

# BMC® Remedy® Action Request System® 7.0

# C API Reference



May 2006  
Part No: 58480

Copyright 1991–2006 BMC Software, Inc. All rights reserved.

BMC, the BMC logo, all other BMC product or service names, BMC Software, the BMC Software logos, and all other BMC Software product or service names, are registered trademarks or trademarks of BMC Software, Inc. All other trademarks belong to their respective companies.

BMC Software, Inc., considers information included in this documentation to be proprietary and confidential. Your use of this information is subject to the terms and conditions of the applicable end user license agreement or nondisclosure agreement for the product and the proprietary and restricted rights notices included in this documentation.

For license information about the OpenSource files used in the licensed program, please read [OpenSourceLicenses.pdf](#). This file is in the \DOC folder of the distribution CD-ROM and in the documentation download portion of the product download page.

### **Restricted Rights Legend**

U.S. Government Restricted Rights to Computer Software. UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT LAWS OF THE UNITED STATES. Use, duplication, or disclosure of any data and computer software by the U.S. Government is subject to restrictions, as applicable, set forth in FAR Section 52.227-14, DFARS 252.227-7013, DFARS 252.227-7014, DFARS 252.227-7015, and DFARS 252.227-7025, as amended from time to time. Contractor/Manufacturer is BMC Software, Inc., 2101 CityWest Blvd., Houston, TX 77042-2827, USA. Any contract notices should be sent to this address.

### **Contacting Us**

If you need technical support for this product, contact Customer Support by email at [support@remedy.com](mailto:support@remedy.com). If you have comments or suggestions about this documentation, contact Information Development by email at [doc\\_feedback@bmc.com](mailto:doc_feedback@bmc.com).

This edition applies to version 7.0 of the licensed program.

# Contents

Preface . . . . .	15
Audience . . . . .	16
AR System documents . . . . .	17
Learn about the AR System Developer Community . . . . .	19
Why should you participate in the Developer Community? . . . . .	19
How do you access the Developer Community? . . . . .	19
Chapter 1 Introduction . . . . .	21
API overview . . . . .	22
C API . . . . .	22
Java API . . . . .	23
Plug-in API . . . . .	24
AR System C and Java API architecture . . . . .	25
AR System plug-in API architecture . . . . .	27
C API call components . . . . .	28
When to use API programming . . . . .	28
BMC Remedy User and API terminology . . . . .	29
Chapter 2 Getting started . . . . .	31
C API package contents . . . . .	32
Header files . . . . .	32
Library files . . . . .	33
Compiler information . . . . .	35

Link information . . . . .	36
Sample Source code . . . . .	36
<b>Working with makefiles . . . . .</b>	<b>36</b>
UNIX . . . . .	36
Windows . . . . .	37
 <b>Chapter 3 Data structures . . . . .</b>	<b>39</b>
<b>Data relationships . . . . .</b>	<b>40</b>
C data types. . . . .	41
Lists . . . . .	42
High-level object relationships . . . . .	43
Login and session information . . . . .	45
Status information. . . . .	48
Permission information . . . . .	51
Representing values . . . . .	53
Representing qualifications. . . . .	55
Dynamic qualifications . . . . .	57
Qualifications that use conditional operators . . . . .	57
Qualifications that use relational operators . . . . .	58
<b>Schemas . . . . .</b>	<b>61</b>
<b>Fields . . . . .</b>	<b>63</b>
Defining field limits . . . . .	63
Defining field display properties. . . . .	65
Mapping fields in schemas . . . . .	68
<b>Entries . . . . .</b>	<b>70</b>
Retrieving entry lists. . . . .	70
Manipulating individual entries . . . . .	75
Retrieving multiple entries . . . . .	76
<b>Filters, escalations, and active links . . . . .</b>	<b>77</b>
Server object properties . . . . .	77
Server object actions. . . . .	81
Set fields action. . . . .	86
Push fields action . . . . .	94
Automation action . . . . .	95
<b>Character menus . . . . .</b>	<b>100</b>

---

<b>Containers</b>	102
Retrieving container lists	102
Manipulating individual containers	103
<b>Importing and exporting</b>	109
<b>Freeing allocated memory</b>	112
<b>XML formats</b>	114
<b>Chapter 4 AR System C API calls</b>	117
<b>Related files</b>	121
<b>Types of functions</b>	121
Object manipulation functions	121
Alert functions	125
Other functions	125
<b>Function descriptions</b>	127
<b>ARBeginBulkEntryTransaction</b>	127
<b>ARCreatActiveLink</b>	128
<b>ARCreatAlertEvent</b>	132
<b>ARCreatCharMenu</b>	134
<b>ARCreatContainer</b>	136
<b>ARCreatEntry</b>	139
<b>ARCreatEscalation</b>	140
<b>ARCreatField</b>	143
<b>ARCreatFilter</b>	160
<b>ARCreatLicense</b>	163
<b>ARCreatSchema</b>	164
<b>ARCreatSupportFile</b>	168
<b>ARCreatVUI</b>	169
<b>ARDateToJulianDate</b>	175
<b>ARDecodeAlertMessage</b>	176
<b>ARDecodeARAssignStruct</b>	179
<b>ARDecodeARQualifierStruct</b>	180
<b>ARDecodeDiary</b>	181
<b>ARDecodeStatusHistory</b>	182
<b>ARDeleteActiveLink</b>	183
<b>ARDeleteCharMenu</b>	184
<b>ARDeleteContainer</b>	185

<b>ARDeleteEntry</b>	186
<b>ARDeleteEscalation</b>	187
<b>ARDeleteField</b>	189
<b>ARDeleteFilter</b>	190
<b>ARDeleteLicense</b>	191
<b>ARDeleteMultipleFields</b>	192
<b>ARDeleteSchema</b>	193
<b>ARDeleteSupportFile</b>	195
<b>ARDeleteVUI</b>	196
<b>ARDeregisterForAlerts</b>	197
<b>AREncodeARAssignStruct</b>	198
<b>AREncodeARQualifierStruct</b>	199
<b>AREncodeDiary</b>	200
<b>AREncodeStatusHistory</b>	201
<b>AREndBulkEntryTransaction</b>	202
<b>ARExecuteProcess</b>	203
<b>ARExpandCharMenu</b>	204
<b>ARExport</b>	206
<b>ARExportLicense</b>	210
<b>ARGetActiveLink</b>	211
<b>ARGetAlertCount</b>	214
<b>ARGetApplicationState</b>	215
<b>ARGetCharMenu</b>	216
<b>ARGetClientCharSet</b>	218
<b>ARGetContainer</b>	219
<b>ARGetCurrencyRatio</b>	222
<b>ARGetEntry</b>	223
<b>ARGetEntryBLOB</b>	225
<b>ARGetEntryBlock</b>	227
<b>ARGetEntryStatistics</b>	228
<b>ARGetEscalation</b>	230
<b>ARGetField</b>	233
<b>ARGetFilter</b>	237
<b>ARGetListActiveLink</b>	240
<b>ARGetListAlertUser</b>	241
<b>ARGetListApplicationState</b>	242

<b>ARGetListCharMenu</b>	243
<b>ARGetListContainer</b>	244
<b>ARGetListEntry</b>	246
<b>ARGetListEntryBlocks</b>	249
<b>ARGetListEntryWithFields</b>	252
<b>ARGetListEscalation</b>	254
<b>ARGetListExtSchemaCandidates</b>	256
<b>ARGetListField</b>	257
<b>ARGetListFilter</b>	258
<b>ARGetListGroup</b>	260
<b>ARGetListLicense</b>	261
<b>ARGetListRole</b>	262
<b>ARGetListSchema</b>	263
<b>ARGetListSchemaWithAlias</b>	266
<b>ARGetListServer</b>	269
<b>ARGetListSQL</b>	270
<b>ARGetListSupportFile</b>	271
<b>ARGetListUser</b>	273
<b>ARGetListVUI</b>	274
<b>ARGetLocalizedValue</b>	275
<b>ARGetMultipleActiveLinks</b>	276
<b>ARGetMultipleCharMenus</b>	281
<b>ARGetMultipleContainers</b>	284
<b>ARGetMultipleCurrencyRatioSets</b>	288
<b>ARGetMultipleEntryPoints</b>	289
<b>ARGetMultipleEntries</b>	293
<b>ARGetMultipleEscalations</b>	295
<b>ARGetMultipleExtFieldCandidates</b>	298
<b>ARGetMultipleFields</b>	299
<b>ARGetMultipleFilters</b>	304
<b>ARGetMultipleLocalizedValues</b>	308
<b>ARGetMultipleSchemas</b>	309
<b>ARGetMultipleVUIs</b>	314
<b>ARGetSchema</b>	316
<b>ARGetServerCharSet</b>	320
<b>ARGetServerInfo</b>	322

ARGetSessionConfiguration . . . . .	349
ARGetServerStatistics . . . . .	351
ARGetSupportFile . . . . .	358
ARGetTextForErrorMessage . . . . .	360
ARGetVUI . . . . .	360
ARIImport . . . . .	363
ARIImportLicense . . . . .	366
ARIInitialization . . . . .	367
ARJulianDateToDate . . . . .	368
ARLoadARQualifierStruct . . . . .	369
ARMergeEntry . . . . .	370
ARRegisterForAlerts . . . . .	372
ARSetActiveLink . . . . .	373
ARSetApplicationState . . . . .	378
ARSetCharMenu . . . . .	379
ARSetContainer . . . . .	381
ARSetEntry . . . . .	384
ARSetEscalation . . . . .	386
ARSetField . . . . .	389
ARSetFilter . . . . .	393
ARSetImpersonatedUser . . . . .	397
ARSetLogging . . . . .	397
ARSetSchema . . . . .	399
ARSetServerInfo . . . . .	403
ARSetServerPort . . . . .	405
ARSetSessionConfiguration . . . . .	406
ARSetSupportFile . . . . .	408
ARSetVUI . . . . .	410
ARSignal . . . . .	412
ARTermination . . . . .	413
ARValidateFormCache . . . . .	413
ARValidateLicense . . . . .	416
ARValidateMultipleLicenses . . . . .	417
ARVerifyUser . . . . .	418
FreeAR . . . . .	419

---

<b>Chapter 5</b>	<b>Creating and executing AR System C API programs . . . . .</b>	<b>433</b>
	Program structure . . . . .	434
	Performing common tasks . . . . .	436
	Specifying fields or keywords . . . . .	440
	Error checking . . . . .	441
	Executing C API programs in workflow . . . . .	443
	Program design tips . . . . .	445
	Multithreaded C API clients . . . . .	445
	Impersonating a user . . . . .	446
<b>Chapter 6</b>	<b>AR System automation . . . . .</b>	<b>447</b>
	<b>Overview . . . . .</b>	<b>449</b>
	<b>Understanding the automation type library. . . . .</b>	<b>450</b>
	Accessing type information. . . . .	450
	Invoking member methods. . . . .	451
	<b>Building an application to control BMC Remedy User . . . . .</b>	<b>451</b>
	Including the type library file . . . . .	451
	Handling errors . . . . .	452
	Studying a sample program. . . . .	452
	<b>The BMC Remedy User automation object model . . . . .</b>	<b>455</b>
	<b>The application object: ICOMAppObj . . . . .</b>	<b>456</b>
	<b>Login . . . . .</b>	<b>457</b>
	<b>Logout . . . . .</b>	<b>458</b>
	<b>GetServerList . . . . .</b>	<b>458</b>
	<b>GetFormList . . . . .</b>	<b>459</b>
	<b>OpenForm . . . . .</b>	<b>459</b>
	<b>LoadForm . . . . .</b>	<b>460</b>
	<b>GetActiveForm . . . . .</b>	<b>461</b>
	<b>HasDefaultSession. . . . .</b>	<b>462</b>
	<b>OpenGuide . . . . .</b>	<b>462</b>
	<b>RunMacro . . . . .</b>	<b>463</b>
	<b>The form object: ICOMFormWnd. . . . .</b>	<b>463</b>
	<b>Submit. . . . .</b>	<b>465</b>
	<b>Modify. . . . .</b>	<b>465</b>
	<b>Close. . . . .</b>	<b>465</b>

<b>MakeVisible</b> . . . . .	466
<b>GetField</b> . . . . .	466
<b>GetFieldById</b> . . . . .	466
<b>GiveFieldFocus</b> . . . . .	467
<b>GiveFieldFocusById</b> . . . . .	467
<b>Query</b> . . . . .	467
<b>GetServerName</b> . . . . .	468
<b>GetFormName</b> . . . . .	468
The field object: <b>ICOMField</b> . . . . .	469
<b>MakeVisible</b> . . . . .	469
<b>MakeReadWrite</b> . . . . .	470
<b>Disable</b> . . . . .	470
The query result object: <b>ICOMQueryResult</b> . . . . .	470
The query result set object: <b>ICOMQueryResultSet</b> . . . . .	471
<b>Item</b> . . . . .	471
 <b>Chapter 7</b>	
<b>Debugging and maintenance</b> . . . . .	473
<b>Logging AR System activity</b> . . . . .	474
<b>Using the driver program</b> . . . . .	475
Using print.C routines . . . . .	476
Using the driver program from the command line . . . . .	478
Creating and using driver scripts . . . . .	480
 <b>Chapter 8</b>	
<b>AR System plug-ins</b> . . . . .	481
<b>Overview</b> . . . . .	483
<b>LDAP plug-ins</b> . . . . .	483
<b>Plug-ins that you create</b> . . . . .	483
Installing plug-in files and components . . . . .	485
Setting up the environment . . . . .	485
Creating plug-ins . . . . .	485
Plug-in conventions . . . . .	488
Running the arplugin server . . . . .	489
Accessing the plug-ins . . . . .	490
AREA plug-in . . . . .	490
ARDBC plug-in . . . . .	492
ARF plug-in . . . . .	494

---

<b>Plug-in logging facility</b> . . . . .	496
Log function . . . . .	496
Logging levels . . . . .	496
<b>Plug-in API calls</b> . . . . .	497
<b>ARPluginCreateInstance</b> . . . . .	498
<b>ARPluginDeleteInstance</b> . . . . .	498
<b>ARPluginEvent</b> . . . . .	499
<b>ARPluginIdentify</b> . . . . .	499
<b>ARPluginInitialization</b> . . . . .	500
<b>ARPluginSetProperties</b> . . . . .	501
<b>ARPluginTermination</b> . . . . .	502
<b>AREA API</b> . . . . .	502
AREA API data structure . . . . .	502
<b>AREA API calls</b> . . . . .	505
<b>AREAFreeCallback</b> . . . . .	505
<b>AREANeedToSyncCallback</b> . . . . .	506
<b>AREAVerifyLoginCallback</b> . . . . .	506
<b>ARDBC API</b> . . . . .	508
ARDBC API data structure . . . . .	508
<b>ARDBC API calls</b> . . . . .	509
<b>ARDBCCommitTransaction</b> . . . . .	509
<b>ARDBCCreateEntry</b> . . . . .	510
<b>ARDBCDeleteEntry</b> . . . . .	511
<b>ARDBCGetEntry</b> . . . . .	513
<b>ARDBCGetEntryBLOB</b> . . . . .	515
<b>ARDBCGetEntryStatistics</b> . . . . .	516
<b>ARDBCGetListEntryWithFields</b> . . . . .	519
<b>ARDBCGetListSchemas</b> . . . . .	521
<b>ARDBCGetMultipleFields</b> . . . . .	522
<b>ARDBCRollbackTransaction</b> . . . . .	523
<b>ARDBCSetEntry</b> . . . . .	524
<b>ARF API calls</b> . . . . .	526
ARF API data structure . . . . .	526
<b>ARFilterApiCall</b> . . . . .	526

<b>Chapter 9</b>	<b>XML support . . . . .</b>	<b>529</b>
	<b>Transforming XML and AR System objects . . . . .</b>	<b>530</b>
	Schema definition files . . . . .	531
	<b>Object API calls . . . . .</b>	<b>531</b>
	Using the object XML calls . . . . .	533
	<b>Function descriptions . . . . .</b>	<b>535</b>
	<b>ARGetActiveLinkFromXML . . . . .</b>	<b>536</b>
	<b>ARGetContainerFromXML . . . . .</b>	<b>539</b>
	<b>ARGetDSOMappingFromXML . . . . .</b>	<b>542</b>
	<b>ARGetDSOPoolFromXML . . . . .</b>	<b>545</b>
	<b>ARGetEscalationFromXML . . . . .</b>	<b>547</b>
	<b>ARGetFieldFromXML . . . . .</b>	<b>550</b>
	<b>ARGetFilterFromXML . . . . .</b>	<b>553</b>
	<b>ARGetListXMLObjects . . . . .</b>	<b>556</b>
	<b>ARGetMenuFromXML . . . . .</b>	<b>557</b>
	<b>ARGetSchemaFromXML . . . . .</b>	<b>559</b>
	<b>ARGetVUIFromXML . . . . .</b>	<b>563</b>
	<b>ARParseXMLDocument . . . . .</b>	<b>564</b>
	<b>ARSetActiveLinkToXML . . . . .</b>	<b>565</b>
	<b>ARSetContainerToXML . . . . .</b>	<b>569</b>
	<b>ARSetDSOMappingToXML . . . . .</b>	<b>572</b>
	<b>ARSetDSOPoolToXML . . . . .</b>	<b>575</b>
	<b>ARSetEscalationToXML . . . . .</b>	<b>577</b>
	<b>ARSetFieldToXML . . . . .</b>	<b>580</b>
	<b>ARSetFilterToXML . . . . .</b>	<b>583</b>
	<b>ARSetMenuToXML . . . . .</b>	<b>586</b>
	<b>ARSetSchemaToXML . . . . .</b>	<b>588</b>
	<b>ARSetVUIToXML . . . . .</b>	<b>592</b>
	<b>ARSetXMLDocFooterToXML . . . . .</b>	<b>593</b>
	<b>ARSetXMLDocHeaderToXML . . . . .</b>	<b>594</b>
<b>Appendix A</b>	<b>Migrating to the Action Request System 7.0 API . . . . .</b>	<b>597</b>
	<b>Existing program changes . . . . .</b>	<b>598</b>
	New and changed C data types . . . . .	598
	Changed parameters . . . . .	599

Data structure changes and additions . . . . .	599
New server information options. . . . .	600
<b>Index. . . . .</b>	<b>603</b>



# Preface

---

**Important:** The compatibility information listed in the product documentation is subject to change. See the compatibility matrix at <http://supportweb.remedy.com> for the latest, most complete information about what is officially supported.

Carefully read the system requirements for your particular operating system, especially the necessary patch requirements.

---

This chapter describes the audience level required to use this guide and lists related documents and BMC® Remedy® Action Request System® (AR System®) API-related classes.

# Audience

This guide is intended for software developers who want to use the AR System APIs to further customize the AR System.

You should know how to write API programs and be familiar with AR System, including the overall architecture and the information in the following guides:

- *Concepts*
- *Installing*
- *Getting Started*
- *Form and Application Objects*
- *Workflow Objects*
- *Configuring*
- *Installing and Administering BMC Remedy Mid Tier*
- *Integrating with Plug-ins and Third-Party Products*
- *Error Messages*
- *Database Reference*
- *Optimizing and Troubleshooting*

In addition, you should be knowledgeable about the operating system for your environment and have some experience with client/server applications and plug-ins. This guide assumes that you are proficient in C programming and are familiar with arrays, pointers, structure types, and allocated memory.

To use the XML import and export capabilities, you should be knowledgeable about XML.

# AR System documents

The following table lists documentation available for AR System products.

Unless otherwise noted, online documentation in Adobe Acrobat (PDF) format is available on AR System product installation CDs, on the Customer Support web site ([supportweb.remedy.com](http://supportweb.remedy.com)), or both.

You can access product Help through each product's Help menu or by clicking on Help links.

