetrieve**. This provides for better performance with queries that match more than what is desired. With this parameter, the search must continue in order to complete the count of potential matches. There is a gain from a reduction of data returned, but the search costs are unchanged.

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARGetListEntry(3)

ARGetListUser

NAME

ARGetListUser – retrieve a list of users from the AR System server

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetListUser (control, userListType, userList, status)
    ARControlStruct          *control;
    unsigned int             userListType;
    ARUserInfoList           *userList;
    ARStatusList             *status;
```

DESCRIPTION

ARGetListUser retrieves a list of all the users known on a given server or of all the users currently accessing a given server, or information about the current user.

If the operation is to get information about yourself (the current user), the operation can be performed by any user. To get information about other users, you must have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
userListType	Code for the type of user list wanted. The code can be either AR_USER_LIST_MYSELF to get information about the current user, AR_USER_LIST_REGISTERED to get a list of all registered users, or AR_USER_LIST_CURRENT to get a list of all the users who are currently accessing the server.

RETURN VALUES

userList	A list of all the users in the category specified by **userListType**. The information returned consists of a list of the user names along with the license type of the user. If the category is AR_USER_LIST_CURRENT, the list also contains a timestamp for the last time the user accessed the server.
status	A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later. The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARGetListGroup (3), **FreeARUserInfoList** (3), **FreeARStatusList** (3)

ARGetSchema

NAME

ARGetSchema – retrieve information about a schema in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARGetSchema (control, name, groupList, adminGroupList, getListFields, indexList, helpText, timestamp,
        owner, lastChanged, changeDiary, status)
    ARControlStruct          *control;
    ARNameType               name;
    ARInternalIdList         *groupList;
    ARInternalIdList         *adminGroupList;
    AREntryListFieldList     *getListFields;
    ARIndexList              *indexList;
    char                     **helpText;
    ARTimestamp              *timestamp;
    ARNameType               owner;
    ARNameType               lastChanged;
    char                     **changeDiary;
    ARStatusList             *status;
```

DESCRIPTION

ARGetSchema will retrieve information about the schema indicated by name on the specified server. The information that is available is the global schema information, not information on specific fields within the schema.

Access to schema information is restricted to users who have been granted access to the schema. If you are granted permission to the schema, you can see all data except groupList information which is limited to users with Administrator capability. If you are not granted permission, you cannot retrieve any information about the schema.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the schema to get information about.

RETURN VALUES

groupList	A list of 0 or more groups defining the groups whose users will be allowed to access this schema. Only a user with Administrator capability can retrieve the groupList information. Any other user will receive an empty list along with a warning, AR_WARN_PERM_REQUIRES_ADMIN, indicating that no groupList information was returned because the user did not have Administrator capability. If you do not want the groupList returned, specify NULL for this parameter.
adminGroupList	A list of 0 or more groups defining the groups whose users have potential sub-administration access to this schema. Only a user with Administrator capability can retrieve the adminGroupList information. Any other user will receive an empty list along with a warning, AR_WARN_PERM_REQUIRES_ADMIN, indicating that no adminGroupList information was returned because the user did not have Administrator capability. If you do not want the adminGroupList returned, specify NULL for this parameter.
getListFields	A list of 0 or more fields that define the information that will be returned in the description component of an ARGetListEntry call. If you do not want getListFields returned, specify NULL for this parameter.
indexList	A list of the indexes to define on the schema. If you do not want the indexList returned, specify NULL for this parameter.
helpText	The help text that is associated with the schema. Space is allocated for the text and a pointer to that space returned. If there is no help text for the schema, a NULL pointer is returned. If you do not want the help text returned, specify NULL for this parameter. This is useful when you will not be using the help text information as you are calling this routine to verify the existence of a schema. By not retrieving the help text, the operation is more efficient and takes less time and space.
timestamp	Timestamp indicating when this schema was last changed. Changes include changes to the schema information itself, to any existing field within the schema (including changes to character menus), and the addition of new fields. So, this timestamp can be used to detect any change to the schema definition. If you do not want the timestamp returned, specify NULL for this parameter.

owner The user who is defined as the owner of this schema definition.

If you do not want the owner returned, specify NULL for this parameter.

lastChanged The user who last changed this schema definition in some way.

If you do not want the lastChanged user returned, specify NULL for this parameter.

changeDiary The change diary that is associated with the schema. Space is allocated for the text and a pointer to that space returned. If there is no change diary for the schema, a NULL pointer is returned.

The text returned is a formatted diary string. Use the ARDecodeDiary call to decode this string into an array of timestamp, user name, and text string pieces.

If you do not want the change diary returned, specify NULL for this parameter. This is useful when you will not be using the change diary information as you are calling this routine to verify the existence of a schema. By not retrieving the change diary, the operation is more efficient and takes less time and space.

status A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateSchema (3), **ARDecodeDiary** (3), **ARDeleteSchema** (3), **ARGetField** (3), **ARGetListField** (3), **ARGetListSchema** (3), **ARSetSchema** (3), **FreeAREntryListFieldList** (3), **FreeARIndexList** (3), **FreeARInternalIdList** (3), **FreeARStatusList** (3)

ARGetServerInfo

NAME

ARGetServerInfo – get information from an AR System server

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"

int ARGetServerInfo (control, requestList, serverInfo, status)
    ARControlStruct             *control;
    ARServerInfoRequestList     *requestList;
    ARServerInfoList            *serverInfo;
    ARStatusList                *status;
```

DESCRIPTION

ARGetServerInfo retrieves one or more pieces of information about the AR System server environment.

Access to this information is available to all users.

INPUT ARGUMENTS

control — The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

requestList — This is a list of one or more codes indicating the information that is desired. The various codes are discussed below under the serverInfo parameter.

RETURN VALUES

Some return values (for example, for full text search) do not apply to the Windows NT server.

serverInfo — The information retrieved from the server. If there is an error retrieving any piece of information, its value will be NULL (have datatype set to AR_DATA_TYPE_NULL).

Following are the various codes that can be requested:

- **AR_SERVER_INFO_DB_TYPE**: A character string containing the type of underlying database being used by the server.
- **AR_SERVER_INFO_SERVER_LICENSE**: A character string noting the type of server license defined for the server.
- **AR_SERVER_INFO_USER _LICENSE**: An integer noting the number of fixed write licenses defined on the server.
- **AR_SERVER_INFO_VERSION**: A character string holding the version of the AR System server.

- **AR_SERVER_INFO_ALLOW_GUESTS**: An integer flag with 1 indicating that guest users are allowed in the system and 0 indicating they are not allowed. Guest users are users who are not registered with the AR System. If allowed, they can access only data with "Public" view access and submit new entries with fields that have a create mode of "Open".
- **AR_SERVER_INFO_USE_ETC_PASSWD**: An integer flag with 1 indicating that the /etc/passwd file will be searched if the user is not registered with the AR System and 0 indicating that /etc/passwd will not be searched.
- **AR_SERVER_INFO_XREF_PASSWORDS**: An integer flag with 1 indicating that the system will check passwords in /etc/passwd for any registered user with a blank password in the AR System and 0 indicating there is no cross reference check.
- **AR_SERVER_INFO_DEBUG_MODE**: An integer bitmask that specifies which debugging modes are active in the system (bit 1 is the low order bit): bit 1 specifies SQL tracing, bit 2 specifies Filter tracing, bit 3 specifies User tracing, bit 4 specifies Escalation tracing, and bit 5 specifies API tracing.
- **AR_SERVER_INFO_DB_NAME**: A character string containing the name of the database/tablespace being used by the AR System. Empty string if using a flat file database.
- **AR_SERVER_INFO_HARDWARE**: A character string containing the type of hardware on which the server is running.
- **AR_SERVER_INFO_OS**: A character string containing the type of operating system under which the server is running.
- **AR_SERVER_INFO_SERVER_DIR**: A character string containing the path specified for the Server directory. For flat file installations, this is where the data files are located. For all installations, this is where miscellaneous support data files are placed.
- **AR_SERVER_INFO_DBHOME_DIR**: A character string containing the path to the SQL database home directory. This is used only if the database being used is an SQL database.
- **AR_SERVER_INFO_SET_PROC_TIME**: An integer set to the maximum time to wait for a filter run process operation that is returning a value.
- **AR_SERVER_INFO_EMAIL_FROM**: A character string containing the name of the user who will be specified as the source of all email notifications.
- **AR_SERVER_INFO_SQL_LOG_FILE**: A character string containing the filename (relative or absolute) where the information from the SQL tracing operation is placed.

- **AR_SERVER_INFO_FLOAT_LICENSE**: An integer noting the number of floating write licenses defined on the server.
- **AR_SERVER_INFO_FLOAT_TIMEOUT**: An integer noting the number of hours before a floating write license will automatically timeout.
- **AR_SERVER_INFO_UNQUAL_QUERIES**: An integer flag with 1 indicating that the server will respond to unqualified queries from users and 0 indicating that unqualified queries will return an error.
- **AR_SERVER_INFO_FILTER_LOG_FILE**: A character string containing the filename (relative or absolute) where the information from the Filter tracing operation is placed.
- **AR_SERVER_INFO_USER_LOG_FILE**: A character string containing the filename (relative or absolute) where the information from the User tracing operation is placed.
- **AR_SERVER_INFO_REM_SERV_ID**: A character string containing the Remedy server id that is associated with the server license.
- **AR_SERVER_INFO_EMBEDDED_SQL**: An integer flag with 1 indicating that the server is running with an embedded SQL database and 0 indicating that the database is not embedded in (was purchased separately from) the AR System.
- **AR_SERVER_INFO_MAX_SCHEMAS**: An integer which if set to 0 indicates no limit on the number of schemas allowed on the server and if set to a number indicates the maximum number of schemas that can be created on the server.
- **AR_SERVER_INFO_DB_VERSION**: A character string containing the version of the database that is being used by the AR System.
- **AR_SERVER_INFO_MAX_ENTRIES**: An integer that identifies the maximum number of entries that will be returned in response to a single ARGetListEntry call. This value works in combination with the value that can be defined by the user in the call to **ARGetListEntry** with the minimum of the two values taking precedence in a given call.
- **AR_SERVER_INFO_MAX_F_DAEMONS**: An integer that specifies the maximum number of “fast” servers that will be run in a multi-process server environment. (This option is effective only if the multi-process server option has been activated for the software.)
- **AR_SERVER_INFO_MAX_L_DAEMONS**: An integer that specifies the maximum number of “list” servers that will be run in a multi-process server environment. (This option is effective only if the multi-process server option has been activated for the software.)

- **AR_SERVER_INFO_ESCALATION_LOG_FILE**: A character string containing the filename (relative or absolute) where the information from the Escalation tracing operation is placed.
- **AR_SERVER_INFO_ESCL_DAEMON**: An integer that specifies whether a separate escalation servers will be run in a multi-process server environment. If set to 0, no separate escalation server will run. If set to 1, a separate escalation server will run. (This option is effective only if the multi-process server option has been activated for the software.)
- **AR_SERVER_INFO_SUBMITTER_MODE**: An integer flag that when set to AR_SUBMITTER_MODE_LOCKED indicates that the value in the Submitter field will be locked at submit time and not changeable thereafter and the Submitter is allowed to change values within permissions with or without a license and when set to AR_SUBMITTER_MODE_CHANGEABLE indicates that the value in the Submitter field can be changed at any time within permissions, but that any change using the Submitter group permissions requires a license.
- **AR_SERVER_INFO_API_LOG_FILE**: A character string containing the filename (relative or absolute) where the information from the API tracing operation is placed.
- **AR_SERVER_INFO_FTEXT_FIXED**: An integer noting the number of fixed full text licenses defined on the server.
- **AR_SERVER_INFO_FTEXT_FLOAT**: An integer noting the number of floating full text licenses defined on the server.
- **AR_SERVER_INFO_FTEXT_TIMEOUT**: An integer noting the number of hours before a floating full text license will automatically timeout.
- **AR_SERVER_INFO_RESERV1_A**: Reserved for future use.
- **AR_SERVER_INFO_RESERV1_B**: Reserved for future use.
- **AR_SERVER_INFO_RESERV1_C**: Reserved for future use.
- **AR_SERVER_INFO_SERVER_IDENT**: A character string containing a unique identifier of a server machine. The string can be used to test two servers to see if they are really the same machine.
- **AR_SERVER_INFO_DS_SVR_LICENSE**: A character string noting the type of distributed server license defined for the server.
- **AR_SERVER_INFO_DS_MAPPING**: A character string containing the name of the schema that contains the distributed mapping definitions for the distributed server system.

- **AR_SERVER_INFO_DS_PENDING**: A character string containing the name of the schema that contains the pending operation list for the distributed server system.
- **AR_SERVER_INFO_DS_RPC_SOCKET**: An integer that holds the socket number of the RPC socket being used by the distributed server environment. If NULL, the distributed server is using the default socket. If set, the specific socket specified is being used.
- **AR_SERVER_INFO_DS_LOG_FILE**: A character string containing the filename (relative or absolute) where the information from the distributed server tracing operation is placed.
- **AR_SERVER_INFO_SUPPRESS_WARN**: A character string containing a list of one or more note/warning numbers (separated by spaces). The messages that are tied to these numbers will be suppressed by the server.
- **AR_SERVER_INFO_HOSTNAME**: A character string containing the hostname of the server machine. This name is the “short” name of the system.
- **AR_SERVER_INFO_FULL_HOSTNAME**: A character string containing the full (DNS) hostname of the server machine. This name is the “long” name of the system.
- **AR_SERVER_INFO_SAVE_LOGIN**: An integer value that indicates whether to save login information in client tools and who controls that saving. Can be set to one of the following:
  0 - User controlled (default), 1 - Admin controlled, set to save the login information, 2 - Admin controlled, set to not save the login information.
- **AR_SERVER_INFO_U_CACHE_CHANGE**: An integer value that indicates the time at which the user cache last changed.
- **AR_SERVER_INFO_G_CACHE_CHANGE**: An integer value that indicates the time at which the group cache last changed.

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information

- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARGetFullTextInfo (3), **ARSetFullTextInfo** (3), **ARSetServerInfo** (3), **FreeARServerInfoList** (3), **FreeARServerInfoRequestList** (3), **FreeARStatusList** (3)

ARGetServerStatistics

NAME

ARGetServerStatistics – get statistics from an AR server

SYNOPSIS