Title	Description	Audience
<i>Concepts</i>	Overview of AR System architecture and features with in-depth examples; includes information about other AR System products as well as a comprehensive glossary for the entire AR System documentation set.	Everyone
<i>Installing</i>	Procedures for installing AR System.	Administrators
<i>Getting Started</i>	Introduces topics that are usually only learned when first starting to use the system, including logging in, searching for objects, and so on.	Everyone
<i>Form and Application Objects</i>	Describes components necessary to build applications in AR System, including applications, fields, forms, and views.	Developers
<i>Workflow Objects</i>	Contains all of the workflow information.	Developers
<i>Configuring</i>	Contains information about configuring AR System servers and clients, localizing, importing and exporting data, and archiving data.	Administrators
<i>Installing and Administering BMC Remedy Mid Tier</i>	Contains information about the mid tier, including mid tier installation and configuration, and web server configuration.	Administrators
<i>Integrating with Plug-ins and Third-Party Products</i>	Discusses integrating AR System with external systems using plug-ins and other products, including LDAP, OLE, and ARDBC.	Administrators /Developers
<i>Optimizing and Troubleshooting</i>	Server administration topics and technical essays related to monitoring and maintaining AR System for the purpose of optimizing performance and troubleshooting problems.	Administrators

Title	Description	Audience
<i>Database Reference</i>	Database administration topics and rules related to how AR System interacts with specific databases; includes an overview of the data dictionary tables.	Administrators
<i>Administering BMC Remedy DSO</i>	Server administration and procedures for implementing a distributed AR System server environment with the BMC Remedy Distributed Server Option (DSO).	Administrators
<i>Administering BMC Remedy Flashboards</i>	Flashboards administration and procedures for creating and modifying flashboards and flashboards components to display and monitor AR System information.	Administrators /Programmers
<i>C API Reference</i>	Information about AR System data structures, C API function calls, and OLE support.	Administrators /Programmers
<i>C API Quick Reference</i>	Quick reference to C API function calls.	Administrators /Programmers
<i>Java API</i> *	Information about Java classes, methods, and variables that integrate with AR System.	Administrators /Programmers
<i>Administering BMC Remedy Email Engine</i>	Procedures for installing, configuring, and using the BMC Remedy Email Engine.	Administrators
<i>Error Messages</i>	List and expanded descriptions of AR System error messages.	Administrators /Programmers
<i>Master Index</i>	Combined index of all books.	Everyone
<i>Release Notes</i>	Information about new features list, compatibility lists, international issues, and open and fixed issues.	Everyone
<i>BMC Remedy User Help</i>	Procedures for using BMC Remedy User.	Everyone
<i>BMC Remedy Import Help</i>	Procedures for using BMC Remedy Import.	Administrators
<i>BMC Remedy Administrator Help</i>	Procedures for creating and modifying an AR System application for tracking data and processes.	Administrators
<i>BMC Remedy Alert Help</i>	Procedures for using BMC Remedy Alert.	Everyone
<i>BMC Remedy Mid Tier Configuration Tool Help</i>	Procedures for configuring the BMC Remedy Mid Tier.	Administrators

\* A JAR file containing the Java API documentation is installed with the AR System server.

Typically, it is stored in `C:\Program Files\AR System\Arserver\Api\doc\ardoc70.jar` on Windows and `/usr/ar/<server_name>/api/doc/ardoc70.jar` on UNIX.

# Learn about the AR System Developer Community

If you are interested in learning more about AR System, looking for an opportunity to collaborate with fellow AR System developers, and searching for additional resources that can benefit your AR System solution, then this online global community sponsored by BMC Remedy is for you.

In the Developer Community, you will find collaboration tools, product information, resource links, user group information, and be able to provide BMC Remedy with feedback.

The Developer Community offers the following tools and information:

- Community message board
- Community Downloads
- AR System Tips & Tricks
- Community recommended resources
- Product information
- User Experience Design tips

## Why should you participate in the Developer Community?

You can benefit from participating in the Developer Community for the following reasons:

- The community is a direct result of AR System developer feedback.
- BMC Remedy provides unsupported applications and utilities by way of Community Downloads, an AR System application.
- BMC Remedy posts the latest AR System product information in the Developer Community to keep you up to date.
- It is an opportunity to directly impact product direction through online and email surveys.
- It's free!

## How do you access the Developer Community?

Go to [supportweb.remedy.com](http://supportweb.remedy.com), and click the Developer Community link.



Chapter

# 1

# Introduction

This chapter provides an overview of the AR System API and explains when to use API programming.

The following topics are provided:

- API overview (page 22)
- AR System C and Java API architecture (page 25)
- AR System plug-in API architecture (page 27)
- C API call components (page 28)
- When to use API programming (page 28)

# API overview

The API suite is composed of a C application program interface (API), a Java API, a plug-in API, and three APIs that use plug-ins.

This guide describes the C APIs and plug-in APIs. This introduction briefly discusses all APIs.

For more information about the Java API or the Java file names and locations, see the Java API documentation HTML files in the `ardoc70.jar` file in the `\Arserver\Api\doc` folder (Windows) or the `/ARServer/api/doc` folder (UNIX). To access the Java API documentation, unzip the `ardoc70.jar` file, then open the `index.html` file.

## C API

The AR System clients use the C APIs to manipulate data. For example, the clients uses C APIs to create, retrieve, update, and delete entries, forms, and menus, and to control workflow. You can use the C API to extend AR System functionality.

The AR System API contains data structures that store both simple and complex information. Structures that store simple information, such as the type of value or the product of an arithmetic operation, serve as the building blocks for complex structures.

The C APIs consist of sets of library routines that you can compile to run on Windows or UNIX platforms.

You can run API programs from a command line or from the AR System; in the `Set Field $PROCESS$` action from an active link, filter, or escalation; or with the `Run Process` action in an active link, filter, or escalation.

## Java API

The AR System Java API is a collection of Java classes that provide the full AR System C API functionality in a Java development environment.

The Java API:

- Provides an object model of AR System server entities (also called server objects), definitions, and data.
- Accesses the AR System server indirectly, through the `ARServerProxy` class and methods on the object model classes.
- Is easier to program than the C API, in part, because you do not have to establish and manage connections, or manipulate complex setup API parameters.

You will find it easier to use the Java API if you are already familiar with the C API.

### Choosing to use the Java API

Use the AR System Java API to perform the following tasks:

- Write server-side web applications that you access through the Java server page (JSP) or Java servlets web tier layer.
- Implement AR System clients in Java.

The Java classes and C API methods use similar structures to encapsulate information and functionality. However, not every Java class results in a C API call to the server.

### Comparison of the Java and C APIs

There are a number of differences between the AR System Java and C APIs. For example:

- The Java virtual machine (JVM) uses garbage collecting functions to automatically deallocate objects that are created with the Java API. The C API includes a set of memory deallocating functions that you can use to deallocate memory.
- In the C API, the control parameter provides server access information with every call. In the Java API, this information is encapsulated in an `ARServerUser` object.
- The C API and Java API have different naming conventions.

## Java API requirements

To build and run a Java API program on either Windows or UNIX, you need the following environment components:

- J2SE Software Development Kit (SDK), version 1.4.2 or higher
- Windows dynamic link library (DLL) files UNIX library files:

Platform	Files
Windows	<ul style="list-style-type: none"><li>■ arrpc70. dl l</li><li>■ arutil 70. dl l</li><li>■ arapi 70. dl l</li><li>■ arapi 70. j ar</li><li>■ arjni 70. dl l</li><li>■ aruti l 70. j ar</li><li>■ arjni 70. dl l</li></ul>
UNIX	<ul style="list-style-type: none"><li>■ arapi 70. j ar</li><li>■ l i barjni 70. so (<b>Solaris, AIX, Linux</b>)</li><li>■ l i barjni 70. sl (<b>HP-UX</b>)</li></ul>

## Plug-in API

AR System offers built-in Lightweight Directory Access Protocol (LDAP) plug-ins. For information about configuring the LDAP plug-ins, see the *Integrating with Plug-ins and Third-Party Products* guide. For information about how to run the plug-ins, see the *Form and Application Objects* guide. For other plug-in information, see Chapter 8, “AR System plug-ins.”

AR System offers a plug-in service (arpl ugi n) that extends its functionality to external data (data that is not contained in the AR System database).

The plug-in service supports three types of plug-ins with corresponding application program interfaces (APIs):

BMC® Remedy® Action Request System® External Authentication (AREA) plug-in—Accesses network directory services or other authentication services to verify user login name and password. When you use the AREA plug-in, you do not have to maintain duplicate user authentication data in the AR System directories because the AR System server can access user identification information and user passwords from external sources.

Action Request Database Connectivity (ARDBC) plug-in—Accesses external sources of data and enables you to manipulate the data. You can integrate ARDBC with external data sources through their own APIs. The ARDBC plug-in, which you access through a vendor form, enables you to perform the following tasks:

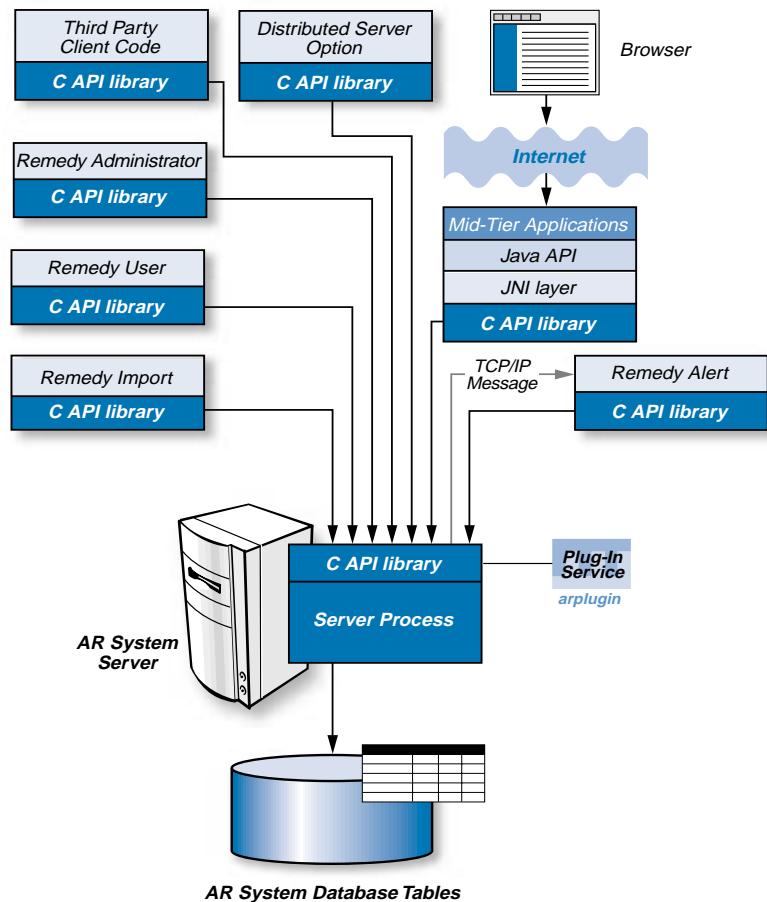
- Search external data sources.
- Create, delete, modify, and set entries.
- Populate search-style character menus.
- Implement workflow.

AR System filter (ARF) plug-in—Offers an alternate method to send information requests to and from external servers. In previous versions of the AR System, run processes performed external information requests. ARF uses fewer system resources than run processes use and enables the AR System server to return to its workflow faster. ARF also applies to escalations.

## AR System C and Java API architecture

As the following figure shows, AR System client code links to the C API libraries provided. The C APIs create an interface layer between the AR System server and AR System clients. AR System clients perform all database operations through the API interface. AR System Java calls pass through a JNI layer and the C API to the AR System server.

Figure 1-1: C and Java API architecture

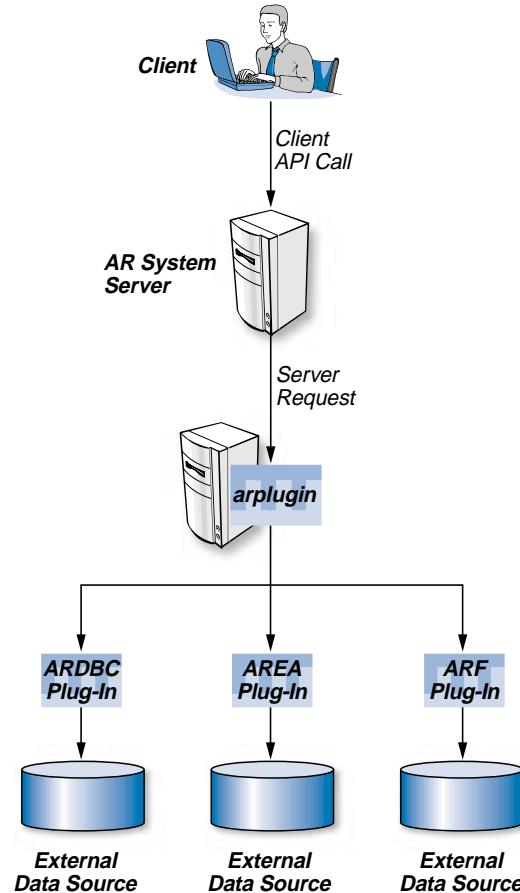


Note: The arrows in the preceding figure identify the directions in which each program or process can initiate API function calls. Data can flow in any direction.

# AR System plug-in API architecture

As the following figure shows, AR System clients perform external data source operations on external forms through the AR System server, plug-in service, and plug-in related APIs. The plug-in service extends the AR System server to integrate with external data sources. The AR System server connects to the plug-in service, which activates the plug-ins.

Figure 1-2: Plug-in architecture



**Note:** The arrows in this figure identify the directions in which each program or process can initiate API function calls. Data can flow in any direction.

## C API call components

The APIs consist of a set of function calls, most of which perform a specific database or data source operation. For example, you can create an entry or retrieve information about a particular AR System object.

Chapter 4, “AR System C API calls,” describes the purpose and use of each AR System C API function. Chapter 8, “AR System plug-ins,” describes the purpose and use of the AR System plug-in, AREA, ARDBC, and ARF API functions. For more information about the Java API, see the Java documentation HTML files in the `ardoc70.jar` file.

In addition, almost all of the AR System C API functions accept one or more structure-type parameters. Chapter 3, “Data structures,” explains some of the most common structures and the operations they apply to. If you understand the functions and structures that comprise the API, you can interact with the AR System server to customize and extend your applications.

The `dri ver` directory contains source code for the `dri ver` program. This program provides a command line interface for calling AR System C API functions. The `dri ver` program also includes print routines for every data structure in the API, making it a useful debugging tool. See “Using the driver program” on page 475 for additional information about this topic.

The `JavaDri ver` directory contains source code for a similar program that uses the AR System Java API.

## When to use API programming

The primary reason to write your own API program is to satisfy a requirement that you cannot accomplish with the AR System clients.

In addition, API programming gives you flexibility to customize your application. However, API solutions are more complex to design, implement, and maintain. If you write an API program, you must manage all cross-platform issues and issues related to AR System upgrades. See “Migrating to the Action Request System 7.0 API” on page 597 for information about changes to the API for version 7.0.

You might want to write your own an API program if:

- You need to access multiple AR System components at the same time or integrate with programs or data outside the AR System.
- You need to perform complex operations that involve multiple forms.
- You need a two-way interface (or gateway) between AR System and another application.
- The values you want to specify for \$PROCESS\$ or the Run Process action (see Chapter 5, “Creating and executing AR System C API programs”) exceed the size limitation of the command line.

On some systems, for example, the command-line area for \$PROCESS\$ or Run Process parameters is limited to 256 bytes (expandable to 4 KB). If you use an API program instead, you can pass a parameter to an API program and manipulate larger-sized data within the program.

- You need to create reports or log files in a format that is not compatible with standard report writing tools.
- You want to create, modify, and delete objects.

## BMC Remedy User and API terminology

Some terminology has changed from version to version of AR System and the BMC Remedy User client, but the API terms have stayed the same so that code does not need to be rewritten. The following table compares BMC Remedy User terms with API terms.

BMC Remedy User term	C API term
request	entry
form	schema
search	getlist
create/submit	create
display	get
modify	set



Chapter

# 2 Getting started

This chapter describes the preliminary steps you must follow before you create and use C API function calls.

The following topics are provided:

- C API package contents (page 32)
- Working with makefiles (page 36)

# C API package contents

You can either install the API files when you first install the AR System server or install them later.

---

**Note:** If you have already installed the AR System server and want to add the API package, make sure that you select the option to *share* the database rather than overwrite it.

---

The API package includes header files, library files, and source code for the driver sample programs.

## Header files

The `include` directory contains the following API header files:

File Name	Contents
<code>ar.h</code>	Data type and structure definitions, size limits, and constant definitions.
<code>ardbc.h</code>	ARDBC definitions.
<code>area.h</code>	AREA definitions.
<code>arerrno.h</code>	Error code definitions.
<code>arextern.h</code>	External declarations for the API functions, specified with and without prototypes for use with standard C, ANSI C, or C++ compilers.
<code>arfileterapi.h</code>	ARF definitions.
<code>arfree.h</code>	External declarations for the FreeAR functions. These functions recursively free all allocated memory associated with a particular data structure. Like <code>arextern.h</code> , declarations are specified with and without prototypes.
<code>arplugind.h</code>	AR System plug-in definitions.
<code>arstruct.h</code>	Core and reserved field ID definitions, database separator characters, and labels for exporting structure definitions.

## Library files

In addition to the requirements in the following table, you must:

- Have `arcatalog_eng.dll` in your path at runtime.
- Provide the full path to the AR System libraries.
- For Windows:

Have `arrpc70.dll`, `arutil70.dll`, and `arapi70.dll` in the path.

### *For XML API*

If you use XML API calls, also have the following files in the path:  
`arxmlutil70.dll`, `icudata.dll`, `icuuc.dll`, and `xerces-c_2_6.dll`.

- For UNIX:

Have `libbar.a` (static library) and `libbar.so` (shared object library).

Library file names depend on environment settings. The `lib` directory contains the following API library files:

Platform	Files
Windows	<ul style="list-style-type: none"> <li>■ <code>arapi70.dll</code></li> <li>■ <code>arcatalog_eng.dll</code></li> <li>■ <code>arrpc70.dll</code></li> <li>■ <code>arsrv70.dll</code></li> <li>■ <code>arutil70.dll</code></li> <li>■ <code>arxmlutil70.dll</code></li> <li>■ <code>icudt32.dll</code></li> <li>■ <code>icuin32.dll</code></li> <li>■ <code>icuuc32.dll</code></li> <li>■ <code>rcmn70.dll</code></li> <li>■ <code>Xalan-C_1_9.dll</code></li> <li>■ <code>XalanMessages_1_9.dll</code></li> <li>■ <code>xerces-c_2_6.dll</code></li> <li>■ <code>xerces-depdom_2_6.dll</code></li> </ul>

Platform	Files
Solaris	<ul style="list-style-type: none"> <li>■ I i bi cudata. so. 32. 0</li> <li>■ I i bi cui 18n. so. 32. 0</li> <li>■ <b>libicuio.so.32.0</b></li> <li>■ I i bi cuuc. so. 32. 0</li> <li>■ I i bxal an-c. so. 19. 0</li> <li>■ I i bxal anMsg. so. 19. 0</li> <li>■ I i bxerces-c. so. 26. 0</li> <li>■ I i bxerces-depdom. so. 26. 0</li> </ul>
AIX	<ul style="list-style-type: none"> <li>■ I i bi cudata32. 0. a</li> <li>■ I i bi cui 18n32. 0. a</li> <li>■ <b>linicuio32.0.a</b></li> <li>■ I i bi cuuc32. 0. a</li> <li>■ I i bxal an-c19. 0. a</li> <li>■ I i bxal anMsg19. 0. a</li> <li>■ I i bxerces-c26. 0. a</li> <li>■ I i bxerces-depdom26. 0. a</li> </ul>
HP-UX	<ul style="list-style-type: none"> <li>■ I i bi cudata. sl . 32. 0</li> <li>■ I i bi cui 18n. sl . 32. 0</li> <li>■ I i bi cuuc. sl</li> <li>■ I i bi cuuc. sl . 32. 0</li> <li>■ I i bxal an-c. sl . 19. 0</li> <li>■ <b>libxalanMsg.sl.19.0</b></li> <li>■ I i bxerces-c. sl . 26. 0</li> <li>■ I i bxerces-depdom. sl . 26. 0</li> </ul>
Linux	<ul style="list-style-type: none"> <li>■ I i bi cudata. so. 32. 0</li> <li>■ I i bi cui 18n. so. 32. 0</li> <li>■ <b>libicuio.so.32.0</b></li> <li>■ I i bi cuuc. so. 32. 0</li> <li>■ I i bxal an-c. so. 19. 0</li> <li>■ I i bxal anMsg. so. 19. 0</li> <li>■ I i bxerces-c. so. 26. 0</li> <li>■ I i bxerces-depdom. so. 26. 0</li> </ul>

*For Solaris and Linux:* To load dynamic libraries, you need to include the `-l dl` link flag in the link command. See the `dri ver` sample makefile.

## Compiler information

Before you write C API programs, you must verify system requirements and install the API package, which includes the following components:

- Microsoft Visual Studio .NET 2003 or later, or built with Microsoft Visual C++ version 7.0 (for Windows)
- Code generation: Multi threaded DLL
- Structure member alignment: 8 bytes (default)
- Set code generation to Multi threaded DLL, not Debug Multi threaded DLL. Where other included libraries cause conflicts, add /nodefaultlib: "MSVCRTD" to the project options to avoid using the Debug C runtime library. If your program references this library at runtime, memory management errors will occur when memory pointers are referenced by both the Debug and Release C runtime libraries.
- ANSI standard C or C++ compiler (for UNIX)
- For compilers that require strict ANSI, defining AR\_STRICT\_ANSI will resolve compile errors such as the following item:

Line xxx from ar.h "Alternative token 'not' may not be used as an identifier" struct ARQualifierStruct \*not;

To resolve the error, modify the code by adding define of AR\_STRICT\_ANSI as follows:

```
typedef struct ARQualifierStruct {
    unsigned int operation;
    union {
        struct {
            struct ARQualifierStruct *operandLeft;
            struct ARQualifierStruct *operandRight;
        } andor;
    #ifndef AR_STRICT_ANSI
        struct ARQualifierStruct *not;
    #else
        struct ARQualifierStruct *notQual;
    #endif
        ARRelOpStruct *relOp;
        ARInternalId fileId;
    } u;
} ARQualifierStruct;
```

With AR\_STRICT\_ANSI defined, the C preprocessor should pick up 'notQual' instead.

## Link information

The following table lists the API libraries required to link to.

Platform	Library
Windows	arapi 70. lib
Solaris, Linux	libbar.a (archive) libbar.so (shared library)
HP-UX	libbar.a (archive) libbar.sl (shared library)
AIX	libbar.a (archive) libbar.so (shared library)

## Sample Source code

The API includes source code for a sample program. As the sample program on the next page shows, the `dri ver` program is located in separate subdirectories under the `src` directory. In Windows environments, the API includes source code for the `dri ver` program only.

## Working with makefiles

While makefile content is generally the same in both UNIX and Windows environments, the process of creating and editing makefiles is different. The `dri ver` program source code includes makefiles that you can use as templates for creating your own makefiles.

## UNIX

The following sample program shows a typical set of flags in the UNIX environment.

```
# Makefile.h /* sample makefile for HP platform */

# Parameters
PROGRAM=daysOpen
SOURCES=daysOpen.c
OBJECTS=daysOpen.o

# Compiler flags
CC= c89
DEBUG= -g
CFLAGS= $(DEBUG) -DHP -D_REENTRANT -Wc, -DA1.0, -DS1.0, -Aa \
        -I/usr/include -I.. /include -I.. /include \
        -I.. /.. /include
LDLIBS= -L.. /lib -L.. /.. /lib -L.. /.. /.. /lib
ARCHLIBS= -Ldce
ARLIB= -Lar

# Standard targets
all: $(PROGRAM)
objects: $(OBJECTS)
$(PROGRAM): $(OBJECTS)
        $(CC) -o $(PROGRAM) $(OBJECTS) $(LDLIBS) $(ARCHLIBS)
        $(ARLIB)

clean:
        $(RM) $(OBJECTS) core
```

---

**Note:** See your system manual to find out if you must place library files in a specific order in the makefile. With Solaris, you must include the `-L` pthread switch for the makefile line that starts with `LDLIBS`.

---

## Windows

Creating and editing makefiles in the Windows environment is considerably simpler. The Microsoft Visual C++ compiler creates makefiles for you and provides a graphical interface for specifying the options you want. You specify the appropriate makefile when you compile your program:

```
nmake /f <filename>.mak
```

The `dri ver.mak` file that comes with the `dri ver` program contains a sample makefile for Windows.



Chapter

# 3

# Data structures

This chapter describes the most common elements and functions of AR System C API key data structures. See the header files listed in “[Sample Source code](#)” on page 36 for complete data type and structure specifications.

The following topics are provided:

- Data relationships ([page 40](#))
- C data types ([page 41](#))
- Lists ([page 42](#))
- High-level object relationships ([page 43](#))
- Login and session information ([page 45](#))
- Status information ([page 48](#))
- Permission information ([page 51](#))
- Representing values ([page 53](#))
- Representing qualifications ([page 55](#))
- Schemas ([page 61](#))
- Fields ([page 63](#))
- Entries ([page 70](#))
- Filters, escalations, and active links ([page 77](#))
- Character menus ([page 100](#))
- Containers ([page 102](#))
- Importing and exporting ([page 109](#))

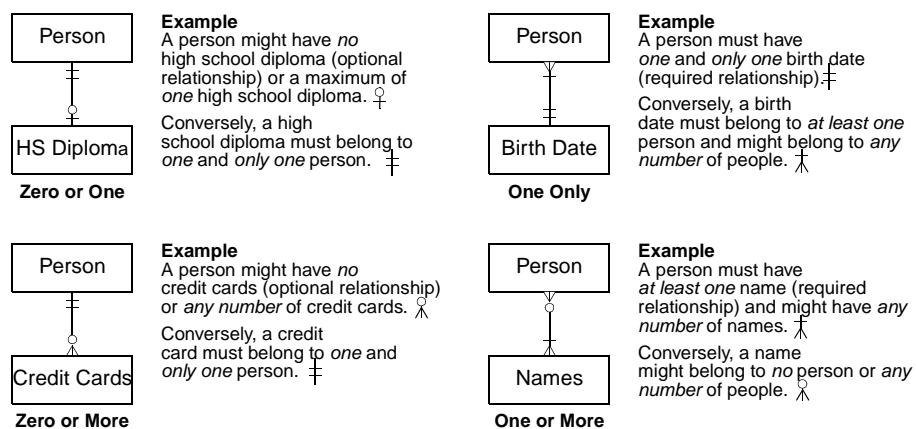
- Freeing allocated memory (page 112)
- XML formats (page 114)

## Data relationships

The AR System C API contains structures that store both simple and complex information. Structures that store simple information, such as the type of value or the product of an arithmetic operation, serve as the building blocks for complex structures. This chapter contains a series of data structure diagrams that explain these hierarchical relationships. The diagrams also identify which structures, or groups of structures, the AR System uses to manipulate objects.

The following figure explains the data relationship notation.

Figure 3-1: Data relationship notation




---

**Note:** For all data structure diagrams in this chapter, lack of notation indicates a *one only* relationship.

---

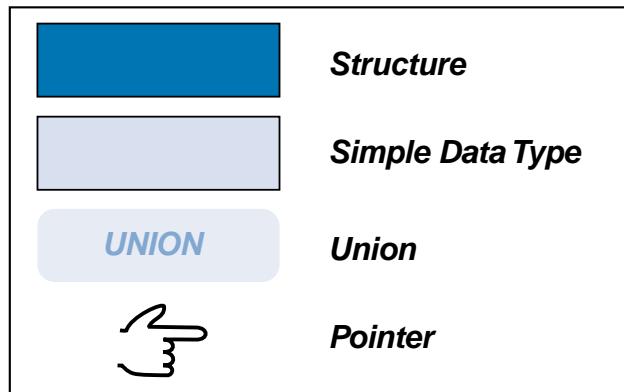
# C data types

The AR System API uses simple C data types in addition to the following AR System data types:

Data type	Definition
typedef char ARAccessNameType[AR_MAX_ACCESS_NAME_SIZE + 1]	Structure that holds a user or group name or password.
typedef char ARPasswordType[AR_MAX_PASSWORD_SIZE + 1]	Structure that holds a password.
typedef char ARAAuthType[AR_MAX_AUTH_SIZE + 1]	Structure that holds an authentication string.
typedef char AREntryIdType[AR_MAX_ENTRY_ID_SIZE + 1]	Structure that holds an entry identification value.
typedef char ARLocal eType[AR_MAX_LANG_SIZE + 1]	Locale; language-country.
typedef char ARNameType[AR_MAX_NAME_SIZE + 1]	Structure that holds object names.
typedef char ARServerNameType[AR_MAX_SERVER_SIZE + 1]	Structure that holds a server name.
typedef ARLong32 ARTimestamp	UNIX style time stamp; seconds since Jan. 1, 1970.
typedef unsigned char ARBool	Boolean flag set to TRUE or FALSE.
typedef ARLong32 ARInternalId	Structure that holds an internal identification value.
typedef int ARLong32	A 32-bit signed integer for either 32-bit or 64-bit machines. This data type is exactly 32 bits and replaces the long data type in the C API. For more information, see the artypes.h file.
typedef unsigned int ARULong32	A 32-bit unsigned integer for either 32-bit or 64-bit machines. This data type is exactly 32 bits and replaces the unsigned long data type in the C API. For more information, see the artypes.h file.

The legend in the following figure applies to all data structure diagrams in this chapter. For simplicity, the data types defined here are considered simple types.

Figure 3-2: Legend for data structure diagrams



The purpose of the following structure diagrams is to illustrate the relationships between various data structures. Some of the simple data elements contained within a structure are not shown. Not all structures are broken down to the level of simple data types.

Unless otherwise specified, see the `.ar.h` file for complete structure and variable definitions.

## Lists

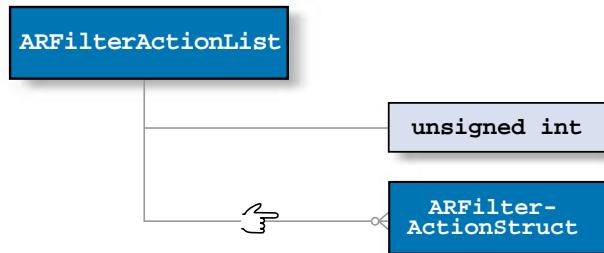
Lists manipulate sets of objects of a particular type. For example, `ARNameList` manipulates any set of names. `ARDiaryList` manages the group of entries contained in a diary field. `AREntryList` represents any group of entries in a schema. `ARFilterActionList` defines the set of actions that the server performs when it executes a filter or escalation.

While each type of list has its own data structure, all list structures have the same form:

```
typedef struct {
    unsigned int numItems;
    AR<xxx>Struct*<xxx>List;
} AR<xxx>List;
```

Each list structure consists of an integer that identifies the number of items in the list and a pointer to the allocated memory containing those items. If the number of items is zero, the server or client ignores the pointer and the pointer is usually set to `NULL`. The following figure shows an example that uses the `ARFilterActionList` structure.

Figure 3-3: Generic list structure




---

**Note:** For simplicity, subsequent structure diagrams do not illustrate the `numItems` element (`unsigned int`) for any list structures. All list structures in the API have the number of items in the list as their first component.

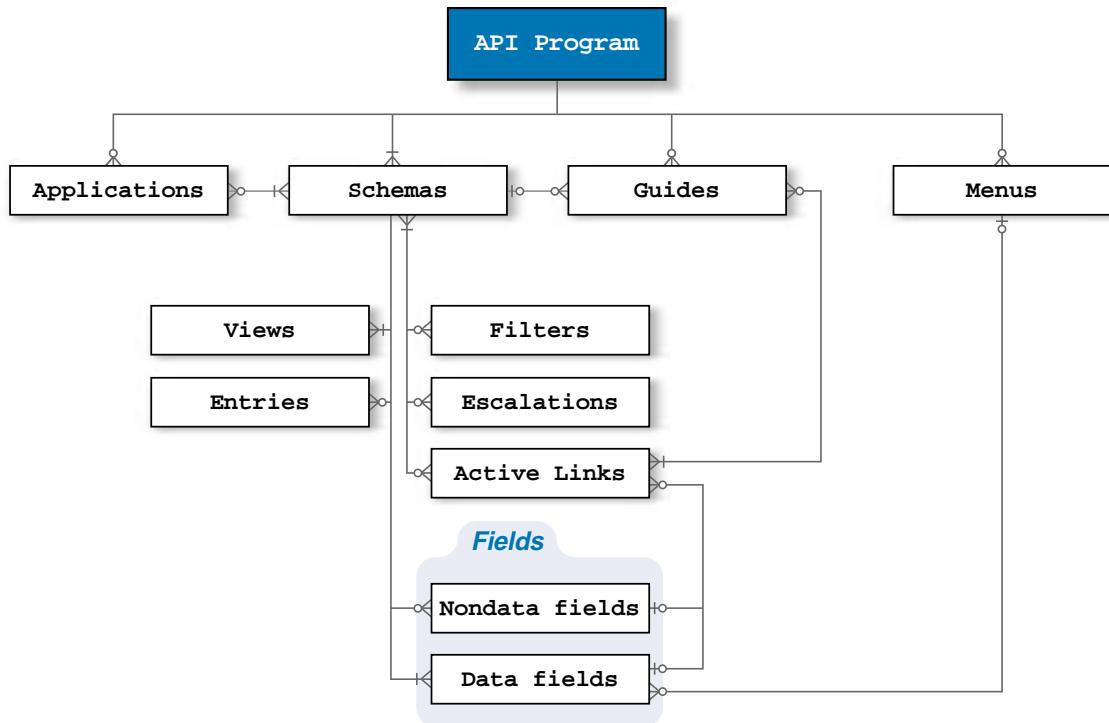
---

In general, the AR System server handles lists as arrays instead of linked lists. It then stores all items in the structure in a single area of memory, with the pointer identifying the memory address of the first item in the array. If the item structure itself (`ARFilterActionStruct` in page 82) contains a pointer, the server allocates the nested memory separately.

## High-level object relationships

Before discussing specific data structures, it is helpful to understand the high-level relationships between AR System objects (see the *Form and Application Objects* guide for definitions and descriptions of these objects). As the following figure shows, the foundation of all client programs built on the AR System is the schema. Although the AR System API still uses schema, the AR System clients now see schemas as forms. A client must access at least one and can access many schemas.

Figure 3-4: High-Level object relationships



Note: For a description of the notation used to represent data relationships, see “Data relationships” on page 40.

Your client program can also have optional **menus** and **containers**. Either servers or schemas (forms) can own containers, which the server uses to create **guides** and **applications**. Applications can contain one to many schemas. Guides can be shared and can contain one to many active links.

There are five types of schemas. The most common, **data schemas**, contain data in the AR System database. **Join schemas** show data from two or more data schemas with a common matching field, and **display-only schemas** show data from one data schema. These latter two schema types do not contain data themselves. **View schemas** show data from external database tables and act like data schemas. **Vendor schemas** acquire data from the plug-in service and act like data schemas. When you modify values in a join schema, you are changing the data in its parent data schemas.

Each schema must have at least one **view** (the default view) but can have many views. Similarly, each data schema must have at least one **field** but is likely to have many fields. (In actual practice, every data schema is automatically created with nine *core fields* that cannot be deleted. For more information about core fields, see the *Form and Application Objects* guide.)

Fields come in two types—**data fields** and **nondata fields** (trim, control, page, and table). A schema must have at least one data field. Nondata fields are optional.

Most schemas have many **entries**, but a schema can exist without an entry. Although the AR System API still uses this term, the AR System clients now see entries as **requests**. Similarly, any number of **filters**, **escalations**, or **active links** can be associated with a schema or list of schemas, but these objects are not required. See “Schemas” on page 61 for more information.

Both active links and menus can be associated with a field. While active links can be associated with either data or control fields, menus can only be associated with data fields. The relationships between them, however, are exactly reversed.

An active link must be associated with a particular schema. A data or control field can have any number of active links associated with it. A particular execute-on condition of an active link can only be associated with one field. Different execute-on conditions in the same active link can reference different fields.

A menu can be associated with *any number* of data fields, both within an individual schema and across multiple schemas. However, a data field can have only *one* menu associated with it.

## Login and session information

Almost every AR System C API function has a `control` parameter as its first input argument. This parameter contains the login and session information necessary for connecting to an AR System server and, thus, is required for almost every operation.

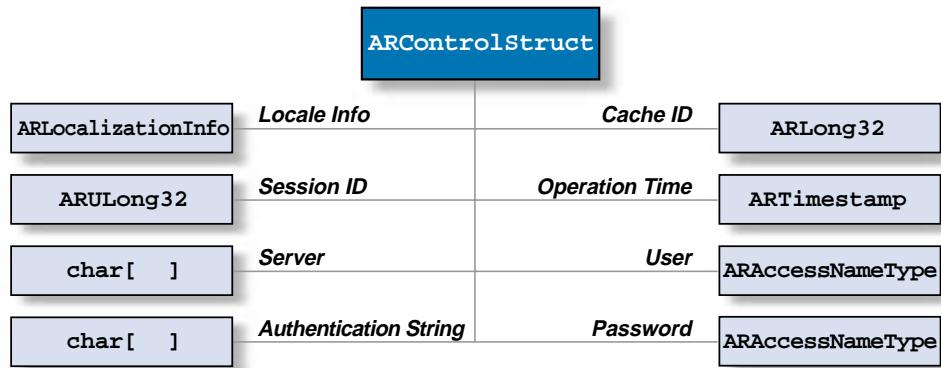
---

**Note:** The control parameter was optional in AR System version 3.x and earlier. Therefore, you must add the control parameter to recompiled pre 4.x API programs if you use these programs with later versions of the AR System API.

---

The control parameter is a pointer to an ARControl Struct structure, as the following figure shows.

Figure 3-5: Structure used to provide required login information



This structure consists of the following elements:

<b>Cache ID</b>	Identifies the cache area allocated by the server to store user information. It should be initialized to zero before the first API call. This value is assigned by the server and should not be changed, enabling the server to use the cache instead of reloading user information for each API call. Required.
<b>Operation Time</b>	A time stamp identifying the date and time the operation occurred on the server. The server assigns this value for each API call.
<b>User</b>	The login name to use when connecting to the server. The privileges associated with this user determine whether the API function call can be performed. Required.
<b>Password</b>	The password for the specified user name, in clear text. The API encrypts this parameter before sending it to the server. Required.

<b>Locale Information</b>	A string that specifies the language of returning error messages (if a message catalog exists), formatting date and time information, sorting or comparing values, and locale information. The locale information will be in the form: <code>language[_territory[.codeset]][@modifier]</code> for example: en_US.1508859-15@euro The string should match the language strings that setlocale function uses. Default is a C or an empty string. Required.
<b>Session ID</b>	An identifier for the session control block. The session control block is a structure containing all data needed to maintain the state of API operations. Each API session has a unique session control block, created when you initialize the session by calling <code>ARNitalization</code> .
<b>Server</b>	A string specifying the name of the server to connect to. Required.
<b>Authentication String</b>	The authentication string that the client provides, such as the domain.

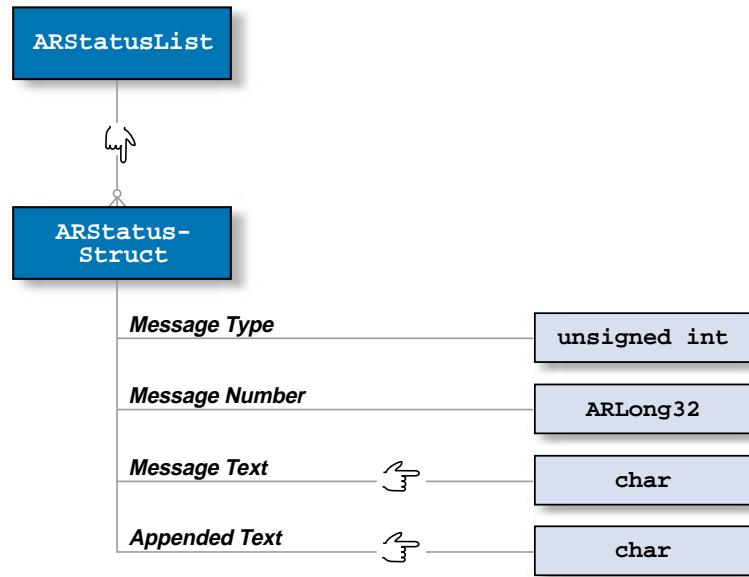
Nearly all function calls require the login and session information that the `ARControl Struct` contains (stored in both single- and multiple-server environments) because the API does not always maintain a server connection between calls.

When you call `ARNitalization` at the beginning of your program, the AR System C API returns the data in `ARControl Struct`. This is the structure that you pass as an input parameter in subsequent API calls.

# Status information

Nearly every API function has a `status` parameter as its last return value. This parameter contains status information about the success or failure of the call. The `status` variable is a pointer to an `ARStatusList` structure, as the following figure shows.