```
#include “ar.h”
        #include “arerrno.h”
        #include “arextern.h”

int ARGetServerInfo(control, requestList, serverInfo, status)

        ARControlStruct*control;
        ARServerInfoRequestList*requestList;
        ARServerInfoList*serverInfo;
        ARStatusList*status;
```

DESCRIPTION

ARGetServerStatistics retrieves one or more statistics about the running AR server environment. In general the numbers are all raw counters. They contain a number of times of a cumulative time since the server being queried has started. If the values hit a value of MAX_LONG, they will roll to 0 and start growing from there again.

Access to this information is available to all users.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
requestList	This is a list of one or more codes indicating the statistics that are desired. The various codes are discussed below under the serverInfo parameter.

RETURN VALUES

serverInfo	The statistics retrieved from the server. If there is an error retrieving any piece of information, its value will be NULL (have datatype set to AR_DATA_TYPE_NULL).

Following are the various codes that can be requested:

- **AR_SERVER_STAT_START_TIME**: A timestamp containing the UNIX time at which this server was started.
- **AR_SERVER_STAT_BAD_PASSWORD**: An integer containing the total number of times a bad password was specified during login.
- **AR_SERVER_STAT_NO_WRITE_TOKEN**: An integer containing the total number of times a user tried to connect and there was no floating write token available.
- **AR_SERVER_STAT_NO_FULL_TOKEN**: An integer containing the total number of times a user tried to connect and there was no floating Full Text token available.
- **AR_SERVER_STAT_CURRENT_USERS**: An integer containing the total number of users currently accessing the system.
- **AR_SERVER_STAT_WRITE_FIXED**: An integer containing the total number user that are currently connected that have a fixed write license.
- **AR_SERVER_STAT_WRITE_FLOATING**: An integer containing the total number user that are currently connected that have a floating write license.
- **AR_SERVER_STAT_WRITE_READ**: An integer containing the total number user that are currently connected that have no write license.
- **AR_SERVER_STAT_FULL_FIXED**: An integer containing the total number user that are currently connected that have a fixed Full Text license.
- **AR_SERVER_STAT_FULL_FLOATING**: An integer containing the total number user that are currently connected that have a floating Full Text license.
- **AR_SERVER_STAT_FULL_NONE**: An integer containing the total number user that are currently connected that have no Full Text license.
- **AR_SERVER_STAT_API_REQUESTS**: An integer containing the total number of API requests that have been received.
- **AR_SERVER_STAT_API_TIME**: An integer containing the total time (clock time) spent in API calls.
- **AR_SERVER_STAT_ENTRY_TIME**: An integer containing the total time (clock time) spent in API calls manipulating an Entry.
- **AR_SERVER_STAT_RESTRUCT_TIME**: An integer containing the total time (clock time) spent in API calls performing a restructuring operation.
- **AR_SERVER_STAT_OTHER_TIME**: An integer containing the total time (clock time) spent in API calls that are not manipulating an entry and are not restructuring the database.

- **AR_SERVER_STAT_CACHE_TIME**: An integer containing the total time (clock time) spent loading the internal cache to improve performance.
- **AR_SERVER_STAT_GET_E_COUNT**: An integer containing the total number of calls made to the ARGetEntry API call.
- **AR_SERVER_STAT_GET_E_TIME**: An integer containing the total time (clock time) spent in the ARGetEntry API call.
- **AR_SERVER_STAT_SET_E_COUNT**: An integer containing the total number of calls made to the ARSetEntry API call.
- **AR_SERVER_STAT_SET_E_TIME**: An integer containing the total time (clock time) spent in the ARSetEntry API call.
- **AR_SERVER_STAT_CREATE_E_COUNT**: An integer containing the total number of calls made to the ARCreateEntry API call.
- **AR_SERVER_STAT_CREATE_E_TIME**: An integer containing the total time (clock time) spent in the ARCreateEntry API call.
- **AR_SERVER_STAT_DELETE_E_COUNT**: An integer containing the total number of calls made to the ARDeleteEntry API call.
- **AR_SERVER_STAT_DELETE_E_TIME**: An integer containing the total time (clock time) spent in the ARDeleteEntry API call.
- **AR_SERVER_STAT_MERGE_E_COUNT**: An integer containing the total number of calls made to the ARMergeEntry API call.
- **AR_SERVER_STAT_MERGE_E_TIME**: An integer containing the total time (clock time) spent in the ARMergeEntry API call.
- **AR_SERVER_STAT_GETLIST_E_COUNT**: An integer containing the total number of calls made to the ARGetListEntry API call.
- **AR_SERVER_STAT_GETLIST_E_TIME**: An integer containing the total time (clock time) spent in the ARGetListEntry API call.
- **AR_SERVER_STAT_E_STATS_COUNT**: An integer containing the total number of calls made to the ARGetEntryStatistics API call.
- **AR_SERVER_STAT_E_STATS_TIME**: An integer containing the total time (clock time) spent in the ARGetEntryStatistics API call.
- **AR_SERVER_STAT_FILTER_PASSED**: An integer containing the total number of filters that passed their qualification and were fired.
- **AR_SERVER_STAT_FILTER_FAILED**: An integer containing the total number of filters that failed their qualification and were skipped.
- **AR_SERVER_STAT_FILTER_DISABLE**: An integer containing the total number of filters that were checked, but were marked as disabled.

- **AR_SERVER_STAT_FILTER_NOTIFY**: An integer containing the total number of notify filter actions that were performed.
- **AR_SERVER_STAT_FILTER_MESSAGE**: An integer containing the total number of message filter actions that were performed.
- **AR_SERVER_STAT_FILTER_LOG**: An integer containing the total number of log filter actions that were performed.
- **AR_SERVER_STAT_FILTER_FIELDS**: An integer containing the total number of set fields filter actions that were performed.
- **AR_SERVER_STAT_FILTER_PROCESS**: An integer containing the total number of run process filter actions that were performed.
- **AR_SERVER_STAT_FILTER_TIME**: An integer containing the total time (clock time) spent during the checking and processing of filters.
- **AR_SERVER_STAT_ESCL_PASSED**: An integer containing the total number of escalations that passed their qualification and were fired.
- **AR_SERVER_STAT_ESCL_FAILED**: An integer containing the total number of escalations that failed their qualification and were skipped.
- **AR_SERVER_STAT_ESCL_DISABLE**: An integer containing the total number of escalations that were checked, but were marked as disabled.
- **AR_SERVER_STAT_ESCL_NOTIFY**: An integer containing the total number of notify escalation actions that were performed.
- **AR_SERVER_STAT_ESCL_LOG**: An integer containing the total number of log escalation actions that were performed.
- **AR_SERVER_STAT_ESCL_FIELDS**: An integer containing the total number of set fields escalation actions that were performed.
- **AR_SERVER_STAT_ESCL_PROCESS**: An integer containing the total number of run process escalation actions that were performed.
- **AR_SERVER_STAT_ESCL_TIME**: An integer containing the total time (clock time) spent during the checking and processing of escalations.
- **AR_SERVER_STAT_TIMES_BLOCKED**: An integer containing the total number of times that at least one API call was blocked behind the API call being processed.
- **AR_SERVER_STAT_NUMBER_BLOCKED**: An integer containing the total number of processes that were blocked when there was at least one blocked. So, if there were 5 times that a call was blocked and the total number of times blocked was 6, it means that 4 times there was 1 process blocked and 1 time there were 2.

- **AR_SERVER_STAT_CPU**: An integer containing the total time (CPU time) that has been used by the server.
- **AR_SERVER_STAT_SQL_DB_COUNT**: An integer containing the total number of SQL commands that have been issued to the database.
- **AR_SERVER_STAT_SQL_DB_TIME**: An integer containing the total time (clock time) spent during processing of a database operation.
- **AR_SERVER_STAT_FTS_SRCH_COUNT**: An integer containing the total number of FTS search operations that have been performed.
- **AR_SERVER_STAT_FTS_SRCH_TIME**: An integer containing the total time (clock time) spent during processing of FTS searches.
- **AR_SERVER_STAT_SINCE_START**: An integer containing the number of seconds since the start time. This represents the number of seconds (wall time) that the process has been running.

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARGetFullTextInfo(3), **ARGetServerInfo**(3), **FreeARServerInfoList**(3), **FreeARServerInfoRequestList**(3), **FreeARStatusList**(3)

ARImport

NAME

ARImport – import existing schema, filter, escalation, active link, menu, and administrator extension definitions

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARImport (control, structItems, importBuf, status)
      ARControlStruct          *control;
      ARStructItemList         *structItems;
      char                     *importBuf;
      ARStatusList             *status;
```

DESCRIPTION

ARImport will import one or more structure definitions (schemas, filters, active links, escalation, character menus, and administrator extensions) into the AR System. This allows the copying of definitions from one server to another.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control

The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

structItems

An array of the items to import. The type and name of the items to import are identified. The items indicated are imported with any other data in the file ignored.

To import the entire file contents, specify a NULL value for this parameter.

The codes for the types of imports allowed are as follows:

- **AR_STRUCT_ITEM_SCHEMA**: Full schema definition
- **AR_STRUCT_ITEM_FILTER**: Filter definition
- **AR_STRUCT_ITEM_ACTIVE_LINK:** Active Link definition
- **AR_STRUCT_ITEM_ADMIN_EXT**: Administrator extension definition
- **AR_STRUCT_ITEM_CHAR_MENU**: Character menu definition
- **AR_STRUCT_ITEM_ESCALATION**: Escalation definition

importBuf — Buffer containing the text of the items to import. The text may include a single item OR it may contain several items. Any one or more of the items in the text may be imported.

Note that if a schema is to be imported, only the full definition format of the schema can be imported. The other formats are used for caching of definition and/or view information and are not the complete definitions needed.

RETURN VALUES

status — A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARExport (3), **FreeARStatusList** (3), **FreeARStructItemList** (3)

ARInitialization

NAME

ARInitialization – initialize interaction with the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"
int ARInitialization (status)
        ARStatusList                    *status;
```

DESCRIPTION

ARInitialization serves to initialize the program for interaction with the AR System. For many systems, this call performs no work, while in others it establishes an initial state for the system. It should always be called in case it is needed by the environment.

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARTermination (3), **FreeARStatusList** (3)

5

ARLoadARQualifierStruct

NAME

ARLoadARQualifierStruct – parse a qualification string into an ARQualiferStruct

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARLoadARQualifierStruct (control, schema, displayTag, qualString, qualifier, status)
    ARControlStruct     *control;
    ARNameType          schema;
    ARNameType          displayTag;
    char                *qualString;
    ARQualifierStruct   *qualifier;
    ARStatusList        *status;
```

DESCRIPTION

ARLoadARQualifierStruct will parse the passed qualification string and if the qualification is legal for the schema, load an ARQualifierStruct that is appropriate for the qualification. This routine simplifies the loading of the qualification structure used to select a set of entries from the server.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
schema	The name of the schema the qualification applies to.
displayTag	Tag that identifies which specific display struct view to use when resolving field names. If there is a display that matches the tag, the name associated with that tag is used. If there is not one that matches, the display that is listed first in the displayList is used. If an empty string or NULL is specified, only the first item in the displayList is used for resolving field names.
qualString	The qualification that is to be parsed. The syntax follows the same rules as any qualification you can enter using the **aruser** program query bar.

RETURN VALUES

qualifier	The qualifier structure that is loaded as appropriate for the qualification that was specified. There may be nested space allocated for this structure so it is important to free the structure when no longer needed.

status — A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

aruser (1), **ARGetEntryStatistics** (3), **ARGetListEntry** (3), **FreeARStatusList** (3), **FreeARQualifierStruct** (3)

ARMergeEntry

NAME

ARMergeEntry – merge an entry into the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARMergeEntry (control, schema, fieldList, newIdIfConflict, entryId, status)
      ARControlStruct       *control;
      ARNameType            schema;
      ARFieldValueList      *fieldList;
      ARBoolean             newIdIfConflict;
      AREntryIdType         entryId;
      ARStatusList          *status;
```

DESCRIPTION

ARMergeEntry will merge an entry into the specified schema. The user can specify any number of fields and associated values. The system will check permissions for each field and report errors if a field does not exist or if the user does not have access. If any one of the fields is in error, the entire merge operation is rejected and no change is made to the schema.

Note that this routine is different from the ARCreateEntry routine in that the entry being created is assumed to already exist in some form in another schema in the AR System. The fields assigned MAY include the entry ID, create date, last modified user, last modified date, and status history fields. Any diary field MUST include the fully formatted diary value, NOT just new text for a diary item.

The system attempts to add an entry using the same entry ID that was supplied. However, if that ID conflicts with an existing ID, the system will return an error if the newIdIfConflict flag is FALSE or generate a new unique ID if the flag is TRUE.

Access to entries is controlled through the security scheme of the AR System. The user identified in the control record in combination with the create mode specified for the field is used to determine which fields can be updated by the user. If a value is provided for a field to which the user does not have write access, an error will be reported on that field and the operation will be cancelled. Each value is checked for to make sure it can be updated, with an error returned if not writable. Remember that the extra system fields are being assigned values so the access requirements against them are checked also.

INPUT ARGUMENTS

control
The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

schema
Identifies the schema to which the entry is to be merged.

fieldList
A list of field/value pairs for all the fields to be set for the new entry. The fields can be in any order in the list. Any nonexistent or inaccessible field will result in an error return. The datatype of the value must match the datatype of the field (or be NULL). All required fields that do not have a default value MUST be assigned a value. To assign no value for a field, assign the value NULL (AR_DATA_TYPE_NULL) as the value. If the field is a required field in the system, you cannot assign it a NULL value. Note that this use overrides any configured default value for the field.

Remember that diary values must be a fully formatted diary value (includes the user and time stamps just like a diary value returned by the ARGetEntry call).

newIdIfConflict
A flag that if set to TRUE indicates that the system should generate a new unique ID for the entry if the entry ID supplied with the request conflicts with an existing definition and if set to FALSE indicates that an error should be returned on conflict. If no entry ID is supplied or if there is no conflict with the existing entry ID, this field is ignored.

RETURN VALUES

entryId
This is the unique identifier for the entry. If an ID was supplied and was unique, it is the ID supplied. If no ID was supplied or there was a conflict and a new ID was generated, this is the new unique ID.

status
A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

arimport (1), **ARCreateEntry** (3), **ARGetListEntry** (3), **FreeARFieldValueList** (3), **FreeARStatusList** (3)

5

ARSetActiveLink

NAME

ARSetActiveLink – update an existing active link in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARSetActiveLink (control, name, newName, order, schema, groupList, executeMask, field, displayList,
                     enable, query, actionList, helpText, owner, changeDiary, status)

ARControlStruct          *control;
ARNameType               name;
ARNameType               newName;
unsigned int             *order;
ARNameType               schema;
ARInternalIdList         *groupList;
unsigned int             *executeMask;
ARInternalId             *field;
ARDisplayList            *displayList;
unsigned int             *enable;
ARQualifierStruct        *query;
ARActiveLinkActionList   *actionList;
char                     *helpText;
ARNameType               owner;
char                     *changeDiary;
ARStatusList             *status;
```

DESCRIPTION

ARSetActiveLink will update an existing active link with the indicated name on the specified server. The updates will be made immediately to the database and will be returned to users who request information about active links. Since the operation of active links is on clients accessing the server, the updated definition will not be available on individual clients until the client reloads configuration from the server (by reconnecting to the schema).

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the active link to update.
newName	The new name for the active link. Remember that all active link names on a given server must be unique.

	If you do not want the name of the active link changed, specify NULL for this parameter.
order	The new order of the active link. The active link order is a code between 0 and 1000, inclusive. It allows the ordering of active links so that active links with lower orders are executed before active links with higher orders. So, the active link order allows you to specify the order in which active links will be processed and lets you insure that given active links will be performed in the order you desire. If you do not want the active link option changed, specify NULL for this parameter.
schema	The name of the schema the active link is linked to. Every active link must be linked to a schema. Changing the schema will break the connection to the old schema and change it to the new one. If you do not want the schema changed, specify NULL for this parameter.
groupList	A list of 0 or more groups identifying the groups which will have access to this active link. Changing this list removes all old permissions and establishes the ones specified here as the new permissions. Defining a groupList with 0 items will define an active link that can be accessed only by users with Administrator capability. Defining a groupList that contains the "Public" group (group id 0) will define an active link that can be accessed by all users. If you do not want the group list changed, specify NULL for this parameter.
executeMask	A bit mask of the conditions under which this active link will be executed. This field consists of or more of the following values OR'ed together: **AR_EXECUTE_ON_BUTTON**, **AR_EXECUTE_ON_RETURN**, **AR_EXECUTE_ON_SUBMIT**, **AR_EXECUTE_ON_MODIFY**, **AR_EXECUTE_ON_DISPLAY**, **AR_EXECUTE_ON_MENU_CHOICE**, and **AR_EXECUTE_ON_SET_DEFAULT**. If you do not want the execute mask changed, specify NULL for this parameter.
field	The ID of the field the active link is tied to if the execute mask includes the AR_EXECUTE_ON_RETURN or ar_EXECUTE_ON_MENU_CHOICE flag. Providing this data will change which field the link is tied to. If you do not want the field changed, specify NULL for this parameter.
displayList	Information about the button and its position if the execute mask includes the AR_EXECUTE_ON_BUTTON flag. Providing this data will change the button on the view. If you do not want the display information changed, specify NULL for this parameter.
enable	A flag with a setting of 0 to indicate that this active link is to be marked as disabled so it will not be executed or 1 to indicate that the link is active and available for use. An active link that is disabled will not be visible to the end user and will not fire. If you do not want the enable flag changed, specify NULL for this parameter.
query	A qualification that is to be performed when the active link is executed. It will allow the conditional execute of the active link.

If there is no qualification, specify NULL or assign a value of AR_COND_OP_NONE to this value.

If you do not want the query changed, specify NULL for this parameter.

actionList The set of one or more actions to take when the active link is executed. Every active link MUST have at least one action and can have up to AR_MAX_ACTIONS.

If you do not want the action list changed, specify NULL for this parameter.

helpText The help text that is to be associated with the active link. The help text can be of any length. Existing help text can be eliminated by setting helpText to point to a 0-length string

If you do not want the help text changed, specify NULL for this parameter.

owner The new owner for the active link.

If you do not want to change the owner for the active link, specify NULL for this parameter.

changeDiary The additional change diary that is to be associated with the active link. The added change diary text can be of any length. The new text will be appended to the end of any existing text. Existing text cannot be deleted or changed. The new text is timestamped and user name stamped as it is added to the existing change diary text.

If the change diary is to be left unchanged, a NULL pointer should be supplied for this parameter.

RETURN VALUES

status A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateActiveLink (3), **ARDeleteActiveLink** (3), **ARGetActiveLink** (3), **ARGetListActiveLink** (3), **FreeARActiveLinkActionList** (3), **FreeARDisplayList** (3), **FreeARInternalIdList** (3), **FreeARQualifierStruct** (3), **FreeARStatusList** (3)

ARSetAdminExtension

NAME

ARSetAdminExtension – update information about an administrator extension in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARSetAdminExtension (control, name, newName, groupList, command, helpText, owner, changeDiary,
      status)
    ARControlStruct      *control;
    ARNameType           name;
    ARNameType           newName;
    ARInternalIdList     *groupList;
    char                 *command;
    char                 *helpText;
    ARNameType           owner;
    char                 *changeDiary;
    ARStatusList         *status;
```

DESCRIPTION

ARSetAdminExtension will update information about the administrator extension indicated by name on the specified server.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the extension to update.
newName	The new name for the extension. Remember that all extension names on a given server must be unique. If you do not want to change the name of the extension, specify NULL for this parameter.
groupList	A list of 0 or more groups identifying the groups which will have access to this active link. Changing this list removes all old permissions and establishes the ones specified here as the new permissions. Defining a groupList with 0 items will define an administrator extension that can be accessed only by users with Administrator capability. Defining a groupList that contains the “Public” group (group ID 0) will define an administrator extension that can be accessed by all users.

If you do not want the group list changed, specify NULL for this parameter.

command

The new command string for this extension.

If you do not want to change the command, specify NULL for this parameter.

helpText

The help text that is to be associated with the extension. The help text can be of any length. Existing help text can be eliminated by setting helpText to point to a 0-length string.

If the help text is to be left unchanged, a NULL pointer should be supplied for this parameter.

owner

The new owner for the extension.

If you do not want to change the owner for the extension, specify NULL for this parameter.

changeDiary

The additional change diary that is to be associated with the extension. The added change diary text can be of any length. The new text will be appended to the end of any existing text. Existing text cannot be deleted or changed. The new text is timestamped and user name stamped as it is added to the existing change diary text.

If the change diary is to be left unchanged, a NULL pointer should be supplied for this parameter.

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateAdminExtension (3), **ARDeleteAdminExtension** (3), **ARExecuteAdminExtension** (3), **ARGetAdminExtension** (3), **ARGetListAdminExtension** (3), **FreeARInternalIdList** (3), **FreeARStatusList** (3)

ARSetCharMenu

NAME

ARSetCharMenu – update an existing character menu in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARSetCharMenu (control, name, newName, refreshCode, menuDefn, helpText, owner, changeDiary,
      status)
    ARControlStruct      *control;
    ARNameType           name;
    ARNameType           newName;
    unsigned int         *refreshCode;
    ARCharMenuStruct     *menuDefn;
    char                 *helpText;
    ARNameType           owner;
    char                 *changeDiary;
    ARStatusList         *status;
```

DESCRIPTION

ARSetCharMenu will update an existing character menu with the indicated name on the specified server. The updates will be made immediately to the database and will be returned to users who request information about character menus. Since the use of character menus is on clients accessing the server, the updated definition will not be available on individual clients until the client requests the character menu definition again (controlled by the refresh code).

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the character menu to update.
newName	The new name for the character menu. Remember that all character menu names on a given server must be unique. If you do not want the name of the character menu changed, specify NULL for this parameter.

refreshCode

Code indicating when the menu should be refreshed. This code allows you to balance performance and the frequency at which the menu is checked for consistency with the server. The refresh code is one of AR_MENU_REFRESH_CONNECT, AR_MENU_REFRESH_OPEN, or AR_MENU_REFRESH_INTERVAL.

If you do not want the refresh code changed, specify NULL for this parameter.

menuDefn

The definition of the character menu.

If you do not want the menu definition changed, specify NULL for this parameter.

helpText

The help text that is to be associated with the character menu. The help text can be of any length. Existing help text can be eliminated by setting helpText to point to a 0-length string.

If you do not want the help text changed, specify NULL for this parameter.

owner

The new owner for the character menu.

If you do not want to change the owner for the character menu, specify NULL for this parameter.

changeDiary

The additional change diary that is to be associated with the character menu. The added change diary text can be of any length. The new text will be appended to the end of any existing text. Existing text cannot be deleted or changed. The new text is timestamped and user name stamped as it is added to the existing change diary text.

If the change diary is to be left unchanged, a NULL pointer should be supplied for this parameter.

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateCharMenu (3), **ARDeleteCharMenu** (3), **ARExpandCharMenu** (3), **ARGetCharMenu** (3), **ARGetListCharMenu** (3), **FreeARCharMenuStruct** (3), **FreeARStatusList** (3)

ARSetEntry

NAME

ARSetEntry – update an entry in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARSetEntry (control, schema, entryId, fieldList, getTime, status)
      ARControlStruct      *control;
      ARNameType           schema;
      AREntryIdType        entryId;
      ARFieldValueList     *fieldList;
      ARTimestamp          getTime;
      ARStatusList         *status;
```

DESCRIPTION

ARSetEntry will update information about the entry indicated by the entry ID in the specified schema. The user can specify any number of fields and associated values. The system will check permissions for each field and report errors if a field does not exist or if the user does not have access. If any one of the fields is in error, the entire set operation is rejected and no change is made to the entry.

Access to entries is controlled through the security scheme of the AR System. The user identified in the control record is used to determine which fields can be updated by the requestor. If a value is provided for a field that the user does not have write access to, an error will be reported on that field and the operation will be cancelled. Each value is checked to make sure it can be updated, with an error returned if not writable.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
schema	Identifies the schema that contains the entry to be updated.
entryId	Identifies the specific entry within the schema whose values are to be updated.
fieldList	A list of field/value pairs for all the fields to be updated. This list must contain at least one pair or an error will be returned. The fields can be in any order in the list. Any nonexistent or inaccessible field will result in an error return. The datatype of the value must match the datatype of the field (or be NULL). To delete a value for a field, assign the value NULL (AR_DATA_TYPE_NULL) as the value. If the field is a required field in the system, you cannot assign it a NULL value.
getTime	The time at which a get operation on this field was last done.

This parameter is important when considering a shared environment where there may be multiple users changing an entry at the same time. There is no record locking performed by the server. Instead, the use of timestamps allows warnings of possible conflicts with the resolution left up to the user.

The getTime value is checked against the "Modified-date" value of the entry. If the getTime is later than the "Modified-date", the update operation is performed. If the getTime is earlier than the "Modified-date", it is assumed that someone else has changed the record since the record was read and an error is returned indicating that fact. You can then react to this error by either ignoring it and overriding the check (with a getTime of 0) OR you can retrieve the record again to look at the changes made before applying your own.

Assigning a value of 0 causes no check to be performed.

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR:** Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateEntry (3), **ARDeleteEntry** (3), **ARGetEntry** (3), **ARGetEntryStatistics** (3), **ARGetListEntry** (3), **ARMergeEntry** (3), **FreeARFieldValueList** (3), **FreeARStatusList** (3)

ARSetEscalation

NAME

ARSetEscalation – update an existing escalation in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARSetEscalation(control, name, newName, escalationTm, schema, enable, query, actionList, helpText,
        owner, changeDiary, status)
    ARControlStruct          *control;
    ARNameType               name;
    ARNameType               newName;
    AREscalationTmStruct     *escalationTm;
    ARNameType               schema;
    unsigned int             *enable;
    ARQualifierStruct        *query;
    ARFilterActionList       *actionList;
    char                     *helpText;
    ARNameType               owner;
    char                     *changeDiary;
    ARStatusList             *status;
```

DESCRIPTION

ARSetEscalation will update an existing escalation with the indicated name on the specified server. The updates will take effect immediately and will remain in effect until changed or deleted.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the escalation to update.
newName	The new name for the escalation. Remember that all escalation names on a given server must be unique. If you do not want the name of the escalation changed, specify NULL for this parameter.
escalationTm	The checking time of the new escalation. It defines a datetime or frequency of AR System checks for the escalation condition. Escalation time has two formats, one is in seconds as a time interval between checks, the other is a datetime mask specifying on what day of the month or week and at what hour and minute of the day the AR System checks the escalation condition.

If you do not want the escalation option changed, specify NULL for this parameter.

schema

The name of the schema the escalation is linked to. Every escalation must be linked to a schema. Changing the schema will break the connection to the old schema and change it to the new one.

If you do not want the schema changed, specify NULL for this parameter.

enable

A flag with a setting of 0 to indicate that this escalation is to be marked as disabled so it will not be executed or 1 to indicate that the escalation is active and will be checked on the specified time interval. An escalation that is disabled will not perform its condition checks and will not fire.

If you do not want the enable flag changed, specify NULL for this parameter.

query

A qualification that is used to search the specified schema. Any records that match the qualification will have the escalation action performed on them.

If there is no qualifying condition, specify NULL or assign a value of AR_COND_OP_NONE to this value.

If you do not want the query changed, specify NULL for this parameter.

actionList

The set of one or more actions to take when the escalation conditions are met. Every escalation MUST have at least one action and can have up to AR_MAX_ACTIONS.

If you do not want the action list changed, specify NULL for this parameter.

helpText

The help text that is to be associated with the escalation. The help text can be of any length. Existing help text can be eliminated by setting helpText to point to a 0-length string.

If you do not want the help text changed, specify NULL for this parameter.

owner

The new owner for the escalation.

If you do not want to change the owner for the escalation, specify NULL for this parameter.

changeDiary

The additional change diary that is to be associated with the escalation. The added change diary text can be of any length. The new text will be appended to the end of any existing text. Existing text cannot be deleted or changed. The new text is timestamped and user name stamped as it is added to the existing change diary text.