Figure 3-6: Structure used to return function status



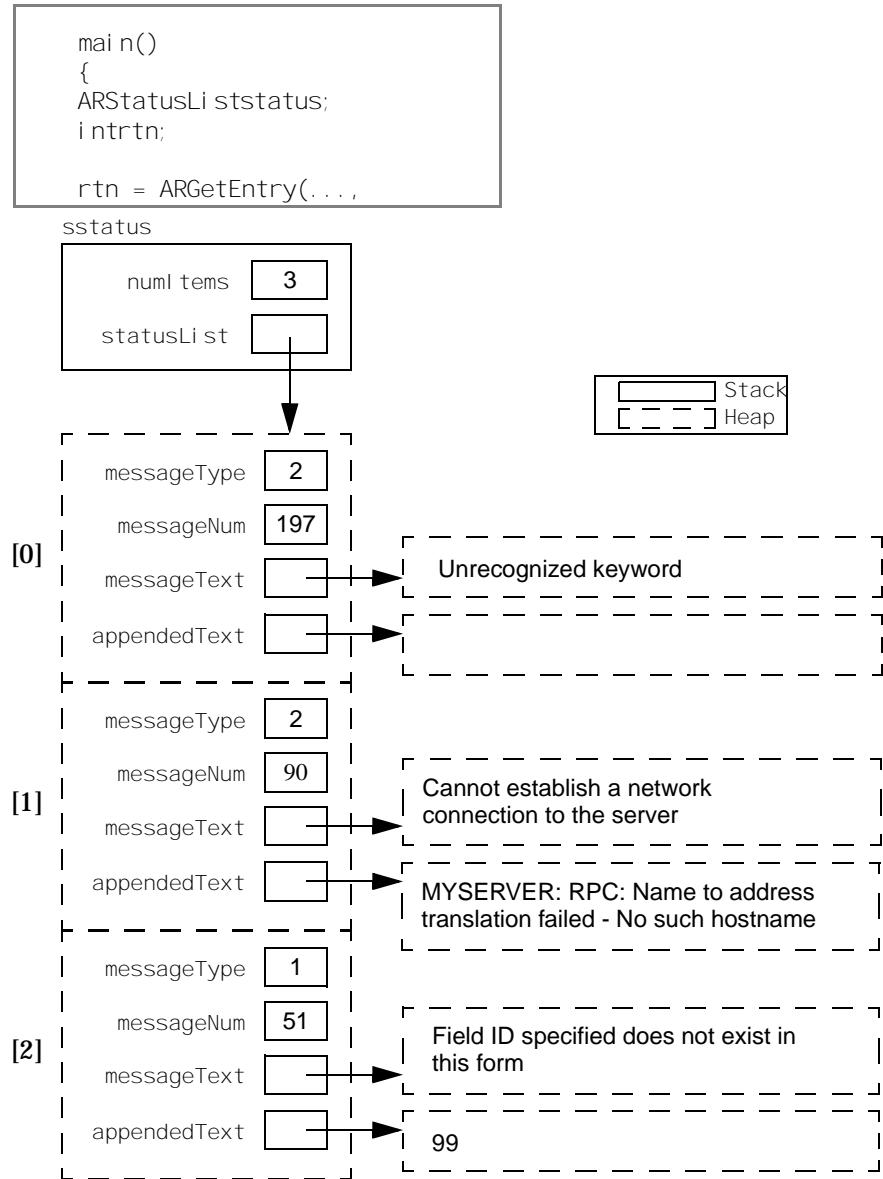
Like all list structures in the AR System (see “Lists” on page 42), ARStatusList consists of zero or more ARStatusStruct items. Each ARStatusStruct item represents a message that the function call generates, and consists of the following elements:

<b>Message Type</b>	An integer value indicating the type of message being returned. The following table describes the possible values.
<b>Message Number</b>	The error number associated with the message (defined in arerrno.h).
<b>Message Text</b>	The message text in the language that the ARControl Struct specifies (if possible). To get the text of the message in the local language, call ARGetTextForErrorMessage.
<b>Appended Text</b>	Text that augments the message text. This text might come from the AR System server, the operating system, or database management system. The AR_MAX_MESSAGE_SIZE limits the length to 255 bytes.

0 AR_RETURN_OK	Operation successful—status <i>might</i> contain one or more <i>informational</i> messages.
1 AR_RETURN_WARNING	Operation successful, but some problems encountered—status contains one or more <i>warning</i> messages and <i>might</i> also contain informational messages.
2 AR_RETURN_ERROR	Operation failed—status contains one or more <i>error</i> messages and <i>might</i> also contain warning or informational messages.
3 AR_RETURN_FATAL	Operation failed—status <i>might</i> contain one or more messages.
4 AR_RETURN_BAD_STATUS	Invalid status parameter—operation cancelled.
5 AR_RETURN_PROMPT	Status for the active link action.
6 AR_RETURN_ACCESSIBLE	Status message for client accessibility.

The following figure shows the physical layout of the ARStatusList structure.

Figure 3-7: Example of ARStatusList structure



In this example, the `status` parameter is defined as a stack variable of type `ARStatusList` and consists of an integer value identifying the number of messages in the list (three in this case) and a pointer to their location on the heap. This memory space is allocated dynamically by the system, based on the size and number of messages being returned.

The messages themselves are stored in an array of `ARStatusStruct` structures, each of which contains the message type, the message number, and pointers to the message text and any appended text. The messages are sorted in decreasing order of severity, based on the message type value (see “Error checking” on page 441). The return value for the function is the message type associated with the first (most recent and most serious) message in the list. In this case, the function return value is 2.

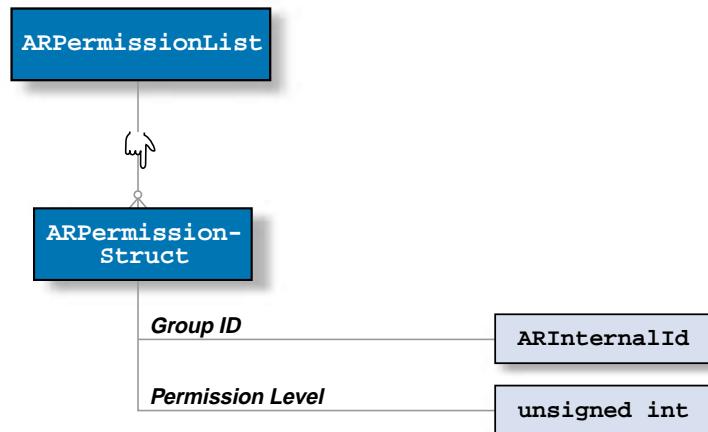
Use standard array notation to access individual elements in the list. For example, `status.statusList[0].messageText` identifies the message text associated with the first message, while `status.statusList[1].messageNum` gives you the message number for the second message

After a returned status list is processed, a program needs to call `FreeARStatusList` to free any memory that was allocated. If `numItems` is equal to zero, the call is optional.

## Permission information

Permission for an AR System object is generally defined by simply specifying the list of groups that can access it (by using the `ARIInternalList` structure). The `groupList` parameter for the active link functions is of this type. For schemas, fields, and containers, however, you can assign different levels of access to different groups. Permission for these objects is defined by the `groupList` (schemas and containers) and `permissions` (fields) parameters, both of which are pointers to structures of type `ARPermissionsList`, as the following figure shows.

Figure 3-8: Structures used to define field and schema permissions



Each **ARPermission** in the list represents a particular access control group and consists of the following elements:

- |                         |   |
|-------------------------|---|
| <b>Group ID</b>         | The internal ID of the group.   |
| <b>Permission Level</b> | An integer value indicating the access privileges for the group. The following tables describe the possible values for schemas, containers, and fields. |

### Schema/Container permission values for ARPermissionStruct

0	AR_PERMISSIONS_NONE	Group has no access to the schema or container.
1	AR_PERMISSIONS_VISIBLE	Group can see the schema or container.
2	AR_PERMISSIONS_HIDDEN	Group cannot see the schema or container.

### Field permission values for ARPermissionStruct

0	AR_PERMISSIONS_NONE	Group has no access to the field.
1	AR_PERMISSIONS_READ	Group has read-only access to the field.
2	AR_PERMISSIONS_CHANGE	Group has read-write access to the field.

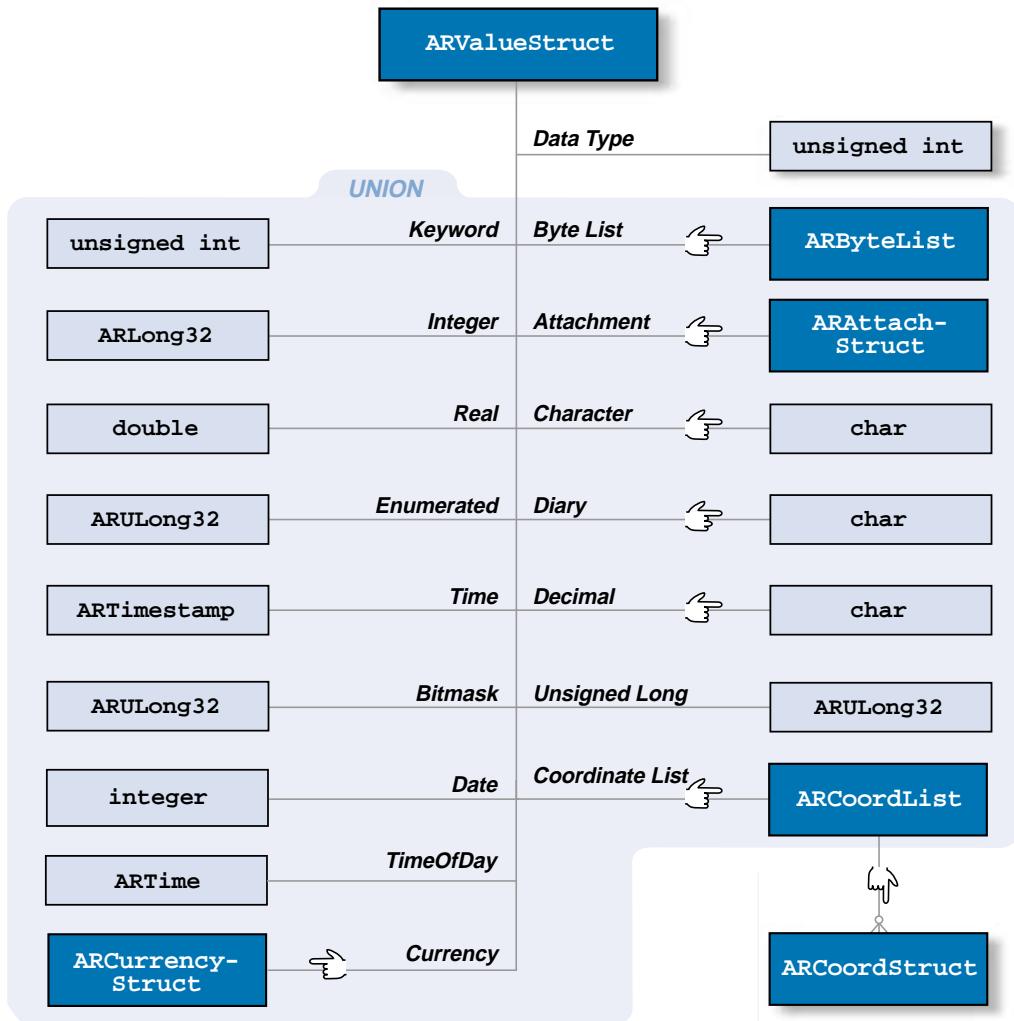
Different privileges are associated with the possible values, depending on whether you are assigning permission to a schema or a field.

# Representing values

One of the most frequently used data structures in the API is `ARValueStruct`. This structure stores values of any data type.

Many calls, such as `ARCreateField` and `ARGetField`, take parameters of type `ARValueStruct` directly. Other calls use parameters that contain `ARValueStruct`, such as `ARQualifier` and `ARFilterAction`.

Figure 3-9: Structures used to represent any value



ARValueStruct consists of the following elements:

Data Type	An integer value indicating the data type of the value being passed. The following table describes the possible values.
Value	The value itself (represented by using data types or structures appropriate to the type of value).
0 AR_DATA_TYPE_NULL	Specifies NULL value.
1 AR_DATA_TYPE_KEYWORD	An integer identifying the particular keyword (defined in the ar.h file).
2 AR_DATA_TYPE_INTEGER	A 32-bit signed integer value.
3 AR_DATA_TYPE_REAL	A 64-bit floating-point value.
4 AR_DATA_TYPE_CHAR	A null-terminated string that requires freeing allocated space. A NULL pointer of this type is equivalent to using AR_DATA_TYPE_NULL.
5 AR_DATA_TYPE_STRING	A null-terminated string that requires freeing allocated space. A NULL pointer of this type is equivalent to using AR_DATA_TYPE_NULL.
6 AR_DATA_TYPE_ENUM	A set of integer values (beginning with zero) that represents possible selection values as an indexed list. You must specify a field limit by using ARNameList to define string values for each selection (see “Lists” on page 42).
7 AR_DATA_TYPE_TIME	A UNIX-style date/time stamp (number of seconds since midnight January 1, 1970).
8 AR_DATA_TYPE_BITMAP	A 32-bit unsigned integer value in which each bit represents a flag turned on or off.
9 AR_DATA_TYPE_BYTES	A list of bytes containing binary data (represented by using the ARByteList structure).
10 AR_DATA_TYPE_DECIMAL	A fixed-point decimal field. Values must be specified in C locale, for example 1234.56
11 AR_DATA_TYPE_ATTACH	An attachment field.
12 AR_DATA_TYPE_CURRENCY	A currency field.
13 AR_DATA_TYPE_DATE	A date field.
14 AR_DATA_TYPE_TIME_OF_DAY	Time of day field.
37 AR_DATA_TYPE_ATTACH_POOL	A pool for grouping attachments.
40 AR_DATA_TYPE ULONG	A 32-bit unsigned integer value.

---

41	AR_DATA_TYPE_COORDS	A list of (x,y) coordinate pairs.
42	AR_DATA_TYPE_WEBVIEW	Data type of a web view field.
43	AR_DATA_TYPE_DDISPLAY	Data type of a dashboards field.

---

**Note:** When passing a diary value to ARCreate or ARSet<object> functions, the ARValueStruct value is the *additional* text to append to the diary field. When retrieving diary values or passing a diary value to ARMergeEntry, the ARValueStruct value is a diary-formatted string containing the full diary text. Use the ARDecodeDiary function to decode this string into an array of time-stamped entries.

---

## Representing qualifications

Many tasks in the AR System use qualifications. For example, a qualification (or lack thereof) determines which entries to retrieve when creating a query result list (ARGetListEntry) or computing entry statistics (ARGetEntryStatistics).

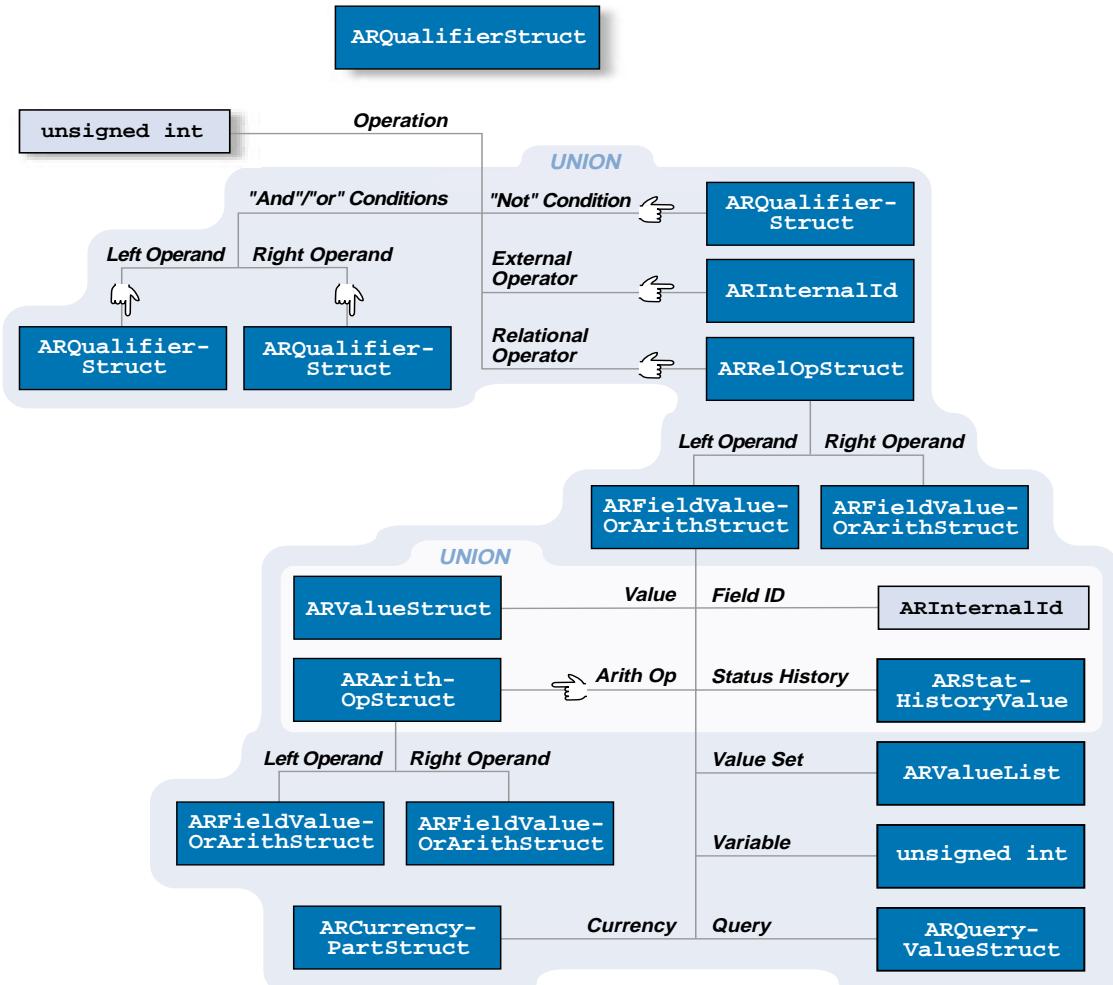
A qualification also determines whether an active link or filter is executed and which entries an escalation acts upon.

Every qualification is composed of a set of zero or more conditions that limit the set of entries retrieved. As the next figure shows, ARQualifierFilterStruct represents each individual condition by using a tree structure composed of an operation and one or two operands (depending on the type of operation).

The operands can point to other structures of type ARQualifierFilterStruct. This recursion allows you to represent any qualification by using a series of nested ARQualifierFilterStruct structures.

You can avoid the complexity of storing a qualification in this structure by using ARLoadARQualifierFilterStruct (see page 369). This function takes the specified qualification string and loads it into an ARQualifierFilterStruct. If you are comfortable working with the ARQualifierFilterStruct structure or your query is static, you might find that using the ARQualifierFilterStruct directly can yield improved performance.

Figure 3-10: Structures that represent any qualification



You can also use the following calls to manipulate the ARQualifierStruct structure:

- ARDecodeARQualifierStruct, which converts information in a serialized qualification string into an ARQualifierStruct structure.
- AREncodeARQualifierStruct, which converts an ARQualifierStruct structure into a serialized qualifier string.

The ARQual i fi erStruct structure consists of the following elements:

<b>Operation</b>	An integer value indicating the type of <i>conditional</i> operation to perform. The following table describes the possible values.
<b>Operands</b>	The components of the operation (represented by using structures appropriate to the type of value).

0	AR_COND_OP_NONE	No qualification.
1	AR_COND_OP_AND	A qualification using the AND operator.
2	AR_COND_OP_OR	A qualification using the OR operator.
3	AR_COND_OP_NOT	A qualification using the NOT operator.
4	AR_COND_OP_REL_OP	A qualification using a relational operator.
5	AR_COND_OP_FROM_FIELD	A qualification located in a field on the form.

If a query has no qualification criteria, you can either set the operation type to zero (AR\_COND\_OP\_NONE) or pass `NULL` for the qual i fi er parameter.

## Dynamic qualifications

Dynamic qualifications take part of the qualification condition from a field whose contents can change. The part of the qualification that is located in a field uses the AR\_COND\_OP\_FROM\_FIELD tag. The value supplied is the ID of the field that contains the qualification.

## Qualifications that use conditional operators

Qualifications that use the AND or OR operators require two operands, while qualifications that use the NOT operator require one. In both cases, the operands consist of pointers to nested ARQual i fi erStruct structures.

## Qualifications that use relational operators

Qualifications that use a relational operator require one operand, consisting of a pointer to an ARRel OpStruct structure. The ARRel OpStruct structure consists of a tag identifying the operation type and the following operands specifying the values to compare:

**Operation** An integer value indicating the type of *relational* operation to perform. The following table describes the possible values.

**Operands** The components of the operation (represented by using the ARFi el dVal ueOrAri thStruct structure).

1 AR_REL_OP_EQUAL	Tests whether the left operand is equal to the right operand.
2 AR_REL_OP_GREATER	Tests whether the left operand is greater than the right operand.
3 AR_REL_OP_GREATER_EQUAL	Tests whether the left operand is greater than or equal to the right operand.
4 AR_REL_OP_LESS	Tests whether the left operand is less than the right operand.
5 AR_REL_OP_LESS_EQUAL	Tests whether the left operand is less than or equal to the right operand.
6 AR_REL_OP_NOT_EQUAL	Tests whether the left operand is not equal to the right operand.
7 AR_REL_OP_LIKE	Tests whether the left operand is LIKE the pattern defined by the right operand.

## Defining operands for a relational operation

The values to compare in a relational operation are represented by using the ARFi el dVal ueOrAri thStruct structure. This structure has the following elements:

<b>Tag</b>	An integer value indicating the type of value. The following table describes the possible values.
<b>Value</b>	The value itself (represented by using data types or structures appropriate to the type of value).

1	AR_FIELD	A schema field value.
2	AR_VALUE	A constant or keyword value represented by using ARVal ueStruct (see page 52).
3	AR_ARITHMETIC	A result value from an arithmetic operation represented by using ARArithOpStruct (see "Defining an arithmetic result value").
4	AR_STAT_HISTORY	A value from the Status-History core field represented by using ARStatHistoryValue (see "Defining a status history value").
5	AR_VALUE_SET	A list of values to set.
6	AR_CURRENCY_FLD	A schema currency field value represented by using ARCurrencyPartStruct.

## Defining an arithmetic result value

ARArithOpStruct represents the result value from an arithmetic operation. Similar in form to ARRelOpStruct (see page 58), this structure consists of a tag identifying the operation type and two operands specifying the values:

<b>Operation</b>	An integer value indicating the type of <i>arithmetic</i> operation to perform. The following table describes the possible values.
<b>Operands</b>	The components of the operation (represented by using the ARFi el dVal ueOrAri thStruct structure).

1	AR_ARITH_OP_ADD	Add the left and right operands.
2	AR_ARITH_OP_SUBTRACT	Subtract the right operand from the left operand.
3	AR_ARITH_OP_MULTIPLY	Multiply the left and right operands.

---

4	AR_ARITH_OP_DIVIDE	Divide the left operand by the right operand.
5	AR_ARITH_OP_MODULO	Find the remainder after dividing the left operand by the right operand.
6	AR_ARITH_OP_NEGATE	Change the sign of the right operand (left operand is ignored).

---

## Defining a status history value

The Status History core field is a special selection field that stores user and time stamp information for each of the defined status values. You can specify a value from this field by using the ARStatHi storyValue structure. This structure consists of the following elements:

- Index** An enumerated value identifying the particular status value to use in the operation.
- Tag** An integer value indicating which information (about that status) to use. The following table describes the possible values.

---

1	AR_STAT_HI_STORY_USER	The user who assigned the specified status.
2	AR_STAT_HI_STORY_TIME	The time when the status was assigned.

---

## Defining a currency field part

Currency fields consist of several parts that combine to represent a complete currency value. You can specify an individual part from this field type or reference the entire field by using the ARCurrencyPartStruct structure. This structure consists of the following elements:

- Field ID** The internal ID of the currency field (see the *Form and Application Objects* guide for information about field IDs).
- Part Tag** An integer value that indicates the type of currency part. The following table describes the possible values.
- Currency Code** A character string that contains the currency code if the Part Tag element is set to AR\_CURRENCY\_PART\_FUNCTIONAL.

---

0	AR_CURRENCY_PART_FIELD	Entire currency field.
1	AR_CURRENCY_PART_VALUE	The numeric currency value.
2	AR_CURRENCY_PART_TYPE	The currency code associated with the currency value.

---

---

3	AR_CURRENCY_PART_DATE	The currency conversion date.
4	AR_CURRENCY_PART_FUNCTIONAL	One of the defined functional currencies.

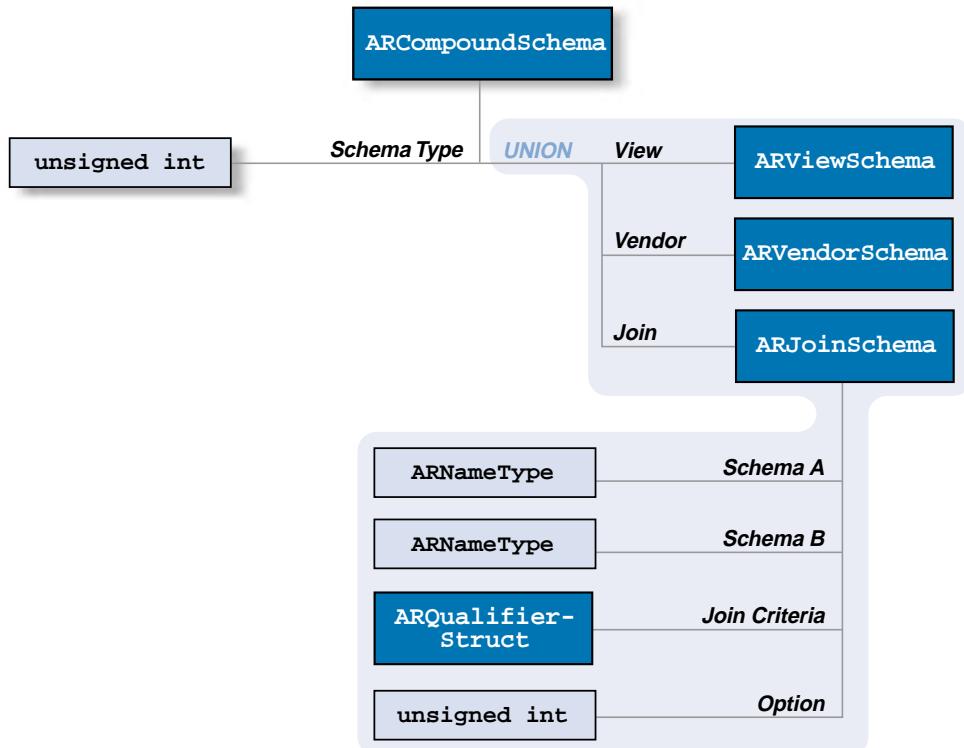
---

## Schemas

The AR System C API still uses the term schema but the AR System clients now see schemas as **forms**.

Schemas are represented by using the `ARCompoundSchema` structure, as this figure shows.

Figure 3-11: Structures used to create join schemas



The ARCompoundSchema structure consists of the following elements:

<b>Schema Type</b>	An integer value indicating the type of schema. The following table describes the possible values.
<b>Join Definition</b>	A union that defines the schema. Join schemas are defined by an ARJoinSchema structure containing the names of the two member schemas (either of which could also be join schemas), the criteria for joining them, and a bitmask indicating various join options.
<b>View Definition</b>	An integer value indicating the type of view. The following table describes the possible values.
<b>Vendor Definition</b>	An integer value indicating the type of vendor form. The following table describes the possible values.

1	AR_SCHEMA_REGULAR	Base schema.
2	AR_SCHEMA_JOIN	Join schema (has two base schemas).
3	AR_SCHEMA_VIEW	View schema.
4	AR_SCHEMA_DISPLAY	Display-only schema (contains only display-only fields).
5	AR_SCHEMA_VENDOR	Vendor schema.

The join options in the following table define the join type and the action to take when users modify entries (see “ARSetEntry” on page 384).

1	AR_JOIN_OPTION_OUTER	Outer join.
0	AR_JOIN_SETOPTION_NONE	Allow users to update fields used in the join criteria.
1	AR_JOIN_SETOPTION_REF	Do not allow users to update fields used in the join criteria.
0	AR_JOIN_DELOPTION_NONE	Individual entries deleted only when the entry can be retrieved through the join schema.
1	AR_JOIN_DELOPTION_FORCE	Individual entries deleted even when the entry cannot be retrieved from the join schema. The AR System server ignores errors for entries that no longer exist.

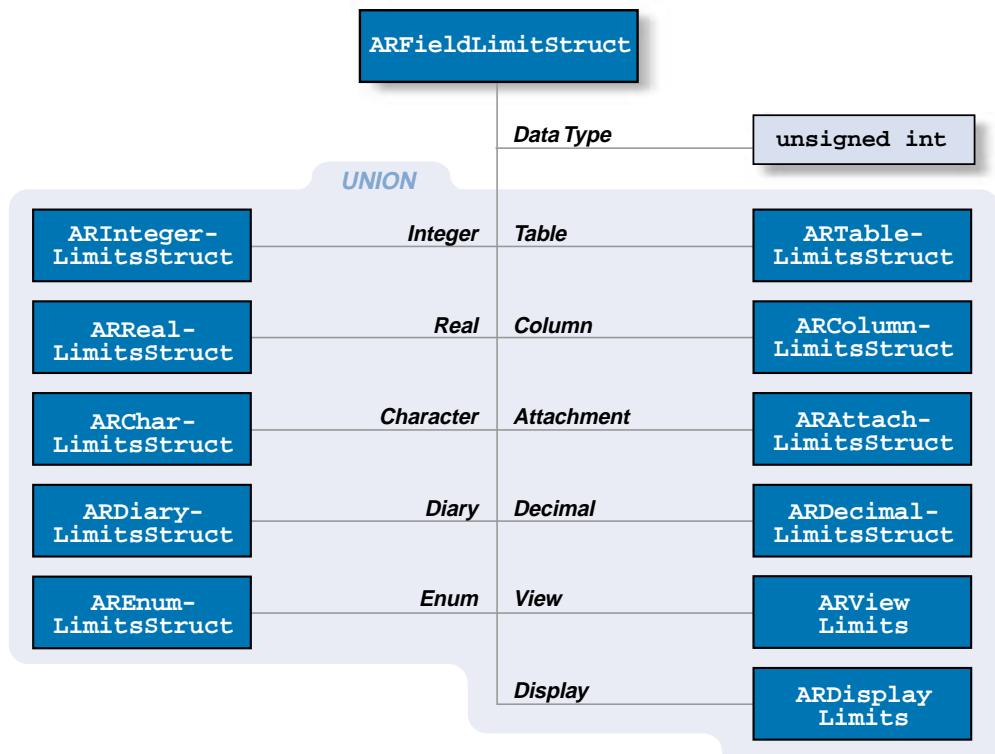
# Fields

The following section describes some of the structure types related to creating, retrieving, and modifying field definitions. Each of these structures is used as a parameter type for the ARCreateField, ARGetField, and ARSetField functions (see “ARCreateField” on page 143, “ARGetField” on page 233, and “ARSetField” on page 389).

## Defining field limits

The `Limit` parameter defines the value limits for a data field, with the data type of the field determining the type of limits you can specify. This parameter is a pointer to an `ARFieldLimitStruct` structure, as the following figure shows.

Figure 3-12: Structures used to define field value boundaries



ARFi el dLi mi tStruct enables you to define value limits for data fields of any type, much like ARVal ueStruct enables you to specify values of any data type (see Figure 3-9 on page 53). The ARFi el dLi mi tStruct structure consists of the following elements:

<b>Data Type</b>	An integer value indicating the data type of the field. The following table describes the possible values.
<b>Limit</b>	The actual limits to assign (represented by using structures appropriate to the type of value).

0 AR_FIELDLIMIT_NONE	Field has no defined limits.
2 AR_DATA_TYPE_INTEGER	Lower- and upper-range limits, defined by using ARIntegerLi mi tsStruct.
3 AR_DATA_TYPE_REAL	Lower- and upper-range limits, defined by using ARReal Li mi tsStruct. You can also specify the display precision to use.
4 AR_DATA_TYPE_CHAR	Maximum field length, defined by using ARCharLi mi tsStruct. Specify zero to indicate no maximum. You can also specify a character menu to attach (and whether selecting from the menu appends or overwrites text already in the field), a default query-by-example qualification type, a field character pattern, and whether the field is indexed for full text search (FTS).
5 AR_DATA_TYPE_DIARY	Specifies whether the field is indexed for FTS by using ARDiaryLi mi tsStruct.  <b>Note:</b> See the Input Length database property in the <i>Form and Application Objects</i> guide for more information about storing long character strings.

---

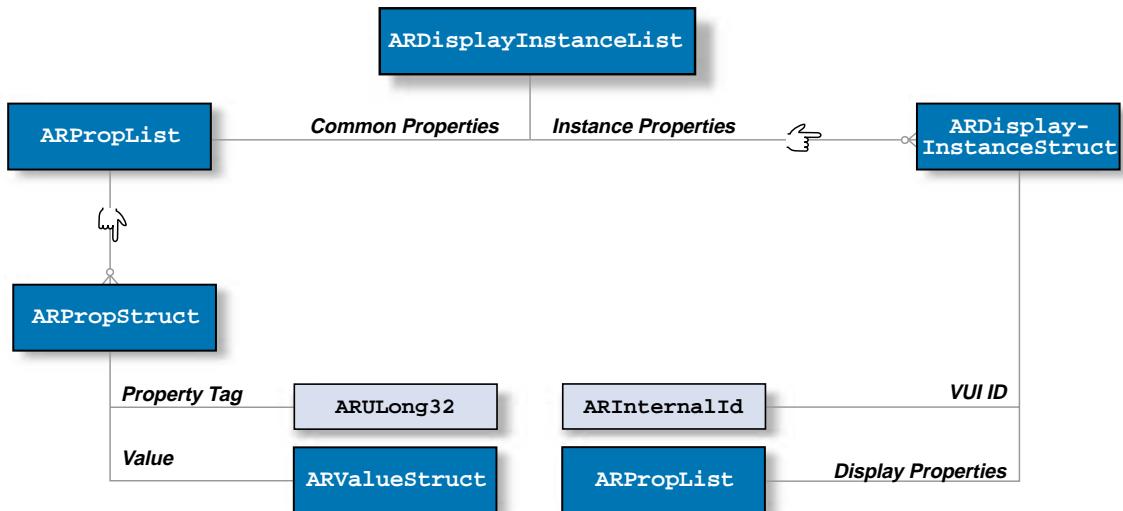
6 AR_DATA_TYPE_ENUM	Values for an enumerated list, defined by using AREnumListStruct ( <i>required</i> ). Only AR_ENUM_STYLE_REGULAR is implemented. The set of integer values (beginning with zero) represents possible selection values; and the field value limit is defined using ARNameList as a string of values for each selection. See “Lists” on page 42 for the form of the list structure.
10 AR_DATA_TYPE_DECIMAL	Lower- and upper-range limits, defined by using ARDecimalListLimitStruct. You can also specify the precision.
11 AR_DATA_TYPE_ATTACH	Maximum size of the attachment and attachment type (embedded is the only type supported at this time), defined by using ARAttachListStruct.
12 AR_DATA_TYPE_CURRENCY	Schema currency field value. Lower- and upper-range limits, precision, allowable currency definitions, and functional currency definitions, defined by using ARCurrencyListStruct.
13 AR_DATA_TYPE_DATE	Lower- and upper-range limits, defined by using ARDateListStruct.
33 AR_DATA_TYPE_TABLE	Number of columns, search qualification, maximum number of rows to retrieve, schema name, and server name of the table, defined by using ARTableListStruct.
34 AR_DATA_TYPE_COLUMN	Parent field ID, data field ID (the remote field ID from the data source), data source (the data source for the data field, which can be a data field on a display-only field), and number of characters to display, defined by using ARColumnListStruct.
42 AR_DATA_TYPE_VIEW	View field, defined by using ARViewListStruct.
43 AR_DATA_TYPE_DISPLAY	Values for dashboards for display fields defined by using ARDisplayListStruct.

---

## Defining field display properties

The display properties associated with a field fall into two categories—those that are common to all schema views and those that are specific to a particular schema view. Both types are represented as lists of zero or more properties by using the ARPropList structure. The *common* display properties are collected in one ARPropList structure. The display *instance* properties are collected in a series of additional ARPropList structures, but each of these is linked to a particular view (VUI). These view-specific property lists are represented by zero or more ARDisplayList instanceStruct structures. All of these structures are then collected in an ARDisplayList instanceList structure, as the following figure shows.

Figure 3-13: Structures used to define field display properties



The `dlInstanceList` parameter is a pointer to a structure of this type and is used to pass all field display properties when creating, retrieving, or modifying field definitions.

The common display properties are represented by using an embedded `ARPropList`. This list contains zero or more items, each of which represents one display property. The `ARPropStruct` structure contains the following elements:

**Property Tag** An integer value identifying the particular display property. See See “Server object property tags” on page 79 for a description of the possible values.

**Value** The value for the property represented by using `ARValueStruct` (see page 52).

The instance display properties are represented as a series of `ARDisplayInstanceStruct` structures, each of which also consists of the following elements:

**VUI ID** The internal ID associated with the view.

**Display Properties** The field display properties specific to that view represented by using `ARPropList`.

Each ARDi spl ayl nstanceStruct represents a **display instance** for the field. Specifying a view in the ARDi spl ayl nstanceLi st structure includes the field in the view, even if you do not define any display properties specific to that view. In this case, the system uses properties defined in the common properties list.

---

**Note:** The ARPropLi st structure is also used by the ARCreateVUI , ARGetVUI , and ARSetVUI functions (see Chapter 4, “AR System C API calls”). The dPropLi st parameter is a pointer to a structure of this type and is used to pass view-type display properties. This data structure is also used for handling object properties (see page 77 for more information).

---

## Using ARCoordList to specify field size and location

The ARCoordLi st data type is used to specify the size and location of certain fields and their components. These values are scaled by the AR System clients so that fields display consistently across different user environments.

### Bounding boxes and coordinate lists

The AR\_DPROP\_\*\_BBOX and AR\_DPROP\_COORDS display properties are all ARCoordLi st structures.

The AR\_DPROP\_\*\_BBOX properties identify the **bounding box**, or screen boundaries, for screen elements. AR\_DPROP\_COORDS identifies the screen location for lines and boxes.

When specifying coordinate values for either of these properties, remember that:

- The order in which you specify coordinate pairs is important.
- Horizontal and vertical are the only line orientations currently supported by BMC Remedy Administrator and BMC Remedy User. Neither of the clients display diagonal lines for lines or boxes if you specify coordinates that are not on the same x- or y-axis.

## Coordinate value scaling

To minimize display differences across a variety of operating systems and screen resolutions, the clients normalize all coordinate values before storing them in the database and then map these values to pixels in the current environment upon retrieval. You must apply the same translation algorithms when specifying or retrieving coordinate data in order for BMC Remedy Administrator and BMC Remedy User to display the field correctly.

Use this algorithm to normalize coordinate values before storing them:

$$\text{Normalized} = (\text{Pixel} * \text{AR_FIXED_PIXEL_PRECISION} * \text{Static Font Metric}) / (\text{Dynamic Font Metric} * \text{Platform Scale})$$

Use this algorithm to map coordinate values to pixels upon retrieval:

$$\text{Pixel} = ((\text{Normalized} * \text{Dynamic Font Metric} / \text{Static Font Metric}) / \text{Platform Scale}) + (\text{AR_FIXED_PIXEL_PRECISION} / 2) / \text{AR_FIXED_PIXEL_PRECISION}$$

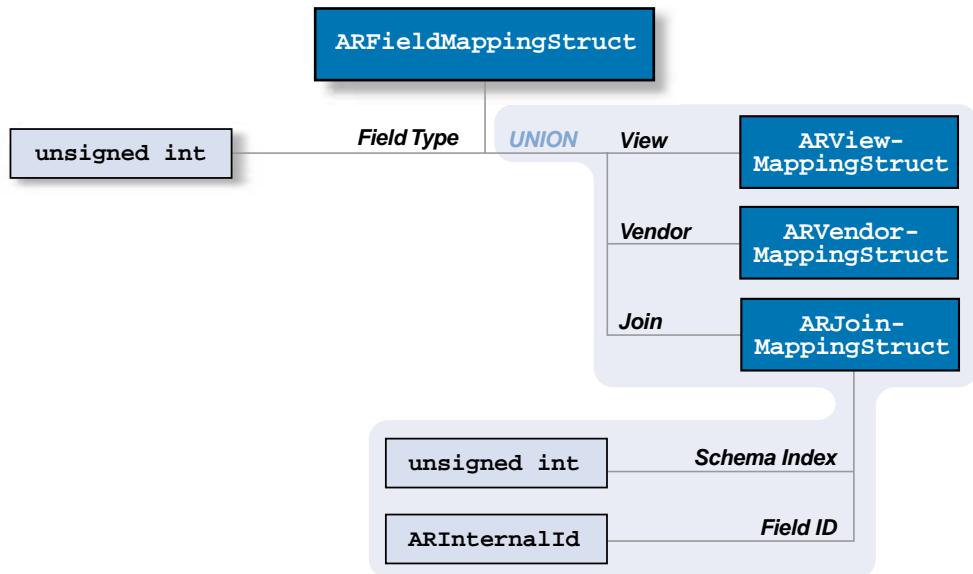
The value for AR\_FIXED\_PIXEL\_PRECISION is 100 on all platforms. The other equation values are platform-specific. The following table contains the Windows values.

Equation component	x-value	y-value
Static Font Metric	9	13
Dynamic Font Metric (using System font style)	(Average character width + maximum character width) / 2	Font height
Platform Scale	1.0	1.0

## Mapping fields in schemas

Because schemas (including join, view, and external forms) are logical objects instead of actual data tables, you must map each field in a schema to a field in an underlying base schema. This mapping is specified by the `field dMap` parameter, which is defined as a pointer to an `ARFieldMappingStruct` structure, as the following figure shows.

Figure 3-14: Structures used to map join fields to schema fields



The `ARFieldMappingStruct` structure consists of the following elements:

1	<code>AR_FIELD_TYPE_REGULAR</code>	Field in a base schema.
2	<code>AR_FIELD_TYPE_JOIN</code>	Field in a join schema.
3	<code>AR_FIELD_TYPE_VIEW</code>	Field in a view schema.
4	<code>AR_FIELD_TYPE_VENDOR</code>	Field in a vendor schema.