If the change diary is to be left unchanged, a NULL pointer should be supplied for this parameter.

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

n **AR_RETURN_OK**: Operation successful

- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateEscalation (3), **ARDeleteEscalation** (3), **ARGetEscalation** (3), **ARGetListEscalation** (3), **FreeARFilterActionList** (3), **FreeARQualifierStruct** (3), **FreeARStatusList** (3)

ARSetField

NAME

ARSetField – update information about a field in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARSetField (control, schema, fieldId, option, createMode, defaultVal, permissions, limit, displayList,
                helpText, owner, changeDiary, status)
    ARControlStruct          *control;
    ARNameType               schema;
    ARInternalId             fieldId;
    unsigned int             *option;
    unsigned int             *createMode;
    ARValueStruct            *defaultVal;
    ARPermissionList         *permissions;
    ARFieldLimitStruct       *limit;
    ARDisplayList            *displayList;
    char                     *helpText;
    ARNameType               owner;
    char                     *changeDiary;
    ARStatusList             *status;
```

DESCRIPTION

ARSetField will update information about the specified field in the indicated schema on the specified server.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
schema	The name of the schema containing the field to update.
fieldId	The internal ID of the specific field within the schema to update.
option	The option flag indicating whether the field is a required or optional field. The value is one of AR_FIELD_OPTION_REQUIRED, AR_FIELD_OPTION_OPTIONAL, or AR_FIELD_OPTION_SYSTEM. If you do not want the option flag changed, specify NULL for this parameter.
createMode	A flag indicating whether the field is open or protected at create time. An open field is one any user, whether having write access or not, can set during the Submit operation. A protected field is one the user must have been given specific write access to in order set the field during the Submit operation. The value is one of AR_FIELD_OPEN_AT_CREATE or AR_FIELD_PROTECTED_AT_CREATE. If you do not want the create mode changed, specify NULL for this parameter.
defaultVal	The value to use for this field if a new entry is submitted without a value for this field. If a value is specified, its datatype must match the datatype of the field or be AR_DEFAULT_VALUE_NONE to indicate that any existing default value should be cleared. If you do not want the default value changed, specify NULL for this parameter.
permissions	The permissions that have been assigned to this field. The information details which groups have access to the field and what access those groups have. If you assign a blank permissions set, you will receive a warning, AR_WARN_ADMIN_ONLY_ACCESS, indicating that the field can be accessed only by the administrator. The field will be updated as specified. If you do not want the permissions changed, specify NULL for this parameter.
limit	The limits that you want assigned for this field. If new limits are assigned, the datatype of the limits must match the datatype of the field or be set to AR_FIELD_LIMIT_NONE to indicate that there are no limits (any existing limits for this field will be cleared). If you do not want the limits changed, specify NULL for this parameter.
displayList	The display list is an array of entries, each of which describes how the field should be displayed on the screen. Each of these items has an associated character tag to identify it. This tag can be specified during Export operations to get only the definition for a given tag. Each entry contains the tag, the name for the field, the position of the field, and the type of control to use on the screen. If you do not want the displayList changed, specify NULL for this parameter.
helpText	The new help text that is to be associated with the field. If you do not want the helpText changed, specify NULL for this parameter. Existing help text can be eliminated by setting helpText to point to a 0-length string.

owner

The new owner for the field.

If you do not want to change the owner for the field, specify NULL for this parameter.

changeDiary

The additional change diary that is to be associated with the field. The added change diary text can be of any length. The new text will be appended to the end of any existing text. Existing text cannot be deleted or changed. The new text is timestamped and user name stamped as it is added to the existing change diary text.

If the change diary is to be left unchanged, a NULL pointer should be supplied for this parameter.

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateField (3), **ARDeleteField** (3), **ARGetField** (3), **ARGetListField** (3), **FreeARDisplayList** (3),**FreeARFieldLimitStruct** (3), **FreeARPermissionList** (3), **FreeARStatusList** (3), **FreeARValueStruct** (3)

5

ARSetFilter

NAME

ARSetFilter – update an existing filter in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARSetFilter (control, name, newName, order, schema, opSet, enable, query, actionList, helpText, owner,
        changeDiary, status)
    ARControlStruct       *control;
    ARNameType            name;
    ARNameType            newName;
    unsigned int          *order;
    ARNameType            schema;
    unsigned int          *opSet;
    unsigned int          *enable;
    ARQualifierStruct     *query;
    ARFilterActionList    *actionList;
    char                  *helpText;
    ARNameType            owner;
    char                  *changeDiary;
    ARStatusList          *status;
```

DESCRIPTION

ARSetFilter will update an existing filter with the indicated name on the specified server. The updates will take effect immediately and will remain in effect until changed or deleted.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.
name	The name of the filter to update.
newName	The new name for the filter. Remember that all filter names on a given server must be unique. If you do not want the name of the filter changed, specify NULL for this parameter.
order	The new order of the filter. The filter order is a code between 0 and 1000, inclusive. It allows the ordering of filters so that filters with lower orders are executed before filters with higher orders. So, the filter order allows you to specify the order in which filters will be processed and lets you insure that given filters will be performed in the order you desire.

	If you do not want the filter option changed, specify NULL for this parameter.
schema	The name of the schema the filter is linked to. Every filter must be linked to a schema. Changing the schema will break the connection to the old schema and change it to the new one. If you do not want the schema changed, specify NULL for this parameter.
opSet	A bit mask of the operations this filter applies to. This field consists of one or more of the following values OR'ed together: **AR_OPERATION_GET**, **AR_OPERATION_SET**, **AR_OPERATION_CREATE**, **AR_OPERATION_DELETE**, and **AR_OPERATION_MERGE**. If you do not want the operation set changed, specify NULL for this parameter.
enable	A flag with a setting of 0 to indicate that this filter is to be marked as disabled so it will not be executed or 1 to indicate that the filter is active and available for use. A filter that is disabled will not be checked for match during any operation and will not fire. If you do not want the enable flag changed, specify NULL for this parameter.
query	A qualification that is used to test the operation/record being accessed. The operation/values of fields for the record must match this qualification to trigger the filter. If there is no qualifying condition, specify NULL or assign a value of AR_COND_OP_NONE to this value. If you do not want the query changed, specify NULL for this parameter.
actionList	The set of one or more actions to take when the filter conditions are met. Every filter MUST have at least one action and can have up to AR_MAX_ACTIONS. If you do not want the action list changed, specify NULL for this parameter.
helpText	The help text that is to be associated with the filter. The help text can be of any length. Existing help text can be eliminated by setting helpText to point to a 0-length string. If you do not want the help text changed, specify NULL for this parameter.
owner	The new owner for the filter. If you do not want to change the owner for the filter, specify NULL for this parameter.
changeDiary	The additional change diary that is to be associated with the filter. The added change diary text can be of any length. The new text will be appended to the end of any existing text. Existing text cannot be deleted or changed. The new text is timestamped and user name stamped as it is added to the existing change diary text. If the change diary is to be left unchanged, a NULL pointer should be supplied for this parameter.

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateFilter (3), **ARDeleteFilter** (3), **ARGetFilter** (3), **ARGetListFilter** (3), **FreeARFilterActionList** (3), **FreeARQualifierStruct** (3), **FreeARStatusList** (3)

ARSetFullTextInfo

NAME

ARSetFullTextInfo – set full text information for an AR System server

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"

int ARSetFullTextInfo (control, fullTextInfo, status)
    ARControlStruct          *control;
    ARFullTextInfoList       *fullTextInfo;
    ARStatusList             *status;
```

DESCRIPTION

ARSetFullTextInfo updates one or more pieces of information about the AR System server full text environment.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control

The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

fullTextInfo

The information to be updated on the server. If there is an error updating any piece of information, an error will be returned and that information will remain unchanged. Other updates will occur.

Following are the various codes for information that can be updated:

- **AR_FULLTEXTINFO_COLLECTION_DIR**: A character string containing the directory in which the full text collection information is stored.
- **AR_FULLTEXTINFO_STOPWORD**: A structure containing the new list of words to ignore (stopwords) for the full text collection. Existing indexes will NOT be rebuilt to reflect changes. Use any AR_FULLTEXTINFO_REINDEX to force reindexing.
- **AR_FULLTEXTINFO_REINDEX**: Existing indexes WILL NOT be rebuilt to reflect any changes.Use the option AR_FULLTEXTINFO_REINDEX to force reindexing. An integer that specifies old fields be reindexed if set to 0 (AR_FULLTEXT_REINDEX). It takes no action to set to 0.
- **AR_FULLTEXTINFO_CASE_SENSITIVE_SRCH**: An integer indicating whether the search will be performed in a case-sensitive (AR_CASE_SENSITIVE_SEARCH) or case-insensitive (AR_CASE_INSENSITIVE_SEARCH) manner.
- **AR_FULLTEXTINFO_STATE**: An integer indicating whether the full text system is active (AR_FULLTEXT_STATE_ON) or inactive (AR_FULLTEXT_STATE_OFF). If the system is inactive, no indexing is performed for new entries and searching using the full text engine is disabled.

Note: When reactivating full text support, you should consider reindexing all entries to insure old data is properly indexed.

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully

- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARGetFullTextInfo (3), **ARGetServerInfo** (3), **ARSetServerInfo** (3), **FreeARFullTextInfoList** (3), **FreeARStatusList** (3)

ARSetSchema

NAME

ARSetSchema – update information about a schema in the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARSetSchema (control, name, newName, groupList, adminGroupList, getListFields, indexList, helpText,
                 owner, changeDiary, status)
    ARControlStruct          *control;
    ARNameType               name;
    ARNameType               newName;
    ARInternalIdList         *groupList;
    ARInternalIdList         *adminGroupList;
    AREntryListFieldList     *getListFields;
    ARIndexList              *indexList;
    char                     *helpText;
    ARNameType               owner;
    char                     *changeDiary;
    ARStatusList             *status;
```

DESCRIPTION

ARSetSchema will update information about the schema indicated by name on the specified server. Information that can be updated is the global schema information and not information on individual fields (see ARSetField).

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control — The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

name	The name of the schema to update.
newName	The new name for the schema. Remember that all schema names on a given server must be unique. All references to the schema by name (for example, within filters) are updated to reflect the new name. If you do not want to change the name of the schema, specify NULL for this parameter.
groupList	The new groupList definition for the schema. This list of 0 or more groups defines the list of groups whose users are allowed to access this schema. Defining a groupList with 0 items will define a schema that can be accessed only by users with Administrator capability. Defining a groupList that contains the “Public” group (group id 0) will define a schema that can be accessed by all users. If you do not want to change the groupList definition for the schema, specify NULL for this parameter.
adminGroupList	The list of groups whose users have potential SubAdministrator access to this schema. This is the list of groups whose members will have sub-administration rights to the schema and its associated filters, escalations, and active links if the user is also a member of the SubAdministrator group. Defining an adminGroupList with 0 items will define a schema that can be administered only by users with Administrator capability. Defining an adminGroupList that contains the “Public” group (group id 0) will define a schema that can be administered by all users who are members of the SubAdministrator group. If you do not want to change the adminGroupList definition for the schema, specify NULL for this parameter.
getListFields	The new getListFields definition for the schema. This list of 0 or fields and formatting information defines the description that will be returned with the ARGetListEntry call. The maximum size of the fields and separators must be less than or equal to AR_MAX_SDESC_SIZE. Defining a getListFields definition with 0 items specifies that the description will be derived from the Short-Description core field (field id is 8). If you do not want to change the getListFields definition for the schema, specify NULL for this parameter.
indexList	The new indexList definition for the schema. This list of 0 or more indexes defines the list of indexes created in the database on the schema. Indexes can be specified on a single or on multiple columns. You cannot index any diary field or character field with a maximum length over 255 bytes. If you do not want to change the indexList definition for the schema, specify NULL for this parameter.
helpText	The help text that is to be associated with the schema. The help text can be of any length. Existing help text can be eliminated by setting helpText to point to a 0-length string. If the help text is to be left unchanged, a NULL pointer should be supplied for this parameter.

owner

The new owner for the schema.

If you do not want to change the owner for the schema, specify NULL for this parameter.

changeDiary

The additional change diary that is to be associated with the schema. The added change diary text can be of any length. The new text will be appended to the end of any existing text. Existing text cannot be deleted or changed. The new text is timestamped and user name stamped as it is added to the existing change diary text.

If the change diary is to be left unchanged, a NULL pointer should be supplied for this parameter.

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARCreateField (3), **ARCreateSchema** (3), **ARDeleteSchema** (3), **ARGetListSchema** (3), **ARGetSchema** (3), **ARSetField** (3), **FreeAREntryListFieldList** (3), **FreeARIndexList** (3), **FreeARInternalIdList** (3), **FreeARStatusList** (3)

ARSetServerInfo

NAME

ARSetServerInfo – set information for an AR System server

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"

int ARSetServerInfo (control, serverInfo, status)
    ARControlStruct      *control;
    ARServerInfoList     *serverInfo;
    ARStatusList         *status;
```

DESCRIPTION

ARSetServerInfo updates one or more pieces of information about the AR System server environment.

This operation can be performed only by users who have Administrator capabilities within the AR System.

INPUT ARGUMENTS

control

The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

serverInfo

The information to be updated on the server. If there is an error updating any piece of information, an error will be returned and that information will remain unchanged. Other updates will occur.

Following are the various codes for information that can be updated:

- **AR_SERVER_INFO_ALLOW_GUESTS**: An integer flag with 1 indicating that guest users are allowed in the system and 0 indicating they are not allowed. Guest users are users who are not registered with the AR System. If allowed, they can access only data with “Public” view access and submit new entries with fields that have a create mode of “Open”.
- **AR_SERVER_INFO_USE_ETC_PASSWD**: An integer flag with 1 indicating that the /etc/passwd file will be searched if the user is not registered with the AR System and 0 indicating that /etc/passwd will not be searched.
- **AR_SERVER_INFO_XREF_PASSWORDS**: An integer flag with 1 indicating that the system will check passwords in /etc/passwd for any registered user with a blank password in the AR System and 0 indicating there is no cross reference check.

- **AR_SERVER_INFO_DEBUG_MODE**: An integer bitmask that specifies which debugging modes are active in the system (bit 1 is the low order bit): bit 1 specifies SQL tracing, bit 2 specifies Filter tracing, bit 3 specifies User tracing, bit 4 specifies Escalation tracking, and bit 5 specifies API tracing.
- **AR_SERVER_INFO_DB_PASSWORD**: A character string containing the password to assign to the AR System database/tablespace. This is the password for the ARAdmin user. This setting is used only if the underlying database is SYBASE or ORACLE.
- **AR_SERVER_INFO_SET_PROC_TIME**: An integer set to the maximum time to wait for a filter run process operation that is returning a value.
- **AR_SERVER_INFO_EMAIL_FROM**: A character string containing the name of the user who will be specified as the source of all email notifications.
- **AR_SERVER_INFO_SQL_LOG_FILE**: A character string containing the filename (relative or absolute) where the information from the SQL tracing operation is placed.
- **AR_SERVER_INFO_FLOAT_TIMEOUT**: An integer noting the number of hours before a floating license will automatically timeout.
- **AR_SERVER_INFO_UNQUAL_QUERIES**: An integer flag with 1 indicating that the server will respond to unqualified queries from users and 0 indicating that unqualified queries will return an error.
- **AR_SERVER_INFO_FILTER_LOG_FILE**: A character string containing the filename (relative or absolute) where the information from the Filter tracing operation is placed.
- **AR_SERVER_INFO_USER_LOG_FILE**: A character string containing the filename (relative or absolute) where the information from the User tracing operation is placed.
- **AR_SERVER_INFO_MAX_ENTRIES**: An integer that identifies the maximum number of entries that will be returned in response to a single ARGetListEntry call. This value works in combination with the value that can be defined by the user in the call to **ARGetListEntry** with the minimum of the two values taking precedence in a given call.
- **AR_SERVER_INFO_MAX_F_DAEMONS**: An integer that specifies the maximum number of “fast” servers that will be run in a multi-process server environment. (This option is effective only if the multi-process server option has been activated for the software.)

- **AR_SERVER_INFO_MAX_L_DAEMONS:** An integer that specifies the maximum number of “list” servers that will be run in a multi-process server environment. (This option is effective only if the multi-process server option has been activated for the software.)
- **AR_SERVER_INFO_ESCALATION_LOG_FILE**: A character string containing the filename (relative or absolute) where the information from the Escalation tracing operation is placed.
- **AR_SERVER_INFO_ESCL_DAEMON**: An integer that specifies whether a separate escalation servers will be run in a multi-process server environment. If set to 0, no separate escalation server will run. If set to 1, a separate escalation server will run. (This option is effective only if the multi-process server option has been activated for the software.)
- **AR_SERVER_INFO_SUBMITTER_MODE**: An integer flag that when set to AR_SUBMITTER_MODE_LOCKED indicates that the value in the Submitter field will be locked at submit time and not changeable thereafter and the Submitter is allowed to change values within permissions with or without a license and when set to AR_SUBMITTER_MODE_CHANGEABLE indicates that the value in the Submitter field can be changed at any time within permissions, but that any change using the Submitter group permissions requires a license.

NOTE: A change to this value will not take affect until the server is next restarted.

- **AR_SERVER_INFO_API_LOG_FILE**: A character string containing the filename (relative or absolute) where the information from the API tracing operation is placed.
- **AR_SERVER_INFO_FTEXT_TIMEOUT**: An integer noting the number of hours before a floating full text license will automatically timeout.
- **AR_SERVER_INFO_DS_RPC_SOCKET**: An integer that holds the socket number of the RPC socket being used by the distributed server environment. If NULL, the distributed server is set to use the default socket. If set, the specific socket will be used. Legal values are 390600 and 390680 through 390694.
- **AR_SERVER_INFO_DS_LOG_FILE:** A character string containing the filename (relative or absolute) where the information from the distributed server tracing operation is placed.
- **AR_SERVER_INFO_SUPPRESS_WARN**: A character string containing a list of one or more note/warning numbers (separated by spaces). The messages that are tied to these numbers will be suppressed by the server.

- **AR_SERVER_INFO_SAVE_LOGIN**: An integer value that indicates whether to save login information in client tools and who controls that saving. Can be set to one of the following:
  0 - User controlled (default), 1 - Admin controlled, set to save the login information, 2 - Admin controlled, set to not save the login information.

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARGetFullTextInfo (3), **ARGetServerInfo** (3), **ARSetFullTextInfo** (3), **FreeARServerInfoList** (3), **FreeARStatusList** (3)

ARTermination

NAME

ARTermination – terminates interaction with the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"
int ARTermination (status)
        ARStatusList            *status;
```

DESCRIPTION

ARTermination serves to terminate the programs interaction with the AR System. For many systems, this call performs no work, while in others it performs some cleanup operations for the system. It should always be called in case it is needed by the environment.

In an environment using Floating licenses, it is essential that this routine be called. This procedure will release the floating license token when access to the AR System is terminated. Otherwise, the floating license token is not released until the configured timeout interval.

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

ARInitialization (3), **FreeARStatusList** (3)

5

ARVerifyUser

NAME

ARVerifyUser – verify user within the AR System

SYNOPSIS

```
#include "ar.h"
#include "arerrno.h"
#include "arextern.h"
#include "arstruct.h"

int ARVerifyUser (control, adminFlag, subAdminFlag, customFlag, status)
    ARControlStruct     *control;
    ARBoolean           *adminFlag;
    ARBoolean           *subAdminFlag;
    ARBoolean           *customFlag;
    ARStatusList        *status;
```

DESCRIPTION

ARVerifyUser will check the user against the user cache on the specified AR System server. The routine is used to check whether the user is a registered user of the AR System.

INPUT ARGUMENTS

control	The control record for the operation. It contains information about the user requesting the operation and where that operation is to be performed. The user and server fields must be supplied in this structure.

RETURN VALUES

adminFlag	A Boolean flag indicating whether the user specified is a member of the Administrator group (has superuser type access to AR System). If the user is invalid or unknown to the system, this value will be FALSE. If this flag is not desired, a NULL value can be passed for this parameter and the value will not be returned.
adminFlag	A Boolean flag indicating whether the user specified is a member of the SubAdministrator group (has qualified superuser type access to AR System). If the user is invalid or unknown to the system, this value will be FALSE. If this flag is not desired, a NULL value can be passed for this parameter and the value will not be returned.

customFlag

A Boolean flag indicating whether the user specified is a member of the Customize group. If the user is invalid or unknown to the system, this value will be FALSE. If the user is a member of the Administrator group, this flag will always be TRUE. The Customize group is an advisory group that indicates whether a user is allowed to perform customizations within the system. It is interpreted as desired by any interface to the system.

If this flag is not desired, a NULL value can be passed for this parameter and the value will not be returned.

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **AR_RETURN_OK**: Operation successful
- **AR_RETURN_WARNING**: Warning during process but operation completed successfully
- **AR_RETURN_ERROR**: Operation failed due to error
- **AR_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **AR_RETURN_BAD_STATUS**: Status parameter was bad

If the user is known to the system but the password is not valid, the error AR_ERROR_PASSWORD_MISMATCH is returned. If the user is unknown to the system and guest users are allowed, this is not considered an error, but the warning AR_WARN_NO_SUCH_USER is returned. This is done as the AR System allows a limited level of access to the system to unknown users (guests). The warning can be trapped and treated as an error condition if you do not wish to allow unknown users. If the user is unknown and guest users are NOT allowed, the error AR_ERROR_NO_SUCH_USER is returned.

SEE ALSO

FreeARStatusList (3)

5

FreeAR

NAME

FreeAR – free space in one of the AR System data structures

SYNOPSIS

```
#include "ar.h"
#include "arfree.h"

void FreeARActiveLinkActionList (value, freeStruct)
        ARActiveLinkActionList      *value;
        ARBoolean                   freeStruct;

void FreeARActiveLinkActionStruct (value, freeStruct)
        ARActiveLinkActionStruct    *value;
        ARBoolean                   freeStruct;

void FreeARArithOpAssignStruct (value, freeStruct)
        ARArithOpAssignStruct       *value;
        ARBoolean                   freeStruct;

void FreeARArithOpStruct (value, freeStruct)
        ARArithOpStruct             *value;
        ARBoolean                   freeStruct;

void FreeARAssignFieldStruct (value, freeStruct)
        ARAssignFieldStruct         *value;
        ARBoolean                   freeStruct;

void FreeARAssignStruct (value, freeStruct)
        ARAssignStruct              *value;
        ARBoolean                   freeStruct;

void FreeARCharMenuItemStruct (value, freeStruct)
        ARCharMenuItemStruct        *value;
        ARBoolean                   freeStruct;

void FreeARCharMenuList (value, freeStruct)
        ARCharMenuList              *value;
        ARBoolean                   freeStruct;

void FreeARCharMenuStruct (value, freeStruct)
        ARCharMenuStruct            *value;
        ARBoolean                   freeStruct;

void FreeARDDEStruct(value, freeStruct)
        ARDDEStruct                 *value;
        ARBoolean                   freeStruct;

void FreeARDiaryList (value, freeStruct)
        ARDiaryList                 *value;
        ARBoolean                   freeStruct;
```

void FreeARDisplayList (value, freeStruct)
ARDisplayList *value;
ARBoolean freeStruct;

void FreeAREntryIdList (value, freeStruct)
AREntryIdList *value;
ARBoolean freeStruct;

void FreeAREntryListFieldList (value, freeStruct)
AREntryListFieldList *value;
ARBoolean freeStruct;

void FreeAREntryListList (value, freeStruct)
AREntryListList *value;
ARBoolean freeStruct;

void FreeARFieldAssignList (value, freeStruct)
ARFieldAssignList *value;
ARBoolean freeStruct;

void FreeARFieldAssignStruct (value, freeStruct)
ARFieldAssignStruct *value;
ARBoolean freeStruct;

void FreeARFieldLimitStruct (value, freeStruct)
ARFieldLimitStruct *value;
ARBoolean freeStruct;

void FreeARFieldValueList (value, freeStruct)
ARFieldValueList *value;
ARBoolean freeStruct;

void FreeARFieldValueStruct (value, freeStruct)
ARFieldValueStruct *value;
ARBooleanfreeStruct;

void FreeARFieldValueOrArithStruct (value, freeStruct)
ARFieldValueOrArithStruct *value;
ARBoolean freeStruct;

void FreeARFilterActionList (value, freeStruct)
ARFilterActionList *value;
ARBoolean freeStruct;

void FreeARFilterActionStruct (value, freeStruct)
ARFilterActionStruct *value;
ARBoolean freeStruct;

void FreeARFullTextInfoList (value, freeStruct)
ARFullTextInfoList *value;
ARBoolean freeStruct;

void FreeARFullTextInfoRequestList (value, freeStruct)
ARFullTextInfoRequestList *value;
ARBoolean freeStruct;

```
void FreeARFunctionAssignStruct (value, freeStruct)
        ARFunctionAssignStruct          *value;
        ARBoolean                       freeStruct;
void FreeARGroupInfoList (value, freeStruct)
        ARGroupInfoList                 *value;
        ARBoolean                       freeStruct;
void FreeARIndexList (value, freeStruct)
        ARIndexList                     *value;
        ARBoolean                       freeStruct;
void FreeARInternalIdList (value, freeStruct)
        ARInternalIdList                *value;
        ARBoolean                       freeStruct;
void FreeARNameList (value, freeStruct)
        ARNameList                      *value;
        ARBoolean                       freeStruct;
void FreeARPermissionList (value, freeStruct)
        ARPermissionList                *value;
        ARBoolean                       freeStruct;
void FreeARQualifierStruct (value, freeStruct)
        ARQualifierStruct               *value;
        ARBoolean                       freeStruct;
void FreeARRelOpStruct (value, freeStruct)
        ARRelOpStruct                   *value;
        ARBoolean                       freeStruct;
void FreeARServerInfoList (value, freeStruct)
        ARServerInfoList                *value;
        ARBoolean                       freeStruct;
void FreeARServerInfoRequestList (value, freeStruct)
        ARServerInfoRequestList         *value;
        ARBoolean                       freeStruct;
void FreeARServerNameList (value, freeStruct)
        ARServerNameList                *value;
        ARBoolean                       freeStruct;
void FreeARSortList (value, freeStruct)
        ARSortList                      *value;
        ARBoolean                       freeStruct;
void FreeARStatusList (value, freeStruct)
        ARStatusList                    *value;
        ARBoolean                       freeStruct;
void FreeARStatusHistoryList (value, freeStruct)
        ARStatusHistoryList             *value;
        ARBoolean                       freeStruct;
```

void FreeARStructItemList (value, freeStruct)
ARStructItemList *value;
ARBoolean freeStruct;

void FreeARUserInfoList (value, freeStruct)
ARUserInfoList *value;
ARBoolean freeStruct;

void FreeARUserLicenseList (value, freeStruct)
ARUserLicenseList *value;
ARBoolean freeStruct;

void FreeARValueList (value, freeStruct)
ARValueList *value;
ARBoolean freeStruct;

void FreeARValueStruct (value, freeStruct)
ARValueStruct *value;
ARBoolean freeStruct;

DESCRIPTION

The **FreeAR** routines take a pointer to one of the AR structures and free all the nested space within the structure. They can optionally free the top level structure space if requested.

Any structure that is returned from the AR API calls that has allocated space can be completely freed by a call to this routine.

INPUT ARGUMENTS

value — A pointer to the structure whose space is to be freed.

freeStruct — A flag indicating whether to free the top level value structure or not. If this flag is TRUE, the top level structure is freed. If FALSE, the top level is *not* freed but its values are set to an "empty" state.

This flag *must not* be set to TRUE unless the top level structure pointed to by the value parameter was dynamically allocated.

SEE ALSO

FreeNT (3)

5

FreeNT

NAME

FreeNT – free space in one of the Notification system data structures

SYNOPSIS

```
#include "nt.h"
#include "ntfree.h"
void FreeNTNameList (value, freeStruct)
        NTBoolean               freeStruct;
void FreeNTServerNameList (value, freeStruct)
        NTServerNameList        *value;
        NTBoolean               freeStruct;
void FreeNTStatusList (value, freeStruct)
        NTStatusList            *value;
        NTBoolean               freeStruct;
```

DESCRIPTION

The **FreeNT** routines take a pointer to one of the Notification structures and frees all the nested space within the structure. They can optionally free the top level structure space if requested.