<b>Field Type</b>	An integer value indicating the type of field. The following table describes the possible values.
<b>Join Mapping</b>	A union that defines the field mapping. Join fields, the only union member supported at this time, are defined by an <code>ARJoinMappingStruct</code> structure containing the index of the base schema and the ID of the field to map to this field.
<b>View Definition</b>	An integer value indicating the type of view. The following table describes the possible values.
<b>Vendor Definition</b>	An integer value indicating the type of vendor form. The following table describes the possible values.

The ARJoinMappingStruct structure also contains the following components:

0	AR_FIELD_MAPPING_PRIMARY	Primary schema in a join.
1	AR_FIELD_MAPPING_SECONDARY	Secondary schema in a join.

**Schema Index** An integer value identifying the member schema the field maps to. The following table describes the possible values.

**Field ID** The internal ID associated with the field.

If the member schema is *also* a join schema, you must create fields in all nested join schemas until you can map the field to an underlying base schema.

## Entries

The most common operations in the AR System involve creating, retrieving, or updating entries. The following section describes some of the structures used to perform these tasks.

### Retrieving entry lists

The first group of structures relate to retrieving a list of entries. You can retrieve entry lists in two ways: with the fields of each entry as one concatenated string (see “ARGetListEntry” on page 246), or as field/value pairs (see “ARGetListEntryWithFields” on page 252). In both cases, the function performs the query specified by the `qualifier` parameter (of type `ARQualifierStruct`) and returns the list of entries that match the search criteria.

---

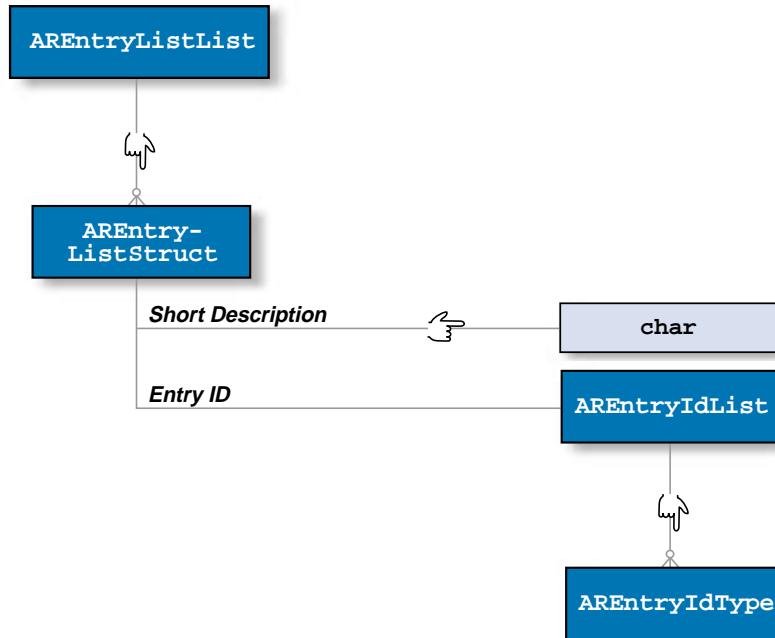
**Note:** The system identifies entries in join schemas by concatenating the entry IDs from the member schemas. As a result, an entry ID can consist of one or more values of type `AREntryIDType` and, therefore, is represented by using the `AREntryIDList` structure.

---

## Entries returned as concatenated strings

The ARGetListEntry function returns an entryList output parameter that is a pointer to an AREEntryListList structure, illustrated in the following figure.

Figure 3-15: Structures used to represent a list of entries as concatenated strings



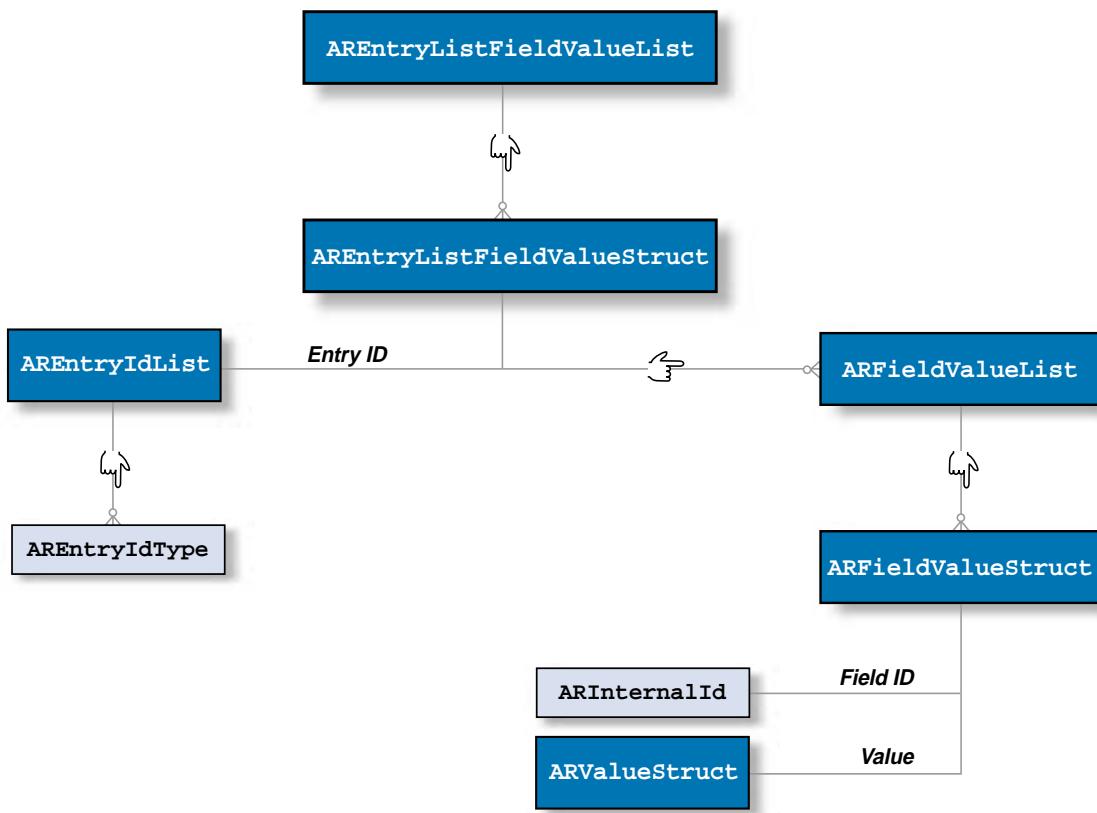
AREEntryList consists of zero or more AREEntryListStruct items, each of which represents a matching entry. The AREEntryListStruct structure consists of the following elements:

- |   |  |
|---|--|
| <b>Entry ID</b><br><b>Short Description</b> | The ID associated with the entry.<br>A 128-character string containing data concatenated from the fields specified by the getListFields parameter. |
|---|--|

## Entries returned as Field/Value pairs

The ARGetListEntryWithFields function returns an entryList output parameter that is a pointer to an AREEntryListFieldsValueList structure, as the following graphic shows.

Figure 3-16: Structures used to represent a list of entries as Field/Value pairs



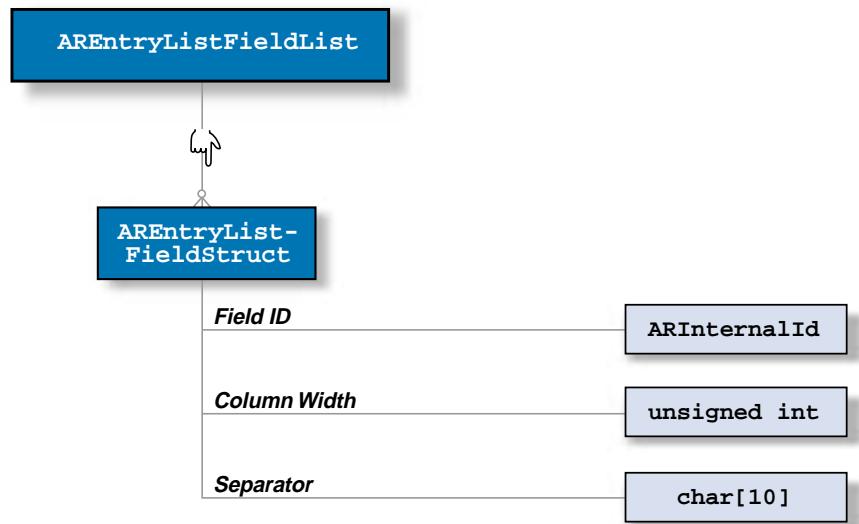
**AREntryListFieldValueList** consists of zero or more **AREntryListFieldValueStruct** items, each of which represents a matching entry. The **AREntryListFieldValueStruct** structure consists of an **Entry ID** and a list of **ARFieldValueStruct** structures, each of which represents a field in the entry. An **ARFieldValueStruct** structure consists of the following elements:

- |                 |  |
|-----------------|--|
| <b>Field ID</b> | The internal ID associated with the field.                                       |
| <b>Value</b>    | The value for the field represented by using <b>ARValueStruct</b> (see page 52). |

## Specifying fields to retrieve

The `ARGetListFields` parameter of both `ARGetListEntry` and `ARGetListEntryWithFields` identifies the fields to return with each entry. It also defines the column width and column separator for each field, which are used by `ARGetListEntry` but ignored by `ARGetListEntryWithFields`. This parameter is a pointer to an `AREntryListFieldList` structure, as this figure shows.

Figure 3-17: Structures used to define column fields in an entry list



Each `AREntryListFieldStruct` item specifies a particular field and associated display information:

<b>Field ID</b>	The internal ID associated with the field.
<b>Column Width</b>	An integer value indicating the number of characters to display for the field ( <code>ARGetListEntry</code> only). For <code>ARGetListEntryWithFields</code> , set this value to a number greater than 0.
<b>Separator</b>	The characters to display after the field data ( <code>ARGetListEntry</code> only). The separator can contain from zero to 10 characters. For <code>ARGetListEntryWithFields</code> , set this value to one blank space.

---

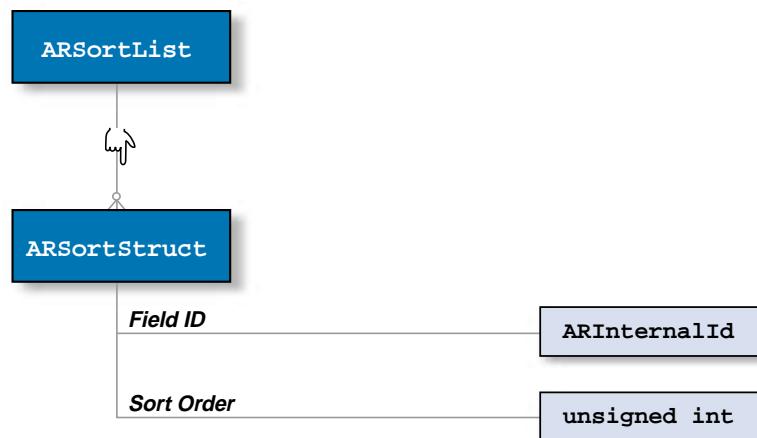
**Note:** For ARGetListEntry, the combined length of all specified fields, including separator characters, can be up to 128 bytes (limited by AR\_MAX\_SDESC\_SIZE).

---

## Sorting a list of entries

The sortList parameter of both ARGetListEntry and ARGetListEntryWithFields specifies the sort order for a list of entries. This parameter is a pointer to an ARSortList structure.

Figure 3-18: Structures used to define sort fields in an entry list



As this figure shows, ARSortList contains zero or more ARSortStruct items. Each ARSortStruct consists of the following elements:

<b>Field ID</b>	The internal ID associated with the field.
<b>Sort Order</b>	An integer value indicating the sort order for the field. The following table describes the possible values.

---

1	AR_SORT_ASCENDING	Sort values in field in ascending order.
2	AR_SORT_DESCENDING	Sort values in field in descending order.

---

The overall order of entries in the list is determined by the value you specify for each field and the order of ARSortStruct items in the ARSortList structure.

## Manipulating individual entries

The ARGetListEntry and ARGetListEntryWithFields functions (see Chapter 4, “AR System C API calls”) return a list of entries that match specified criteria. This list contains the Entry IDs for the matching entries but does not contain all field values. To retrieve or modify individual entries, you must pass the entry ID as an input argument to the ARGetEntry or ARSetEntry functions (see Chapter 4, “AR System C API calls”).

Both of these functions, as well as the ARCreateEntry and ARGetListEntryWithFields functions, use the ARFieldValuelist structure to pass field values for individual entries. The ARFieldValuelist structure in Figure 3-16 on page 72 consists of zero or more ARFieldValuelist items. Each of these structures contains a field and its value (a field/value pair):

<b>Field ID</b>	The internal ID associated with the field.
<b>Value</b>	The value for the field represented by using ARValueStruct (see page 52). This value must be of the same data type as the field or have the data type set to zero (AR_DATA_TYPE_NULL).

All entry data, whether being submitted, retrieved, or modified, is represented by using this structure.

The following is a code fragment that loads data into this structure.

```
#include "arstruct.h" /* symbolic constants for core field IDs*/
int main( int argc, char *argv[] )
{
    ARFieldValuelist*fldList; /* structure to populate */
    ARFieldValuelist*fldStrcp; /* working pointer*/
    ...
    ****
    /*Populate field list with data */
    ****
    fldList.numItems = 3;
    fldList.fldValuelist=malloc(3*sizeof(ARFieldValuelist));
    fldStrcp = fldList.fldValuelist;
    if( fldStrcp == NULL){
        fprintf(stderr, "Out of memory, malloc failure\n");
        exit(1);
    }
}
```

```
fI dStrcp[0]. fi el dl d = AR_CORE_SUBMITTER;
fI dStrcp[0]. val ue. dataType = AR_DATA_TYPE_CHAR;
fI dStrcp[0]. val ue. u. charVal = "Bob";

fI dStrcp[1]. fi el dl d = AR_CORE_STATUS;
fI dStrcp[1]. val ue. dataType = AR_DATA_TYPE_ENUM;
fI dStrcp[1]. val ue. u. enumVal = STATUS_NEW;

fI dStrcp[2]. fi el dl d = AR_CORE_SHORT_DESCRIPTION;
fI dStrcp[2]. val ue. dataType = AR_DATA_TYPE_CHAR;
fI dStrcp[2]. val ue. u. charVal = "Broken computer";
...
}
}
```

---

**Note:** In this example, the `charVal` members for the Submitter and Short Description fields are on the stack. As a result, you cannot use the `FreeARFi el dVal ueLi st` function to free this structure, because the FreeAR routines assume that all structure components are in allocated memory (on the heap). For more information, see “Freeing allocated memory” on page 112.

---

## Retrieving multiple entries

The `ARGetMultipleEntries` function returns a list of entries with the specified Entry IDs. You can retrieve data for specific fields or all (accessible) fields. `ARGetMultipleEntries` performs the same action as `ARGetEntry` but is easier and more efficient than retrieving multiple entries one by one.

This function uses the `ARFi el dVal ueLi stLi st` structure to return field values for the specified entries. This structure consists of zero or more `ARFi el dVal ueLi st` structures, as Figure 3-16 on page 72 shows.

`ARGetMultipleEntries` uses the `AREntryList` structure to pass Entry IDs for the specified entries. This structure consists of zero or more `AREntryList` structures, as Figure 3-15 on page 71 shows. This call can request a maximum of 100 entries, but can be used multiple times.

You can also use the `ARGetMultipleEntries` function to check whether the entries in the `AREntryList` exist without transferring any field data, conserving network traffic. The function has an `existList` parameter that returns a pointer to an `ARBool` list structure. Each `ARBool` flag in that list specifies whether a given Entry ID exists in the database.

# Filters, escalations, and active links

The following section describes some of the structure types related to creating, retrieving, and modifying filters, escalations, and active links definitions.

## Server object properties

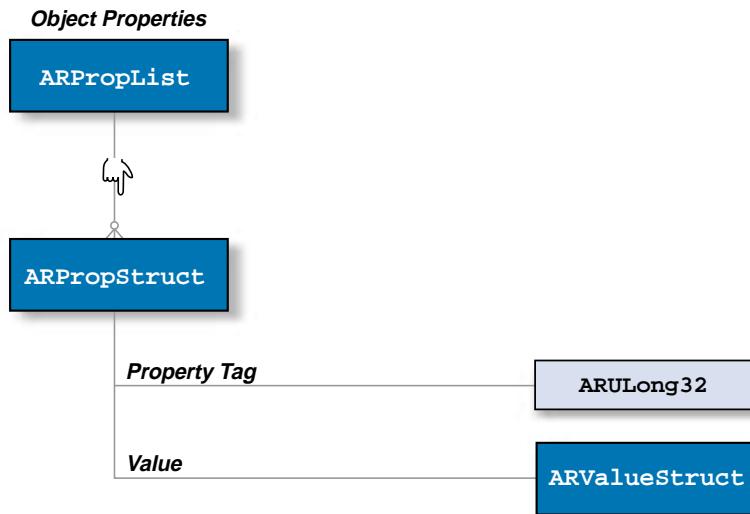
The ARCreate<*object*>, ARGet<*object*>, and ARSet<*object*> functions for server objects such as filters, escalations, and active links all have an obj PropList parameter that defines the set of one or more properties associated with that object. Server objects are any object on the server that can be discretely exported, such that an individual object history can be tracked. The complete list of server objects comprises:

- forms
- escalations
- character menus
- distributed mappings
- filters
- active links
- containers

See the *Form and Application Objects* and *Workflow Objects* guides for more information about source code control system support.

The C API uses the ARPropStruct data structure to store object properties associated with an Source Code Control (SCC) integration. This data structure is also used for the display properties of objects such as fields.

Figure 3-19: ARPropList data structure for object properties



In AR System, server objects have a list of properties, and there are property tags you use to describe the properties of objects. If the obj PropLi st parameter for the ARCreate<*object*>, ARGet<*object*>, and ARSet<*object*> functions for server objects is `NULL`, an object properties list with zero properties will be associated with the object.

Every server object has a list of tag-value pairs associated with it. The ARPropLi st is used to store these. The ARPropStruct contains the tag and an ARVal ueStruct, which is used to store the values for a given tag.

---

**Note:** Object property tags with a value of `AR_OPROP_LAST_RESERVED` or below are reserved. The API programmer might define any tag with a value above `AR_OPROP_LAST_RESERVED`.

---

## Source code control system integration support

You can maintain several versions of server objects by using an external SCC system that can be integrated with the AR System. An SCC system records all changes made to your source files so that you can recover exactly any revision, and recreate any released version, that you need.

## Server object property tags

You can specify the following properties in any order. You can define a property list that contains multiple instances of the same tag; however, API programs might choose to ignore duplicate instances of properties.

The AR System server has the following object property tags:

AR\_OPROP\_VENDOR\_NAME: (CHAR):

Indicates the vendor name, for example, “Arrow Systems, Inc.”

AR\_OPROP\_VENDOR\_PRODUCT (CHAR):

Indicates the product name, for example, “AR System.”

AR\_OPROP\_VENDOR\_VERSION: (CHAR):

Indicates the vendor version, for example, “6.3.”

AR\_OPROP\_GUID: (CHAR):

Indicates the globally unique identifier for the object.

AR\_OPROP\_COPYRIGHT: (CHAR):

Indicates the copyright string.

AR\_OPROP\_SCC\_LOCKED\_BY: (CHAR):

Indicates the name of user who has locked the object.

AR\_OPROP\_SCC\_VERSION (CHAR):

Indicates the version of the SCC system.

AR\_OPROP\_SCC\_TIMESTAMP: (TIME):

Indicates the SCC timestamp.

AR\_OPROP\_SCC\_USER: (CHAR):

Indicates the SCC username.

AR\_OPROP\_SCC\_LOCATION: (CHAR):

Indicates the location within the SCC system where the object might be found.

AR\_OPROP\_SCC\_DATA\_LOCKED\_BY: (CHAR):

Indicates the name of the user who locked the data, or entries, stored in a form under revision control. The AR System server does not enforce this lock and thus it is to be considered an advisory lock only.

AR\_OPROP\_SCC\_DATA\_VERSION: (CHAR):

Indicates the SCC version of the data, or entries, stored in a form under revision control.

AR\_OPROP\_SCC\_DATA\_TIMESTAMP: (CHAR):

Indicates the Source Code Control (SCC) timestamp of the data, or entries, stored in a form under revision control.

AR\_OPROP\_SCC\_DATA\_USER: (TIME):

Indicates the SCC user name for the data, or entries, stored in a form under revision control.

AR\_OPROP\_SCC\_DATA\_LOCATION: (CHAR):

Indicates the location within the SCC system where the data, or entries, stored in a form under revision control.

## Server managed object property tags

The AR System server has the following object properties for AR System that are managed exclusively by the server.

AR\_SMOPROP\_APP\_ACCESS\_POINT

Indicates whether the object is an access point. Zero (0) denotes that the object is not an access point. One (1) denotes that the object is an access point.

AR\_SMOPROP\_APP\_BSM\_TAG

For internal use only.

AR\_SMOPROP\_APP\_LIC\_DESCRIPTOR

Indicates the license descriptor for applications and forms.

AR\_SMOPROP\_APP\_LIC\_USER\_LICENSABLE

A numeric Boolean value that indicates whether the form is user-licensable.

AR\_SMOPROP\_APP\_LIC\_VERSION

Indicates the license version for applications and forms.

AR\_SMOPROP\_APP\_OWNER

The name of the application owner for the object.

---

AR\_SMOPROP\_APP\_STAT\_FORM

Participates in application statistics.

AR\_SMOPROP\_ENTRYPOI\_NT\_DEFAULT\_NEW\_ORDER

Entry point order within entry point list.

AR\_SMOPROP\_ENTRYPOI\_NT\_DEFAULT\_SEARCH\_ORDER

Entry point order within entry point list.

AR\_SMOPROP\_OBJECT\_LOCK\_KEY

Object lock key. This is a string that used as a key (or a password) to enforce the locking.

AR\_SMOPROP\_OBJECT\_LOCK\_TYPE

Object lock type. Valid values for the lock type are:

- 
- 0 Export the objects as unlocked (AR\_LOCK\_TYPE\_NONE).
  - 1 Export the objects with read-only lock. When these objects are imported, they are readable but not modifiable (AR\_LOCK\_TYPE\_READONLY).
  - 2 Export the objects with hidden-lock. When these objects are imported, they are neither readable nor modifiable (AR\_LOCK\_TYPE\_HIDDEN).
- 

AR\_SMOPROP\_PRIMARY\_FIELDSET

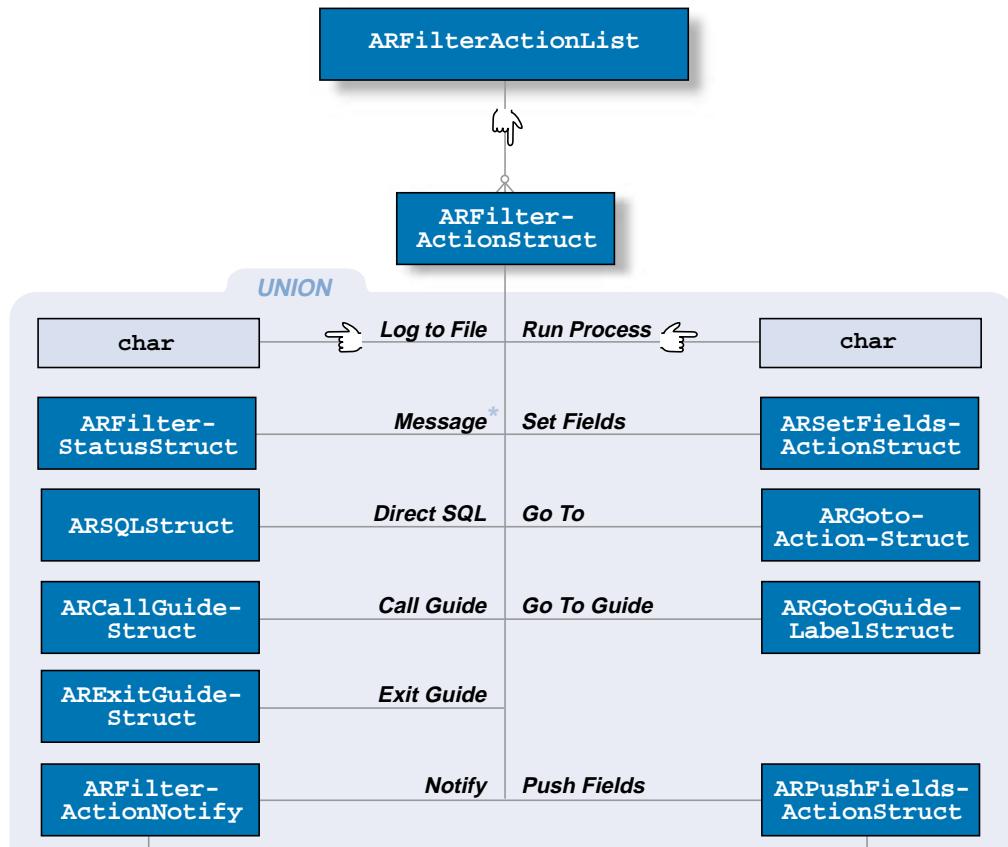
A set of fields used as a unique index while importing application-specific data. This property only applies if the form includes fields 61000 and 61001.

## Server object actions

The ARCreate<*object*>, ARGet<*object*>, and ARSet<*object*> functions for filters, escalations, and active links all have an `actionList` parameter that defines the set of one or more actions performed for entries that match the associated qualification. Because the actions available for filters and escalations are different than those for active links, the actions associated with these objects are represented by using different data structures.

For filters and escalations, the `actionList` parameter is a pointer to an `ARFilterActionList` structure, as the following figure shows.

Figure 3-20: Structures used to define filter or escalation actions



ARFilterActionList contains zero or more ARFilterActionStruct items. Each ARFilterActionStruct represents one filter or escalation action and consists of the following elements:

**Action Type:** An integer value indicating the type of action. The following table describes the possible values.

**Action Definition:** The specific action to perform (represented by using data types or structures appropriate to the type of action).

1	AR_FILTER_ACTION_NOTIFY	Send a notification to the specified user.
2	AR_FILTER_ACTION_MESSAGE	Return the specified message (filters only).

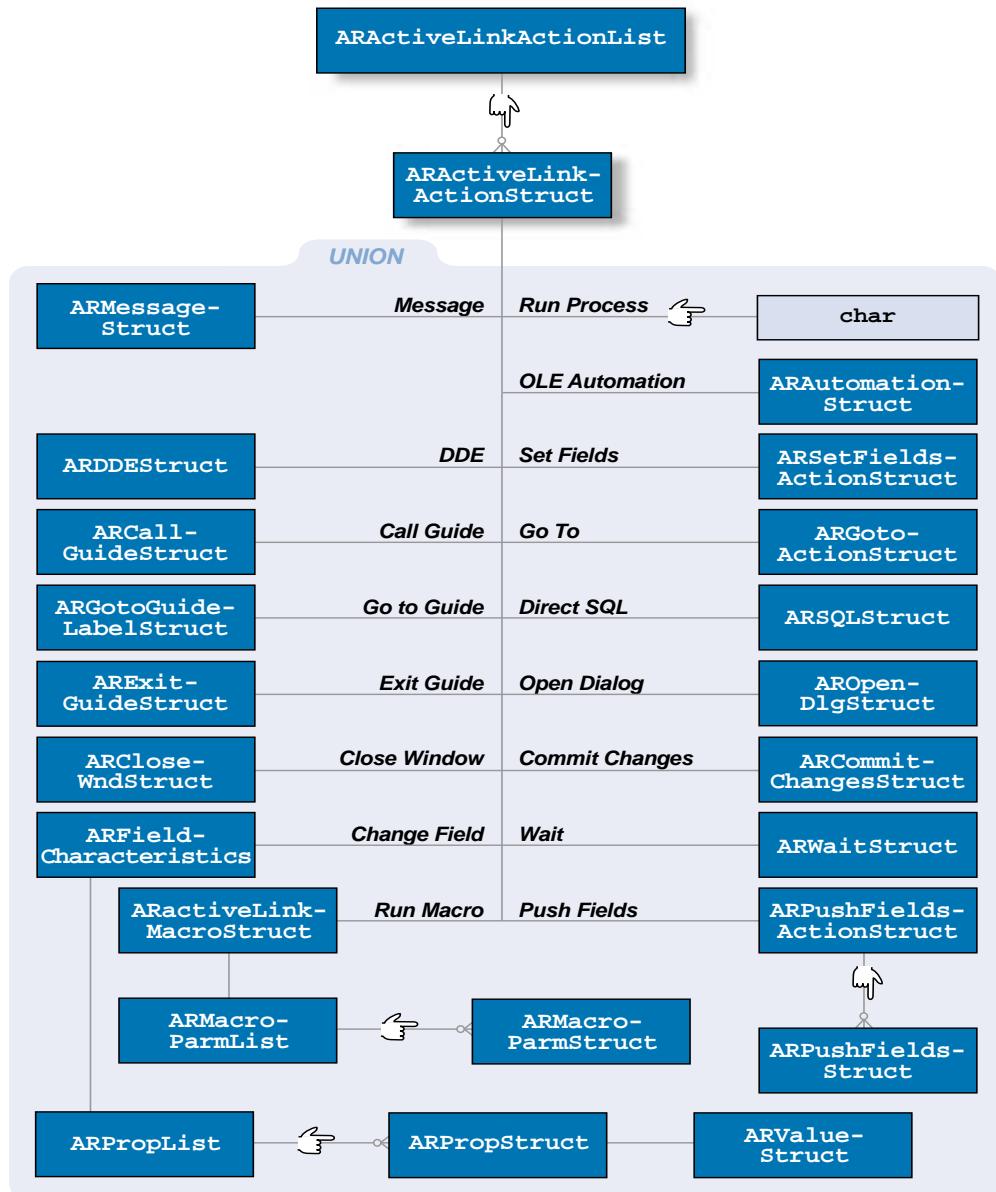
---

3	AR_FILER_ACTI ON_LOG	Write the specified text to a log file.
4	AR_FILER_ACTI ON_FIELDS	Update values for the specified fields.
5	AR_FILER_ACTI ON_PROCESS	Execute the specified command.
6	AR_FILER_ACTI ON_FILEDP	Push data from the current operation to a field in another schema on the same server.
7	AR_FILER_ACTI ON_SQL	Perform the specified SQL statements.
8	AR_FILER_ACTI ON_GOTOACTI ON	Go to the filter with the specified execution order.
9	AR_FILER_ACTI ON_CALLGUIDE	Execute the call guide action.
10	AR_FILER_ACTI ON_EXITGUIDE	Execute the exit guide action.
11	AR_FILER_ACTI ON_GOTOGUIDELABEL	Go to the filter with the specified guide label.

---

For active links, the `actionList` parameter is a pointer to an `ARActiveLinkActionList` structure, as the following figure shows.

Figure 3-21: Structures used to define active link actions



The ARActi veLi nkActi onLi st structure contains zero or more ARActi veLi nkActi onStruct items. Like ARFi l terActi onStruct, each ARActi veLi nkActi onStruct represents one active link action and has the following elements:

**Action Type:** An integer value indicating the type of action. The following table describes the possible values.

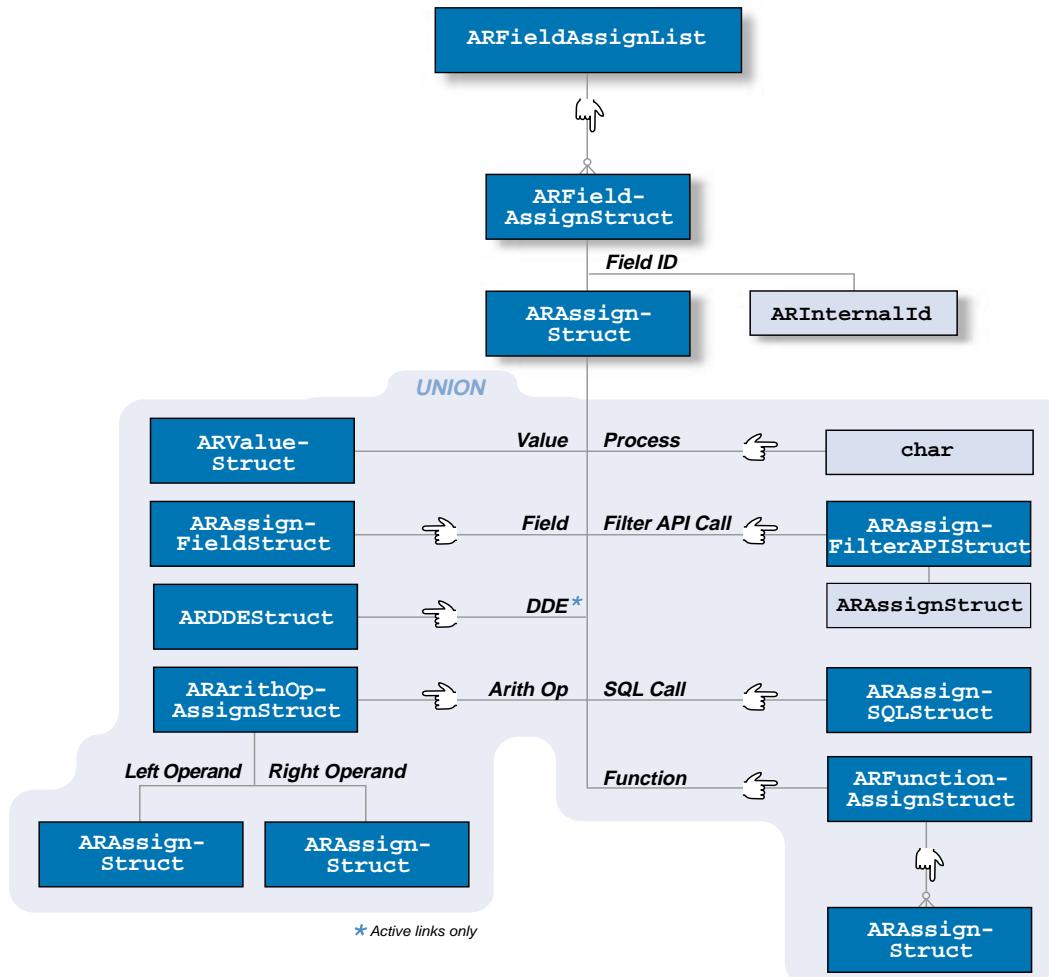
**Action Definition:** The specific action to perform (represented by using data types or structures appropriate to the type of action).

1	AR_ACTI VE_LI NK_ACTI ON_MACRO	Execute the specified macro.
2	AR_ACTI VE_LI NK_ACTI ON_FIELDS	Set values for the specified fields.
3	AR_ACTI VE_LI NK_ACTI ON_PROCESS	Execute the specified command.
4	AR_ACTI VE_LI NK_ACTI ON_MESSAGE	Display the specified message.
5	AR_ACTI VE_LI NK_ACTI ON_SET_CHAR	Set characteristics for the specified field.
6	AR_ACTI VE_LI NK_ACTI ON_DDE	Perform the specified DDE action.
7	AR_ACTI VE_LI NK_ACTI ON_FIELDP	Transfer, or <i>push</i> , data from a field in the current schema to another field in another schema.
8	AR_ACTI VE_LI NK_ACTI ON_SQL	Perform the specified SQL statements.
9	AR_ACTI VE_LI NK_ACTI ON_AUTO	Perform the specified OLE automation.
10	AR_ACTI VE_LI NK_ACTI ON_OPENDLG	Open a schema as a modal dialog.
11	AR_ACTI VE_LI NK_ACTI ON_COMMITC	Commit changes from a dialog to the parent schema window.
12	AR_ACTI VE_LI NK_ACTI ON_CLOSEWND	Close a dialog or schema window.
13	AR_ACTI VE_LI NK_ACTI ON_CALLGUI DE	Perform the specified guide.
14	AR_ACTI VE_LI NK_ACTI ON_EXITGUIDE	Exit the current guide or all guides.
15	AR_ACTI VE_LI NK_ACTI ON_GOTOGUI DELABEL	Go to the specified active link within current guide.
16	AR_ACTI VE_LI NK_ACTI ON_WAIT	Cause the current guide to wait for user input.
17	AR_ACTI VE_LI NK_ACTI ON_GOTOACTI ON	Go to the active link with the specified execution order.

## Set fields action

The Set Fields action is one of the most complex of the filter, escalation, or active link actions. The ARFieldAssignList structure in the following figure is a list of ARFieldAssignStruct items, each of which identifies a field/value pair to update. Each ARFieldAssignStruct structure contains the internal ID of the field to update and the value to assign, specified by using ARAssignStruct, which is one of the more complex structures in the API.

Figure 3-22: Structures used to define set fields action



The ARAssi gnStruct structure contains the following elements:

- Assign Type:** An integer value indicating the type of value to assign. The following table describes the possible values.
- Value:** The actual value to assign (represented by using data types or structures appropriate to the type of value).

---

1 AR_ASSI GN_TYPE_VALUE	A constant or keyword value represented by using ARVal ueStruct (see page 52). The data type of the value must match the data type of the specified field.
2 AR_ASSI GN_TYPE_FIELD	A schema field value represented by using ARAssi gnFi el dStruct (see “Assigning a schema field value” on page 87).
3 AR_ASSI GN_TYPE_PROCESS	An output value from successful execution of an operating system process or command. The system generates an error if the process return code is not zero. This option is not available for active links on Windows.
4 AR_ASSI GN_TYPE_ARITH	A result value from an arithmetic operation represented by using ARArithOpAssi gnStruct (see “Assigning an arithmetic result value” on page 90).
5 AR_ASSI GN_TYPE_FUNCTION	A return value from a function represented by using ARFunctionAssi gnStruct (see “Assigning a function return value” on page 90).
6 AR_ASSI GN_TYPE_DDE	A result value from a DDE request to another application represented by using ARDDEStruct (see “Assigning a DDE result value” on page 93). This option is available for active links on Windows clients only.
7 AR_ASSI GN_TYPE_SQL	A result value from an SQL command represented by using ARAssi gnFi lterAPI Struct (see “Assigning an SQL result value” on page 93).
8 AR_ASSI GN_TYPE_FILTER_API	A result value from a Filter API command represented by using ARAssi gnFi lterApi Struct (see “Assigning a filter API result value” on page 94).

---

## Assigning a schema field value

The ARAssi gnStruct structure contains an ARAssi gnFi el dStruct structure that identifies a schema field value to assign in a Set Fi el ds or Push Fi el ds action. You can specify a value from any entry in any schema on a particular server.

In a Push Fields action, an independent ARAssignFieldStruct structure identifies the schema field that is set by the operation. You can specify a field from any entry in any schema on a particular server. To set or push all matching field values from one schema to the other, use AR\_LINK\_ID in the field.

The ARAssignFieldStruct structure consists of the following elements:

<b>Server</b>	A string specifying the name of the server where the schema is located. The ARGetListServer function retrieves a list of available servers (see Chapter 4, “AR System C API calls”). For filters and escalations, specify the server where the filter or escalation is located (either by name or by passing @). For active links, specify * to retrieve the value from the current window.
<b>Schema</b>	The name of the schema containing the field value to assign. For filters and escalations, specify @ to retrieve or set the value from the current transaction. For active links, specify * to retrieve or set the value from the current window.
<b>Search Criteria</b>	A qualification that identifies the entries to retrieve or set (optional). Specify the search criteria by using ARQualifierStruct (see Figure 3-10 on page 56).
<b>Tag</b>	An integer value indicating the type of field value to retrieve or set. The following table describes the possible values.
<b>Field ID/Stat History/Currency</b>	The field containing the value to assign or set. Specify the field by using the field ID, the ARStatisticValue structure, or the ARCurrencyPartStruct structure. Use the AR_LINK_ID field identification to set or push matching field values between two schemas. See the <i>Workflow Objects</i> guide for more information.
<b>No Match Option</b>	An integer value indicating the action to take if no entries match the search criteria. The following table describes the possible values.
<b>Multiple Match Option</b>	An integer value indicating the action to take if multiple entries match the search criteria (Set Fields) or if any entry matches the search criteria (Push Fields). The following table describes the possible values.

## Tag values for ARAssignFieldStruct

1	AR_FI_ELD	Value from a regular schema field.
4	AR_STAT_HI_STORY	Value from the Status History core field.
6	AR_CURRENCY_FLD	Value from a currency field.

## No match options for ARAssignFieldStruct

1	AR_NO_MATCH_ERROR	Return an error.
2	AR_NO_MATCH_SET_NULL	Assign NULL (Set Fields <b>action only</b> ).
3	AR_NO_MATCH_NO_ACTION	Do nothing (Push Fields <b>action only</b> ).
4	AR_NO_MATCH_SUBMIT	Submit a new entry (Push Fields <b>action only</b> ).

## Multiple-Match options for ARAssignFieldStruct

1	AR_MULTI_MATCH_ERROR	Return an error
2	AR_MULTI_MATCH_SET_NULL	Assign NULL (Set Fields <b>action only</b> ).
3	AR_MULTI_MATCH_USE_FIRST	Assign a value from the first matching entry.
4	AR_MULTI_MATCH_PICKLIST	Display a selection list (Active Links only).
5	AR_MULTI_MATCH_MODIFY_ALL	Modify all matching entries (Push Fields <b>action only</b> ).
6	AR_MULTI_MATCH_NO_ACTION	Do nothing (Push Fields <b>action only</b> ).
7	AR_MULTI_MATCH_USE_LOCALE	Assign a value from the first matching entry based on the locale (Set Fields <b>action or Active Links only</b> ).