Any structure that is returned from the Notification API calls that has allocated space can be completely freed by a call to this routine.

INPUT ARGUMENTS

value	A pointer to the structure whose space is to be freed.
freeStruct	A flag indicating whether to free the top level value structure or not. If this flag is TRUE, the top level structure is freed. If FALSE, the top level is *not* freed but its values are set to an "empty" state. This flag *must not* be set to TRUE unless the top level structure pointed to by the value parameter was dynamically allocated.

SEE ALSO

FreeAR (3)

NTCheckRegisteredClient

NAME

NTCheckRegisteredClient – check with an Notification System client to see if a user is registered

SYNOPSIS

```
#include "nt.h"
#include "nterrno.h"
#include "ntcextrn.h"

int NTCheckRegisteredClient (user, status)
        char                    *clientHost;
        NTNameType              user;
        NTStatusList            *status;
```

DESCRIPTION

NTCheckRegisteredClient checks with the Notification System client to see if the indicated user is registered.

This call is really an internal call between the Notification System client and Notification System server. In general, it should not be called.

INPUT ARGUMENTS

clientHost — The host name for the machine where the Notification System client that you want to check is running.

user — The name of the user we are checking on.

RETURN VALUES

status — A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **NT_RETURN_OK**: Operation successful
- **NT_RETURN_WARNING**: Warning during process but operation completed successfully
- **NT_RETURN_ERROR**: Operation failed due to error
- **NT_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **NT_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

NTDeregisterClient (3), **NTRegisterClient** (3), **FreeNTStatusList** (3)

NTDeregisterClient

NAME

NTDeregisterClient – cancel registration with the Notification System client process

SYNOPSIS

```
#include "nt.h"
#include "nterrno.h"
#include "ntcextrn.h"

int NTDeregisterClient (user, password, filename, status)
        NTNameType          user;
        NTNameType          password;
        char                *filename;
        NTStatusList        *status;
```

DESCRIPTION

NTDeregisterClient will close registration for this process with the Notification System client. The combination of user and filename identifies which instance to deregister.

INPUT ARGUMENTS

user	The name of the user deregistering with the Notification System. The user will be removed from the active list and no more notifications delivered.
password	The password for the user. If the user is a registered user of the system, the password is used to validate that user. This prevents another user from deregistering a user so that the user misses notifications.
filename	The filename of a named pipe that will be used for communication between this process and the Notification System client. This insures that the correct instance of the registration is closed in case the user is registered with more than one Notification tool running.

RETURN VALUES

status	A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **NT_RETURN_OK**: Operation successful
- **NT_RETURN_WARNING:** Warning during process but operation completed successfully
- **NT_RETURN_ERROR**: Operation failed due to error
- **NT_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **NT_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

NTRegisterClient (3), **FreeNTStatusList** (3)

NTDeregisterServer

NAME

NTDeregisterServer – cancel registration with the Notification System server process

SYNOPSIS

```
#include "nt.h"
#include "nterrno.h"
#include "ntsextrn.h"

int NTDeregisterServer (serverHost, user, password, status)
        char            *serverHost;
        NTNameType      user;
        NTNameType      password;
        NTStatusList    *status;
```

DESCRIPTION

NTDeregisterServer will close registration for this process with the Notification System server. The combination of user and hostname identifies which instance to deregister.

This call is really an internal call between the Notification System client and Notification System server. In general, it should not be called.

INPUT ARGUMENTS

serverHost	The name of the host on which the Notification System server is running.
user	The name of the user deregistering with the Notification System. The user will be removed from the active list and no more notifications delivered.

password — The password for the user. If the user is a registered user of the system, the password is used to validate that user. This prevents another user from deregistering a user so that the user misses notifications.

RETURN VALUES

status — A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **NT_RETURN_OK**: Operation successful
- **NT_RETURN_WARNING:** Warning during process but operation completed successfully
- **NT_RETURN_ERROR**: Operation failed due to error
- **NT_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **NT_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

NTRegisterServer (3), **FreeNTStatusList** (3)

NTGetListServer

NAME

NTGetListServer – retrieve a list of servers accessible from the current machine

SYNOPSIS

```
#include "nt.h"
#include "nterrno.h"
#include "ntsextrn.h"

int NTGetListServer (serverList, status)
        NTServerNameList        *serverList;
        NTStatusList            *status;
```

DESCRIPTION

NTGetListServer retrieves a list of all the servers that are accessible from the current machine. It gets the list of servers by processing the Notification directory file `/etc/ar` (in UNIX) or `<ar_config_dir>\ar` (in the Windows NT server), retrieving all registered Notification servers.

Access to the server list is available to all users.

This call is really an internal call between the Notification System client and Notification System server. In general, it should not be called.

RETURN VALUES

serverList A list of all the servers that are registered.

If no servers are registered, the routine will return successfully but this list will contain 0 names.

status A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **NT_RETURN_OK**: Operation successful
- **NT_RETURN_WARNING**: Warning during process but operation completed successfully
- **NT_RETURN_ERROR**: Operation failed due to error
- **NT_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **NT_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

FreeNTStatusList (3), **FreeNTServerNameList** (3)

NTInitializationClient

NAME

NTInitializationClient – initialize interaction with the Notification System client

SYNOPSIS

```
#include "nt.h"
#include "nterrno.h"
#include "ntcextrn.h"

int NTInitializationClient (status)
        NTStatusList                *status;
```

DESCRIPTION

NTInitializationClient serves to initialize the program for interaction with the Notification system client. For many systems, this call performs no work, while in others it establishes an initial state for the system. It should always be

called in case it is needed by the environment.

RETURN VALUES

status — A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **NT_RETURN_OK**: Operation successful
- **NT_RETURN_WARNING**: Warning during process but operation completed successfully
- **NT_RETURN_ERROR**: Operation failed due to error
- **NT_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **NT_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

NTTerminationClient (3), **FreeNTStatusList** (3)

NTInitializationServer

NAME

NTInitializationServer – initialize interaction with the Notification System server

SYNOPSIS

```
#include "nt.h"
#include "nterrno.h"
#include "ntsextrn.h"
int NTInitializationServer (status)
        NTStatusList            *status;
```

DESCRIPTION

NTInitializationServer serves to initialize the program for interaction with the Notification system server. For many systems, this call performs no work, while in others it establishes an initial state for the system. It should always be called in case it is needed by the environment.

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **NT_RETURN_OK**: Operation successful
- **NT_RETURN_WARNING**: Warning during process but operation completed successfully
- **NT_RETURN_ERROR:** Operation failed due to error
- **NT_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **NT_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

NTTerminationServer (3), **FreeNTStatusList** (3)

NTNotificationClient

NAME

NTNotificationClient – deliver a notification to the indicated Notification System client

SYNOPSIS

```
#include "nt.h"
#include "nterrno.h"
#include "ntcextrn.h"

int NTNotificationClient (clientHost, user, notifyText, notifyCode, notifyCodeText, status)
        char            *clientHost;
        NTNameType      user;
        char            *notifyText;
        int             notifyCode;
        char            *notifyCodeText;
        NTStatusList    *status;
```

DESCRIPTION

NTNotificationClient will deliver a notification to the indicated client. The client will then process the message to deliver it on to the processes who have registered with the Notification System client.

This call is really an internal call between the Notification System client and Notification System server. In general, it should not be called.

INPUT ARGUMENTS

clientHost	The host name for the machine where the Notification System client is running.
user	The name of the user the notification is for.
notifyText	The full text of the notification.
notifyCode	A code that indicates the type and source of the notification. The Notification System does not specify these codes, they are solely between the sender and receiver of the notification.
notifyCodeText	Text that is associated with the notifyCode. If there is no associated text, this parameter should be set to NULL.

RETURN VALUES

status	A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later. The return of the function itself is an integer indicating the success of the call. The return will be one of the following values: n **NT_RETURN_OK**: Operation successful n **NT_RETURN_WARNING**: Warning during process but operation completed successfully n **NT_RETURN_ERROR**: Operation failed due to error n **NT_RETURN_FATAL**: Operation failed; the status array may or may not contain any information n **NT_RETURN_BAD_STATUS:** Status parameter was bad

SEE ALSO

NTDeregisterClient (3), **NTRegisterClient** (3), **FreeNTStatusList** (3)

NTNotificationServer

NAME

NTNotificationServer – deliver a notification to the indicated Notification System server

SYNOPSIS

```
#include "nt.h"
#include "nterrno.h"
#include "ntsextrn.h"

int NTNotificationServer (serverHost, user, notifyText, notifyCode, notifyCodeText, status)
    char            *serverHost;
    NTNameType      user;
    char            *notifyText;
    int             notifyCode;
    char            *notifyCodeText;
    NTStatusList    *status;
```

DESCRIPTION

NTNotificationServer will deliver a notification to the indicated server. The server will then process the message to deliver it on to the Notification System client processes who have registered with the Notification System server.

INPUT ARGUMENTS

serverHost — The host name for the machine where the Notification System server is running.

user — The name of the user the notification is for.

notifyText — The full text of the notification.

notifyCode — A code that indicates the type and source of the notification. The Notification System does not specify these codes, they are solely between the sender and receiver of the notification.

As used by Remedy Corporation, there are two codes defined: 1 and 2. Code 1 indicates that the notification is from the NT System. Code 2 indicates that the notification is from the AR System. See below for the format of the text that is used in each of these cases.

notifyCodeText — Text that is associated with the notifyCode. If there is no associated text, this parameter should be set to NULL.

As used by Remedy Corporation, the two codes noted above use the following format for the **notifyCodeText**:

The text for Notification System notifications is simply a character string containing any further details of the notification being sent. This is usually a qualifying error message with further details of the problem generating the notification.

The text for AR System notifications is a formatted string consisting of three parts. The first two are padded to the maximum possible size with the value left-justified, the third is simply the value itself with no need for padding.

- **Entry-Id** The entry ID of the entry that is associated to the notification (padded to AR_MAX_ENTRYID_SIZE)
- **Schema** The name of the schema that contains the associated entry (padded to AR_MAX_NAME_SIZE)
- **Server** The name of the server where the schema is located

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **NT_RETURN_OK**: Operation successful
- **NT_RETURN_WARNING**: Warning during process but operation completed successfully
- **NT_RETURN_ERROR:** Operation failed due to error
- **NT_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **NT_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

NTDeregisterServer (3), **NTRegisterServer** (3), **FreeNTStatusList** (3)

NTRegisterClient

NAME

NTRegisterClient – register with the Notification client process to receive notifications

SYNOPSIS

```
#include "nt.h"
#include "nterrno.h"
#include "ntcextrn.h"

int NTRegisterClient (user, password, filename, status)
        NTNameType          user;
        NTNameType          password;
        char                *filename;
        NTStatusList        *status;
```

DESCRIPTION

NTRegisterClient will register this process with the Notification System client. The user will be validated and registered with each of the Notification System servers on the network. When any message is received by the Notification client, it will be returned in the named pipe specified in the call.

INPUT ARGUMENTS

user

The name of the user registering with the Notification System. All notifications that are directed to this user will be delivered to the pipe specified.

password

The password for the user. If the user is a registered user of the system, the password is used to validate that user. This prevents registration and receipt of notifications by people who are simply posing as the user.

filename

The filename of a named pipe that will be used for communication between this process and the NT System client. The file should have already been opened for read access by this process. The NT System client will open it with write access and will write any notifications received for the specified user into the pipe.

Every message delivered on this pipe will be exactly NT_MAX_FULL_MESSAGE bytes. The actual formatted string will be null terminated and any bytes after that are unused and should be ignored.

The format of the message delivered is as follows. Each of the items in the string is separated by a separator character (NT_NOTIFY_STRING_SEP).

- **notifyCode** - Code indicating the source of the notification. At this time, there are three sources identified:
  0 - The notify is a heartbeat and the message should be ignored,
  1 - Notification from the Notification System, and
  2 - Notification from the AR System.
- **notifyTextLen** - Length in bytes of the notify text
- **notifyText** - The actual notify text

- **notifyCodeText** - This text is optional. If there is additional text, it will be present; otherwise, this text is omitted. For notifications with a notifyCode of 1, this text is details of an error message. For notifications with a notifyCode of 2, this is a formatted string that consists of the entry ID for AR_MAX_ENTRYID_SIZE bytes (left-justified), the schema name for AR_MAX_NAME_SIZE bytes (left-justified), and the server name.

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **NT_RETURN_OK**: Operation successful
- **NT_RETURN_WARNING**: Warning during process but operation completed successfully
- **NT_RETURN_ERROR**: Operation failed due to error
- **NT_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **NT_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

NTDeregisterClient (3), **FreeNTStatusList** (3)

NTRegisterServer

NAME

NTRegisterServer – register with the Notification server process to receive notifications

SYNOPSIS

```
#include "nt.h"
#include "nterrno.h"
#include "ntsextrn.h"

int NTRegisterServer (serverHost, user, password, status)
        char            *serverHost;
        NTNameType      user;
        NTNameType      password;
        NTStatusList    *status;
```

DESCRIPTION

NTRegisterServer will register this process with the Notification System server. The user will be validated and registered. When any message is received by the Notification server, it will be forwarded to the registered machine.

This call is really an internal call between the Notification System client and Notification System server. In general, it should not be called.

INPUT ARGUMENTS

serverHost — The name of the machine on which the Notification System server is to be run.

user — The name of the user registering with the Notification System. All notifications that are directed to this user will be delivered to the machine the process making this call is run on (through the NTNotificationClient call).

password — The password for the user. If the user is a registered user of the system, the password is used to validate that user. This prevents registration and receipt of notifications by people who are simply posing as the user.

RETURN VALUES

status — A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **NT_RETURN_OK**: Operation successful
- **NT_RETURN_WARNING**: Warning during process but operation completed successfully
- **NT_RETURN_ERROR**: Operation failed due to error
- **NT_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **NT_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

NTDeregisterServer (3), **FreeNTStatusList** (3)

NTTerminationClient

NAME

NTTerminationClient – terminates interaction with the Notification System client

SYNOPSIS

```
#include "nt.h"
#include "nterrno.h"
#include "ntcextrn.h"
int NTTerminationClient (status)
        NTStatusList                    *status;
```

DESCRIPTION

NTTerminationClient serves to terminate the program's interaction with the Notification system client. For many systems, this call performs no work, while in others it performs some cleanup operations for the system. It should always be called in case it is needed by the environment.

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **NT_RETURN_OK**: Operation successful

- **NT_RETURN_WARNING**: Warning during process but operation completed successfully
- **NT_RETURN_ERROR**: Operation failed due to error
- **NT_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **NT_RETURN_BAD_STATUS**: Status parameter was bad

SEE ALSO

NTInitializationClient (3), **FreeNTStatusList** (3)

NTTerminationServer

NAME

NTTerminationServer – terminates interaction with the Notification System server

SYNOPSIS

```
#include "nt.h"
#include "nterrno.h"
#include "ntsextrn.h"

int NTTerminationServer (status)
        NTStatusList                    *status;
```

DESCRIPTION

NTTerminationServer serves to terminate the program's interaction with the Notification system server. For many systems, this call performs no work, while in others it performs some cleanup operations for the system. It should always be called in case it is needed by the environment.

RETURN VALUES

status

A list of all the Notes/Warnings/Errors generated from the call. This is a list of 0 or more notes each containing the type of error, its code, and a message generated using the language specified in the control record. The more serious errors are listed first in the list with lesser warnings and notes listed later.

The return of the function itself is an integer indicating the success of the call. The return will be one of the following values:

- **NT_RETURN_OK**: Operation successful
- **NT_RETURN_WARNING**: Warning during process but operation completed successfully
- **NT_RETURN_ERROR**: Operation failed due to error

- **NT_RETURN_FATAL**: Operation failed; the status array may or may not contain any information
- **NT_RETURN_BAD_STATUS:** Status parameter was bad

SEE ALSO

NTInitializationServer (3), **FreeNTStatusList** (3)

File Formats

ar

NAME

ar – directory file to AR and Notification System servers

SYNOPSIS

/etc/ar (for UNIX)

<*ar_config_dir*>\ar (for Windows NT)

DESCRIPTION

The file **/etc/ar** for UNIX and **<*ar_config_dir*>\ar** for Windows NT contains a list of all the AR and Notification System servers that are accessible from the current machine. This file is processed through the ARGetListServer and NTGetListServer routines to return lists of all accessible servers. In order to be recognized by any of the AR or Notification System tools, a server must be registered in this file. This file is required on all machines serving as either a server or client for the AR and/or Notification systems.

Any line beginning with a **#** in column 1 is treated as a comment line and ignored.

An entry in this file is composed of two space or tab separated fields:

server-name ***server-type-list***

The *server-name* field is the name of a machine that is acting as a server in the environment. This name is resolved to a network location using the **/etc/hosts** file on the local machine.

The *server-type-list* field contains a semicolon-separated list of keywords indicating the type of server supported on that machine. Following is a list of all the legal keywords:

AR Server for the Action Request System.

NT Server for the Notification System.

EXAMPLES

The following configuration file:

```
# directory file for AR/NT servers
starlight       AR;NT
capricorn       AR
jason           NT
```

registers three servers: two for the AR System and two for the Notification system. The first, starlight, is registered as a server for both the Action Request and Notification Systems. The second and third, capricorn and jason, are registered for the Action Request System and Notification System, respectively.

SEE ALSO

ARGetListServer (3), **NTGetListServer** (3)

5

ar.conf

NAME

ar.conf – configuration file for the Action Request System

SYNOPSIS

/etc/ar.conf
<*ar_install_dir*>\ar.cfg (Windows NT)

DESCRIPTION

The file **/etc/ar.conf** or **<ar_install_dir>\ar.cfg** contains configuration information used by the Action Request System. This file is read by any process needing a piece of configuration information about the server. It is located only on systems that are serving as servers for the Action Request System.

Information from this file can be retrieved using the API call **ARGetServerInfo** and can be set using the call **ARSetServerInfo**. If set using this call, all changes take affect immediately. If set by manually changing the file, changes do not take affect until the **arserverd** process is restarted.

Any line beginning with a **#** in column 1 is treated as a comment line and ignored.

An entry in this file is composed of a header followed by a value that corresponds to the header. The header and value are separated by any number of spaces or tabs.

header ***value***

The recognized *header* and *value* pairs are described below. Any other line is simply ignored.

API-Log-File The name of the file to be used if API tracing (see **Debug-Mode:**) is activated.

Allow-Guest-Users Flag indicating whether the AR System server will accept guest users. A guest user is a user that is not registered with the AR System. If guest users are allowed, the user is accepted as a user without permissions but they can do basic operations in the system and submit entries to schemas where open access at submit time is allowed. If guest users are disallowed, unregistered users are allowed no access.

Legal values for this field are **T** and **F**. The default if neither is specified is T (allow guest users).

Allow-Unqual-Queries Flag indicating whether the AR System server will allow unqualified queries (ARGetListEntry calls with NULL for the qualifier or a qualifier with operation AR_COND_OP_NONE). Queries that match and return ALL the data in the database are potential sources of performance problems in the system. They will match and return ALL the entries for a given schema which is a costly operation for large schemas. Note: You can use a dummy qualification like 1 = 1 to allow queries for all entries when this flag is set to False.

Legal values for this field are **T** and **F**. The default if neither is specified is T (allow unqualified queries).

Case-Insensitive-Search: Code indicating whether the AR System server will perform case-insensitive (1) or case-sensitive (2) searches when using the full text search engine.

Legal values for this field are **1** and **2**. The default if neither is specified is 1 (case-insensitive).

Collection-directory: Directory that is the collection directory for the full text engine. All the full text indexes recorded for the system are stored in this directory. If th

2ere are no fields that are full text indexed, this directory is not used.

Crossref-Blank-Password: Flag indicating whether the AR System server will cross-reference from the AR System to the */etc/passwd* file for password checking if the user in the AR System has no password. This flag allows the definition of users in the AR System to define group membership and allow registration of other support information while keeping a tie to the */etc/passwd* file for passwords.

Legal values for this field are **T** and **F**. The default if neither is specified is F (disallow blank password cross-reference).

Dbhome-directory: Directory that is the home directory for the relational database system that is being used by the Action Request System. If the flat file database is being used, this entry is not used.

Db-password: If you are using one of the SQL databases that allows users to be created (SYBASE or ORACLE), this is the database password that is being used for the ARSystem database/tablespace. It is stored in encrypted form. You can change the database password using the **ARSetServerInfo** call.

If you change the password directly, you must enter the password you provided in clear-text after this keyword, then change the password (maybe to the same value) using the AR System utilities to have it encrypted.

Debug-mode: Bitmask setting indicating debug modes that are possible for the AR System servers.

If bit 1 (the low-order bit) is set, SQL tracing on the arserverd process is activated. By default, tracing will be performed to a file named arsql.log in the directory indicated by the **Server-directory:** label. This name can be overridden by using the **SQL-Log-File:** label.

If bit 2 is set, Filter tracing on the arserverd process is activated. By default, tracing will be performed to a file named arfilter.log in the directory indicated by the **Server-directory:** label. This name can be overridden by using the **Filter-Log-File:** label.

If bit 3 is set, User tracing on the arserverd process is activated. By default, tracing will be performed to a file named aruser.log in the directory indicated by the **Server-directory:** label. This name can be overridden by using the **User-Log-File:** label.

If bit 4 is set, Escalation tracing on the arserverd process is activated. By default, tracing will be performed to a file named arescl.log in the directory indicated by the **Server-directory:** label. This name can be overridden by using the **Escalation-Log-File:** label.

If bit 5 is set, API tracing on the arserverd process is activated. By default, tracing will be performed to a file named arapi.log in the directory indicated by the **Server-directory:** label. This name can be overridden by using the **API-Log-File:** label.

If bit 16 is set, Distributed Server tracing on the arservdsd process is activated. By default, tracing will be performed to a file named ardist.log in the directory indicated by the **Server-directory:** label. This name can be overridden by using the **Distrib-Log-File:** label. (Note that this tracing is meaningful only if the Distributed Server Option is licensed.)

Email-Delivery-System: For NCR and Motorola platforms, there are two major subsystems for delivery of mail: **rmail** and **sendmail**. The default for NCR is **rmail** and for Motorola is **sendmail**. For all other platforms, this option is ignored and **sendmail** is used.

Email-Notify-From: Use the value supplied as the "From" name for any email notifications that are generated from Filters within the AR System where there is no value supplied in the Subject line.

This value can only be used by "trusted" email users. To check on or add new trusted users, refer to documentation on the mail configuration file /etc/sendmail.cf.

Escalation-Log-File: The name of the file to be used if escalation tracing (see **Debug-Mode:**) is activated.

Escalation-Daemon: Flag indicating whether this server runs a private escalation daemon. The server must be licensed to run multi-process.

Legal values for this field are **T** and **F**. The default if neither is specified is F (don't run a separate escalation server).

Filter-Log-File: The name of the file to be used if filter tracing (see **Debug-Mode:**) is activated.

FTS-Debug-Mode: Trace level for the **arservftd** process.

If bit 1 is set (the default), logging for all VDK failure messages in the arft.log file is activated.

If bit 2 is set, logging for incoming index commands is activated.

If bit 3 is set, logging for index drop commands is activated.

If bit 4 is set, logging for VDK commands is activated.

FullText-home: Directory that is the home directory for the full text engine that is being used by the Action Request System. If the server is not licensed for full text support, this entry is not used.

FullText-License-Timeout: The interval (in hours) after which full text licensed users who have performed no action to the AR System server are automatically disconnected. If this is a user holding a floating full text license token, that token is freed at this time.

Informix-DBServer-Name: The name of the INFORMIX DB Server when accessing a remote INFORMIX database. This field is present only if the underlying database is INFORMIX On-line.

Informix-SE-DBPath: Directory where the INFORMIX-SE database is stored for the ARSystem database. This is often the same as the Dbhome-directory setting for the database but it does not have to be. This field is present only if the underlying database is INFORMIX-SE.

Informix-TBConfig: Filename for the *tbconfig* file for the INFORMIX database. This field is present only if the underlying database is INFORMIX.

Ingres-Vnode-Name: The name of the INGRES Vnode when accessing a remote INGRES database. This field is present only if the underlying database is INGRES.

License-Timeout: The interval (in hours) after which users who have performed no action to the AR System server are automatically disconnected. If this is a user holding a floating write license token, that token is freed at this time.

Max-Entries-Per-Query: The maximum number of entries the server will return from a single query. The actual maximum is the lower of this number and the number configured by the user for the actual call.

Max-Fast-Daemons: Number of “fast” servers to run in the system. A “fast” server is one that performs its operations quickly without blocking on the database. No searches are performed through these servers. The server must be licensed to run multi-process.

Legal values for this field are 0 through 15. The default if nothing is specified is 0 (don’t run any “fast” servers).

Max-List-Daemons: Number of “list” servers to run in the system. A “list” server is one that performs a search in the database for its results. This may lead to a short or long block on response from the server.

Legal values for this field are 0 through 35. The default if nothing is specified is 0 (don’t run any “list” servers).

Oracle-SID: System ID for the ORACLE database. This field is present only if the underlying database is ORACLE.

Oracle-Two-Task: Two task environment setting for the ORACLE database. This field is present only if the underlying database is ORACLE.

Server-directory: Directory that serves as the database directory for the Action Request System. For the flat file database, it contains all the database definition and data files. For any of the SQL databases, it contains support files but not the actual database files.

Set-Process-Timeout: Timeout setting for a process run during a set fields action within a filter. To avoid hanging the server, the server will timeout after this many seconds if the process has not completed yet. The value can be between 1 and 20 seconds with a default of 5 seconds if nothing is set.

SQL-Log-File: The name of the file to be used if SQL tracing (see **Debug-Mode:**) is activated.

Submitter-Mode: Current mode for the server. The server can be put into two modes: Locked and Changeable. If Locked, the Submitter field cannot be changed after an entry is submitted and the Submitter can change values via permissions in the Submitter group without needing change permission. If Changeable, the Submitter field can be changed as needed after submit, but the Submitter must have a fixed or floating write license to change data.

Legal values for this field are 1 (locked) and 2 (changeable). The default if nothing is specified is 2 (changeable).

Suppress-warnings: A series of 0 or more numbers of warning or note messages to be suppressed. If a warning or note is encountered by the system, but the code for the message is on this list, the message will be suppressed.

Sybase-Server-Name: Server name of the SYBASE database. This field is present only if the underlying database is SYBASE.

Use-Password-File: Flag indicating whether the AR System server will look in the */etc/passwd* file during user validation if the user is not registered in the AR System. If true, logins are checked in /etc/password if not found in the AR System itself. If a match is found, the user is valid. The UNIX group ID is used as the ID for the group the user found belong to.

Legal values for this field are **T** and **F**. The default if neither is specified is T (allow password file lookup).

User-Log-File: The name of the file to be used for logging of active user registration to the AR System.

EXAMPLES

The following **/etc/ar.conf** file:

```
# Action Request System configuration file
Server-directory:/usr/ar/db
Dbhome-directory:/usr/SQL-DB
```

notes that the AR System database directory is at /usr/ar/db and the home directory of the SQL database system we are using is at /usr/SQL-DB.

SEE ALSO

arservdsd (1), arserverd (1), **arservftd** (1), **arservtcd** (1), **ARGetFullTextInfo** (3), **ARGetServerInfo** (3), **ARSetFullTextInfo** (3), **ARSetServerInfo** (3)

Glossary

access control

Security feature that lets you limit the access users have to specific fields within a schema and to specific functions within the system.

See also access control group, permissions.

access control group

Facility of the Action Request System used primarily to define user access to the contents of a schema field. Each group can have its own member list defining users who belong to that group. The AR System defines a number of *special* groups: Public, Administrator, Subadministrator, Customize, Submitter, and Assignee. You can define additional groups through the Group schema. Once you have defined a group, you can specify the type of access that the group will have to specific fields within a schema. *See also* access control, permissions.

access permissions

See permissions.

action request

AR. A collection of information that describes an event (transaction), such as a problem or a service request.

active link

A cause and effect relationship that you define on a per schema basis. Active links cause the Action Request System to perform specific operations in response to specific user actions. The AR System administrator can define active links that run macros, set fields to specified values, run independent

system processes, send an interactive message to the user, change field characteristics, or execute a DDE operation on a Windows User Tool. Active links run on the client machine.

administrator

Individual responsible for the management of the AR System, including setting up schemas, setting access rights for users, and designing the workflow process. To manage the AR System, you must be a member of the Administrator or Subadministrator group.

administrator default

Value that the administrator assigns to a field while designing the schema. When the user sets defaults, this value is used unless the user has assigned their own default. When a user submits an AR, the AR System automatically enters this value in the field unless the user has assigned their own default or has entered a different value.

administrator command

Menu extension defined by the AR System administrator using the Administrator Tool. Administrator commands allow users to invoke specific commands running on the server. (Administrator commands are not available from PC or Macintosh clients.) *See also* user command.

Administrator group

One of several special access control groups provided by the AR System. Members of this group have full and unlimited access to the AR System, including unlimited ability to create schemas, filters, escalations, active links, menus, and administrator commands. S*ee also* Subadministrator group.

Administrator Tool

The part of the AR System used exclusively by administrators (and to a lesser extent, by subadministrators) to set up the system for use by support staff and end users. This includes setting up schemas, setting access permissions (users and groups), and creating filters, escalations, active links, menus, and administrator commands.

administrator view

The layout of a schema that was designed by the AR System administrator. This is the view that users will see unless they customize their view.

admin server

The arserverd process that can handle *any* AR System operation. The admin server performs *all* admin restructuring operations, guaranteeing the serialization and integrity of data. There can be only *one* admin server process at any time.

API

Application program interface. A set of functions that provide application programmers with access to the full functionality of a product. The AR System API provides a complete interface to the AR System server.

AR

See action request.

arservtcd server

The controller process that handles requests from clients for information on which socket to use for communicating with other server processes.

AR System client

1.Subset of AR System software necessary to allow a user to access an AR System server on the network and run the AR System tools on the local workstation.

2.Hardware (workstation, terminal, Macintosh, or PC) running the AR System client software.

AR System server

1.Full set of AR System software, including the *arservtcd* and the fast, list, and escalation *arserverd* processes. When installed on a workstation on the network, the server software provides access to the full feature set of the AR System and can be accessed by workstations, Macintoshes, terminals and PCs on the network that are running the AR System client software.

2.Hardware (workstation) running the AR System server software.

assignee

The person who is assigned responsibility for resolving an action request.

Assignee group

One of several special access control groups provided by the AR System. This is an *implicit* group; users automatically belong to this group and, if they have a valid AR System license, are granted change access for ARs for which they have been assigned responsibility (their name is in the Assigned-to field).

character data type

Data type used for fields where you will be entering text data. The AR System administrator can specify a maximum length for the field or leave the length unlimited. The administrator can also specify a pattern to restrict the data that users can enter or attach a character menu to the field.

character menu

A type of menu that the AR system administrator can create and attach to any character-type data field. Character menus can be displayed as list boxes or pull-right menus.

client

See AR System client.

command line options

Parameters that you can combine with the commands to start the User, Administrator, Import, and Notification Tools that allow you to specify how the tools will run. For the User Tool, you can execute a macro or open to the Query or Submit window. For the Administrator Tool, you can attach to a specific server or open the tool with a specific category displayed.

core field

One of a set of basic fields that are common to all AR System schemas. Additional limits, such as fixed or maximum sizes, are placed on some core fields.

configuration

1.The process of setting up hardware and/or software so that it operates in a manner consistent with the needs of a location.

2.The physical setup of a device or devices.

3.The operating characteristics of software.

custom extension

See administrator command, user command.

Customize group

One of several special access control groups provided by the AR System. This group grants users the right to customize their schema layout and create custom commands in the User Tool.

database

A collection of information maintained in the form of individual entries. The AR System allows you to create and maintain a history of trouble reporting information.

data type

Property of a field that determines what type of information the field contains. The choices are character, date and time, diary, integer, real, and selection.

date/time data type

Fields with this data type are limited to calendar dates and time.

DDE

Dynamic Date Exchange. This is a standard inter-application communication feature used in Windows applications. For more information, see your Windows documentation.

default

Administrator or user defined setting or value that automatically applies to a field if users do not supply a different setting or value when submitting a new action request.

diary data type

Fields with this data type allow you to capture a history of the actions taken for an AR. Each multiple character entry is stamped with the time, date, and name of the user who entered the item.

dynamic menu

Menu that performs a query at the time a user selects the menu icon and uses the results of the query to build the list of menu items from which the user chooses.

end user

In the AR System, an end user is the person who is responsible for notifying support staff of problems and service requests by submitting ARs.

email

Electronic mail. The AR System allows you to set up an electronic mail handler so that users can submit ARs through email if they do not have access to a User Tool or if the AR server is inaccessible. (If you are running the client tools on a PC, your PC must be equipped with an SMTP gateway to allow email submissions.)

escalations

Facility that tests server transactions at specified times or regular intervals to see if certain conditions are met and responds to the conditions by taking a specific action or actions. The AR System administrator can define escalations to perform actions such as notify an individual, run a process, set specified fields, or make an entry to a log.

This facility is useful if, for example, you want to notify support staff when ARs are in the Assigned state too long.

escalation server

The arserverd process that, if enabled, handles all escalation operations.

export

Facility that lets you move schemas, filters, active links, menus, administrator commands, and mail templates to a file. Exporting is useful if you want to share schemas with another server or generate mail templates.

fast server

The arserverd process that, if enabled, handles the operations that generally run to completion quickly without blocking access to the server.

extension

See user command, administrator command.

filter

Facility that tests every server transaction to see if certain conditions are met and responds to the conditions by taking a specific action or actions. The AR System administrator can define filters that set fields to specified values, run independent system processes, send an interactive message to the user, notify the user when the state of an AR changes, or make an entry in an audit trail log. Filters run on the server.

fixed license

Write license that is permanently assigned to a user so that the user always has access to the AR System.

See also floating license, write license.

floating license

Write license that exists on a server and is allocated to any user who requests a license and who is defined in the User database as having a floating license type. If no floating license is available at the time of the user request, the user must wait until a license becomes available.

See also fixed license, write license.

FTS

See full text search.

FTS license

Fixed or floating license that allows a user to perform a full text search in any large text or diary field indexed for FTS.

full text search (FTS)

Facility that allows a user to quickly search for information in large text or diary fields. The fields must be indexed and FTS-enabled by the AR System administrator, and the user must have an FTS license.

group access

See group type.

Group schema

Schema that lets you add new groups and modify group permissions.

group type

The maximum permission type allowed for a group. May be None, View, or Change. (Note that permission may be set below the group's maximum at the field level.)

guest user

An unregistered user with a limited set of capabilities (submit ARs and possibly review those ARs). Unregistered users may not be allowed at your site.

hidden field

A field that exists but is not visible in a user's view of the schema.

import

Facility that lets you share schemas, filters, escalations, active links, menus, and administrator commands that were created on another server. First, you must export the definitions from the server on which they were created to an ASCII file, then you can import the file to your own server.

Import Tool

The part of the AR System that lets you transfer data from a data file to a server.

integer data type

Fields with this data type contain numeric values between -2147483648 and 2147483647. (The range for a particular field may be limited by the administrator.)

license

See Flashboards license, fixed license, floating license, FTS license, read license, write license.

list server

The arserverd process that, if enabled, handles the operations of the AR System that may take some time to complete: AR Export, ARGetListEntry (high-performance database searches), and ARGetEntryStatistics.

login window

Window that allows you to login to the AR System when you first start an AR System tool.

macro

A set of operations recorded for later execution. Macros are useful for automating frequently used or complex query operations.

mail template

Template that contains the fields that you need to fill in to submit an action request using electronic mail. Templates are generated by the administrator from existing schema using the export facility. (If you are running the client tools on a PC, your PC must be equipped with an SMTP gateway to allow email submissions.)

multiple schema views

Ability of an administrator to create different *views* of what users see when they bring up a schema, including hiding or re-arranging fields by changing a field's display properties.

multi-process server

A product that allows the AR System administrator to distribute operations among different servers and, consequently, improve the performance of the AR System.

For example, the arservtcd process routes the load among the Admin server, Fast server, List server, Escalation server and Private servers as appropriate, automatically starts the specified arserverd processes and restarts the arserverd processes if they terminate.

notification

An alert that tells you that an AR System event has occurred. The alert may be a system beep, flash, the display of a notice window, or the opening of the Notification Tool.

Notification Tool

The part of the AR System that alerts you when specific changes are made to ARs. Also referred to as the Notifier.

operator

One of a number of functions that let you define complex queries or build filter qualifications. The AR System operators are available through use of the query bar palette or the filter qualification palette or you can type them in directly.

permissions

Field property setting that allows you to control who can view and change individual fields of a schema. Permissions are defined for each access control group. View permission limits group members to reading the contents of a field. Change permission allows group members to read and write the contents of a field. *See also* access control group.

pick list

See selection list.

private server

An arserverd process that, if enabled, handles dedicated access to system operations for specific users.

property

An attribute that is defined. For example, the properties of a field include its data type, physical characteristics such as length, and whether it is required or optional.

Public group

One of several special access control groups provided by the AR System. Every user is automatically a member of this view only group.

pull-right menu

See character menu.

query

Process that lets you select a subset of ARs according to search criteria that you define and then perform one of several operations on the selected ARs. *See also* query operation.

query bar

Part of the Query window that lets you define complex query criteria. Includes a palette of operators that you can use in the query you build.

query list

A list that includes a one-line summary of each AR matching a query.

query operation

Action that you can perform on the entries that match the criteria defined in a query. The possible operations are: Query List, Display, Modify, Report, and Delete.

query statement

A complete definition of query criteria constructed in the query bar.

Query window

The User Tool window that lets you search the database for ARs that match specific criteria and display the results of the search. You also use the Query window to view or modify an existing AR. *See also* Submit window.

range

Defines the upper and lower limits of acceptable values. For example, if a field's range is -10 to 100, you will be able to enter any number from negative 10 to positive 100 inclusive.

read license

License that allows a user to query the AR System schemas and submit new ARs but does *not* allow the user to modify or save data on existing ARs. *See also* write license.

real data type

Fields with this data type contain a floating-point number. The range is set by the administrator.

report format

The layout that you specify when you generate a report from an AR System query. You can format a report in columns or as a list of records. You can also choose selected fields to print or print them all. To create a more sophisticated layout, you can export the report to a file and import the file into a desktop publishing application.

reserved field

One of a set of fields defined with specific interpretations. You can use these fields in any schema, if desired.

schema

The definition of the data fields in a database. Each schema represents a database on an AR System server. The AR System comes with several sample schemas and you can build as many additional schemas as needed.

scroll bar

Window element that appears when there is more information to view than will fit in the window. You use the mouse to slide the scroll bar and shift the view area. A scroll bar at the bottom of the window lets you move the viewing area left and right. A scroll bar on the right side of the window lets you move the viewing area up and down.

selection data type

Fields with this data type present a set of mutually exclusive choices from which the user is to choose. The selections are displayed as radio buttons or as items on a menu.

selection list

List that appears as a result of an active link that performs a query that returns more than one AR. The selection list lets the user pick the appropriate AR for the active link to continue processing with.

server

See AR System Server.

status field

Core field that lets you keep a record as an AR moves through the various stages of the process you are using to resolve ARs. The defined states should reflect the workflow process.

status history

Information that shows the progress that has been made on an AR. You can view status history from the Display or Modify window.

subadministrator

Individuals who have *limited* administrative access to the AR System. To be a subadministrator, you must be a member of the Subadministrator group and belong to a group with sub-admin access to a schema.

Subadministrator

Group
One of several special access control groups provided by the AR System. Members of this group have *limited* access to the AR System. You must be a member of this group to be able to administer any schemas that your group has sub-admin access to and to create and administer filters, active links, and escalations connected to schemas that your group has administrative access to.

Submit window

The User Tool window that lets you enter the appropriate information to create and submit a new AR. *See also* Query window.

submitter

The person who submits an action request. The submitter's name is entered in the Submitter field.

Submitter group

One of several special access control groups provided by the AR System. This is an *implicit* group; users submitting ARs automatically belong to this group and, if they have a valid AR System license or if the Submitter Mode is set to Locked, are granted change access for ARs that are submitted with their name in the Submitter field.

support staff

Person or group responsible for resolving action requests. They assign and are assigned ARs, log their progress in appropriate fields, and use information stored in previous ARs to help resolve problems.

toolbar

A floating window (Mac) or part of the Main window (Windows) that allows easy access to some of the more commonly performed functions in the Windows and Macintosh User Tool and Windows Administrator Tool.

user default

Value that a user who has customize permission can assign to a field. When the user sets defaults, the AR System loads this value into the field. When the user submits an AR, the AR System automatically loads this value into the field unless the user has entered a different value.

user command

Facility that allows you to invoke an operating system command (or application capable of being run from an operating system command line) from the Execute menu of the Query window. Each user can define their own set of user commands. (User commands are not available from PC or Macintosh clients.) *See also* administrator command.

User schema

Schema that lets you add users to the AR System and specify the type of access each user will have.

User Tool

The part of the AR System that lets users enter new ARs and track them through the troubleshooting process. Users can also query the database for ARs that match specified criteria, generate reports, and modify existing ARs with the User Tool.

user view

What the user sees when they bring up a schema. If users have permission to customize their views, they are able to change the physical layout and other properties of the schema and schema fields as they appear for that user.

variable

Data element that changes according to user input. In macros, you can include variable definitions that will cause the AR System to prompt the user for certain information when the macro executes.

version

The system release number. To display the version of the AR System that you are running, select *About* (Windows) or *On Version* (Motif) under the *Help* menu in the User Tool. If you have the Macintosh User Tool, select *About AR user Tool* under the Apple menu.

view layout

The location of fields in a user's view of a schema.

wild card

Character that you can enter to represent other characters in a search string. In query statements, for example, you can use wild card characters to match single characters, strings, or characters within a range or set.

write license

License that allows a user to modify and save data on existing ARs as field and schema permission settings allow. Write licenses may be either fixed (permanently attached to a single user) or floating (allocated to users as required).

See also fixed license, floating license, read license.

Index

D

E

F

G

I

K

L

M

N

O

P

Q

R

S

T

U

V

W