## Assigning an arithmetic result value

The ARArithOpAssignmentStruct structure defines an arithmetic result value to assign in a Set Fields action. This structure is very similar to the ARArithOpStruct structure and uses the same set of operation values (see “Defining an arithmetic result value” on page 59). Like ARArithOpStruct, ARArithOpAssignmentStruct consists of a tag identifying the type of arithmetic operation and two operands specifying the values. In this case, however, the operands are represented by using ARAssignmentStruct instead of ARFieldValuedOrArithStruct.

## Assigning a function return value

The ARFunctionAssignmentStruct structure specifies a function return value to assign in a Set Fields action. This structure consists of the following elements:

<b>Function Type</b>	An integer value indicating the type of function to perform. The following table describes the possible values.
<b>Number of Parameters</b>	An integer value identifying the number of function input parameters.
<b>Parameter List</b>	A pointer to the parameter values to use (represented in an ARAssignmentStruct structure).

For more information about the function type values, see the *Workflow Objects* guide.

### Function type values for ARFunctionAssignmentStruct

Function Type	Input	Return	Value
1 AR_FUNCTION_DATE	timestamp	char	Date
2 AR_FUNCTION_TIME	timestamp	char	Time
3 AR_FUNCTION_MONTH	timestamp	long	Month (1-12)
4 AR_FUNCTION_DAY	timestamp	long	Day (1-31)
5 AR_FUNCTION_YEAR	timestamp	long	Year
6 AR_FUNCTION_WEEKDAY	timestamp	long	Weekday (1-7)
7 AR_FUNCTION_HOUR	timestamp	long	Hour (0-23)
8 AR_FUNCTION_MINUTE	timestamp	long	Minute (0-59)
9 AR_FUNCTION_SECOND	timestamp	long	Second (0-59)

Function Type	Input	Return	Value
10 AR_FUNCTION_TRUNC	real	long	Truncated value
11 AR_FUNCTION_ROUND	real	long	Rounded value
13 AR_FUNCTION_LENGTH	char	long	Length of string in bytes
14 AR_FUNCTION_UPPER	char	char	Uppercase string
15 AR_FUNCTION_LOWER	char	char	Lowercase string
16 AR_FUNCTION_SUBSTR	char, long [, long] long]	char	Substring from long1 to long2 (inclusive) in bytes
17 AR_FUNCTION_LEFT	char, long	char	Left-most n bytes
18 AR_FUNCTION_RIGHT	char, long	char	Right-most n bytes
19 AR_FUNCTION_LTRIM	char	char	Leading blanks removed
20 AR_FUNCTION_RTRIM	char	char	Trailing blanks removed
21 AR_FUNCTION_LPAD	char, long, char	char	char1 left-padded with char2 to n bytes
22 AR_FUNCTION_RPAD	char, long, char	char	char1 right-padded with char2 to n bytes
23 AR_FUNCTION_REPLACE	char, char, char	char	char1 with char2 replaced by char3
24 AR_FUNCTION_STRSTR	char, char	int	Position of char2 in char1 in bytes (-1 = not found)
25 AR_FUNCTION_MIN	<x>, <x>, [, <x> <x>] ...		Minimum value
26 AR_FUNCTION_MAX	<x>, <x>, [, <x> <x>] ...		Maximum value
27 AR_FUNCTION_COLSUM			Sum a table column
28 AR_FUNCTION_COLCOUNT		int	Number of non-null values in a table column
29 AR_FUNCTION_COLAVG			Average of non-null values in a table column
30 AR_FUNCTION_COLMIN			Minimum of non-null values in a table column
31 AR_FUNCTION_COLMAX			Maximum of non-null values in a table column

Function Type	Input	Return	Value
32 AR_FUNCTION_DATEADD	char, int, date	date	Number of days, weeks, months, or years to add to date
33 AR_FUNCTION_DATEDIFF	char, date, date	int	Number of days or weeks between the start date and end date
34 AR_FUNCTION_DATENAME	char, date	char	The name of the day or month corresponding to date
35 AR_FUNCTION_DATENUM	char, date	int	Depending on the value of char (year, month, week, day, or weekday), returns the numeric value of the year, month (1 to 12), week (1 to 52), day (1 to 31) or weekday (1=Sunday, 2=Monday, and so on)
36 AR_FUNCTION_CURRENCYSYMBOL	currency, char, timestamp		Currency values to convert, based on timestamp
37 AR_FUNCTION_CURRENTPRECISION	currency, timestamp		Date of currency and functional currency values
38 AR_FUNCTION_CURRENCYPRECISION	currency, char		Type of currency and functional currency values
39 AR_FUNCTION_CURRENCYSYMBOL	currency, char		Value of currency and functional currency values
40 AR_FUNCTION_LENGTH	char	long	Length of string in characters
41 AR_FUNCTION_LEFT	char, long	char	Leftmost x characters
42 AR_FUNCTION_RIGHT	char, long	char	Rightmost x characters
43 AR_FUNCTION_LPAD	char, long, char	char	char1 left-padded with char2 to n characters
44 AR_FUNCTION_RPAD	char, long, char	char	char1 right-padded with char2 to n characters
45 AR_FUNCTION_STRPOS	char, char	int	Position of char2 in char1 in characters (-1 = not found)
46 AR_FUNCTION_SUBSTR	char, long [, long]	char	Substring from long1 to long2 (inclusive) in characters

Function Type	Input	Return	Value
47 AR_FUNCTION_ENCRYPT	(plain text, key)	ciphertext	Encrypted value of a text string (plain text), using the encryption key (key)
48 AR_FUNCTION_DECRYPT	(ciphertext, key)	plaintext	Unencrypted text value of the encrypted text (ciphertext), using the encryption key (key)

---

**Note:** If necessary, the system automatically converts all input parameters to the correct data type.

---

## Assigning a DDE result value

The ARDDEStruct structure defines a DDE result value to assign in a Set Fields action. This option is available for active links on Windows clients only. The ARDDEStruct structure consists of the following elements:

<b>Service Name</b>	A string specifying the service to use.
<b>Topic</b>	A string specifying the topic to use.
<b>Item</b>	A string specifying the item to retrieve.
<b>Action</b>	An integer value identifying the type of DDE action. Specify AR_DDE_REQUEST for this item.
<b>Path</b>	A string specifying the path to the application.
<b>Command</b>	A string specifying the command to execute. Specify NULL for this item.

## Assigning an SQL result value

The ARAssignSQLStruct structure specifies an SQL result value to assign in a Set Fields action. Use this option to assign a value from any SQL database. The ARAssignSQLStruct structure consists of the following elements:

<b>Server</b>	A string specifying the name of the server where the database is located. The ARGetListServer function retrieves a list of available servers (see Chapter 4, “AR System C API calls”).
<b>SQL Command</b>	A string specifying the SQL query to execute.
<b>Value Index</b>	An integer value indicating which of the selected columns contains the value to assign.

No Match Option	An integer value indicating the action to take if <i>no</i> rows match the selection criteria. The following table describes the possible values.
Multiple Match Option	An integer value indicating the action to take if <i>multiple</i> rows match the selection criteria. The following table describes the possible values.

## Assigning a filter API result value

The ARAssignFilterApiStruct structure specifies the name of the plug-in, the input values for the operation, and a pointer to the output values. The ARAssignFilterApiStruct structure consists of the following elements:

Service Name	A string specifying the name of the plug-in.
Input Values	A list of values that will be provided as input to the AR System filter (ARF) API plug-in.
Return Index	An index into the returned list of values that will be used in the set field assignment.

---

**Note:** You can assign ARF API result values for escalation set field actions, but you cannot assign ARF API result values for active link set field actions.

---

## Push fields action

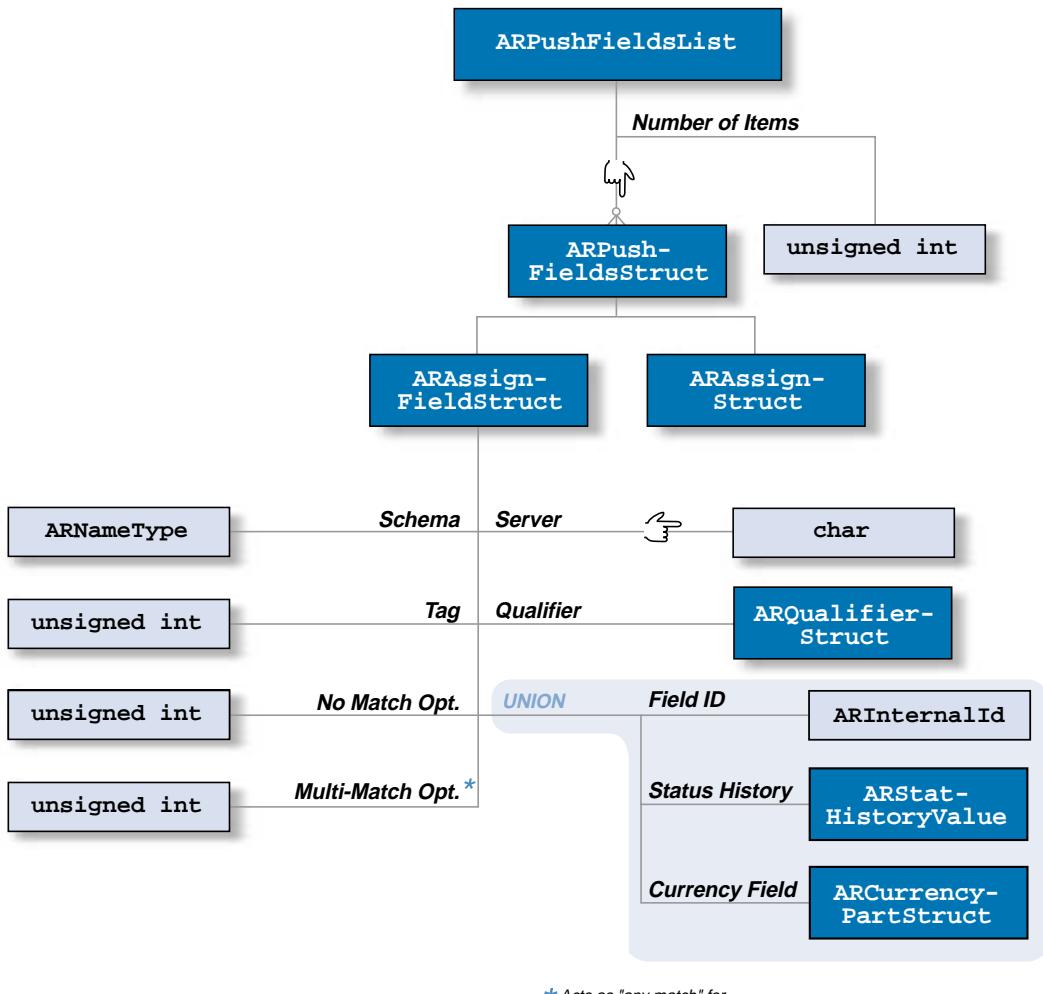
Where the Set Fields action sets the value of a field in the current schema, the Push Fields action modifies or submits entries in any schema. When Push Fields is used in an active link, the schema can be on any server. The ARPUSHFieldsList structure in Figure 3-23 on page 95 is a list of ARPUSHFields items. Each item contains an ARAssignFieldStruct structure specifying the field to update and an ARAssignStruct structure specifying the value to push there. These data structures are also used in Set Fields actions and are explained in that section of this guide.

---

**Note:** You can force a Push Fields action to perform a submit by setting the Qualifier to AR\_COND\_OP\_NONE, the No Match option to AR\_NO\_MATCH\_SUBMIT, and the Multiple Match option to AR\_MULTI\_MATCH\_NO\_ACTION.

---

Figure 3-23: Structures used to define push fields action

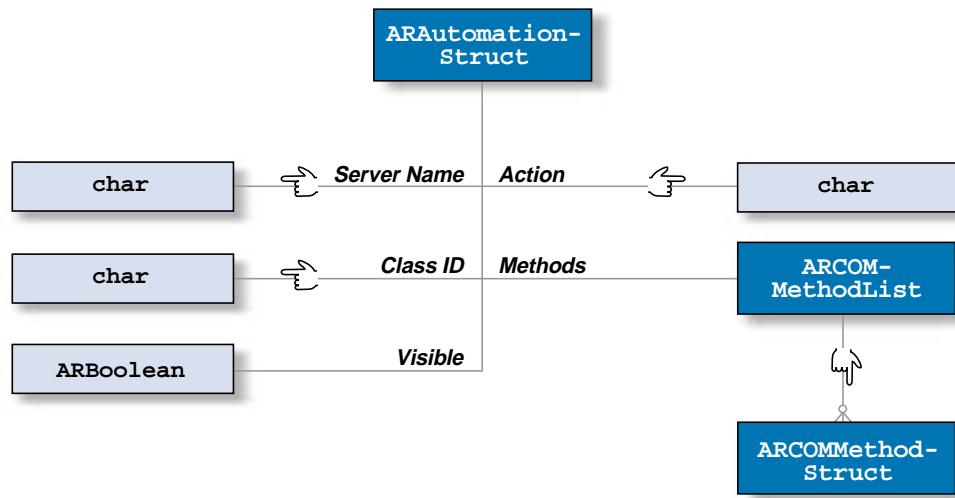


## Automation action

The Automation active link action enables a Windows client to control Automation servers on the same machine, accessing the data and methods that those servers have made available. For information about how to use the Automation action, see the *Workflow Objects* guide.

The data structure used to define an Automation action is the ARAutomationStruct structure, as the following figure shows.

Figure 3-24: Structures used to define automation action

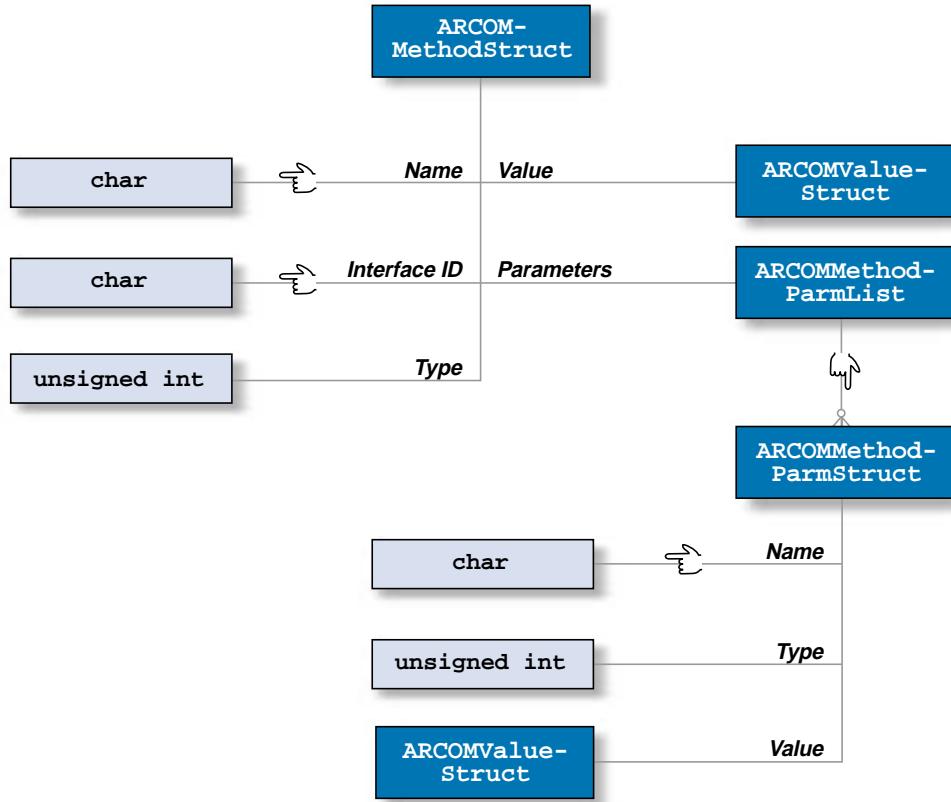


The ARAutomationStruct structure consists of the following elements:

<b>Server Name</b>	A string specifying the name of the Automation server, length limited by AR_MAX_COM_NAME (64 bytes).
<b>Class ID</b>	A string specifying the unique ID assigned to this server in the registry, length limited by AR_MAX_COM_ID_SIZE (128 bytes).
<b>Action</b>	A string specifying the equation defined by this action, including nested methods and the assignment if any. For example, \$Status\$ = MethodA(). MethodB(MethodC()). It is used for display only, and limited by AR_MAX_AUTOMATION_SIZE (2000 bytes).
<b>Visible</b>	Not used. Specify 0 (zero).
<b>Methods</b>	A list of the methods called by this active link action, specified by using an ARCOMMethodList structure. The methods must be listed in order of execution. Therefore, if Method B is passed as a parameter to Method A, Method B should be listed first so that its result value will be available when Method A is called.

Each method in an ARCOMMethodList structure uses the ARCOMMethodStruct structure, as the following figure shows.

Figure 3-25: Structures used to define OLE automation method



The ARCOMMetho-ParmStruct structure consists of the following elements:

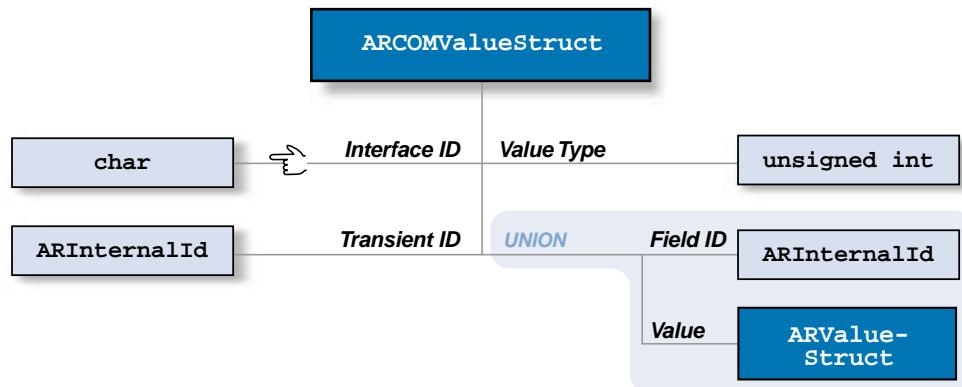
<b>Name</b>	The name of the method, length limited by AR_MAX_COM_METHOD_NAME (128 bytes).
<b>Interface ID</b>	The ID of the interface that owns the method, length limited by AR_MAX_COM_I_D_SI_ZE (128 bytes).
<b>Type</b>	The type of value returned by the method, if any. Possible values are all OLE variant types.
<b>Value</b>	The AR System field to set to the method's return value, if any. This is specified by using an ARCOMValueStruct structure, as Figure 3-26 on page 98 shows.
<b>Parameters</b>	A list of the parameters passed to this method, specified by using an ARCOMMetho-ParmList structure.

Each parameter in an ARCOMMethodParmList structure is specified by using the ARCOMMethodParmStruct structure in Figure 3-26 on page 98, which consists of the following elements:

<b>Name</b>	The name of the parameter.
<b>Type</b>	The data type of the parameter. Possible values are all OLE variant types.
<b>Value</b>	The value of the parameter, specified by using an ARCOMValueStruct structure.

The value returned by a method and the value passed in a parameter are each specified by using the ARCOMValueStruct structure, as the following figure shows.

Figure 3-26: Structures used to define OLE method and parameter values



The ARCOMValueStruct structure consists of the following elements:

<b>Interface ID</b>	When this ARCOMValueStruct structure is used to define the return value of a method, specify the ID of the interface returned by the method. When used to define the value of a parameter that is an OLE interface, specify the ID of that interface. Otherwise, specify NULL.
<b>Transient ID</b>	The Transient ID links a method that takes another method as a parameter to the method that is used as that parameter. For example, Method A takes Method B as a parameter. Use the same integer value here for the ARCOMValueStruct structure that defines the return value for Method B as you do for the structure that defines the parameter value of Method A.

---

<b>Value Type</b>	An integer specifying the type of value represented by the structure. The tables that follow describe the possible values. For Value Type 0, set the other members of the structure to NULL or 0.
<b>Field ID/Value</b>	A union that defines either the field ID or value, depending on the structure's type. For Value Type 1, this is the ID of the field to set to the return value of this method or the field to pass as this parameter, specified by using an ARI internal Id structure. For Value Type 2, this is the value to pass as this parameter, specified by using an ARValueStruct structure.

## Method return value types for ARCOMValueStruct

---

0	AR_COM_METHOD_NULL	No value returned for this method.
1	AR_COM_METHOD_FIELDID	Set the specified AR System field to the value returned by this method.

---

## Parameter value types for ARCOMValueStruct

---

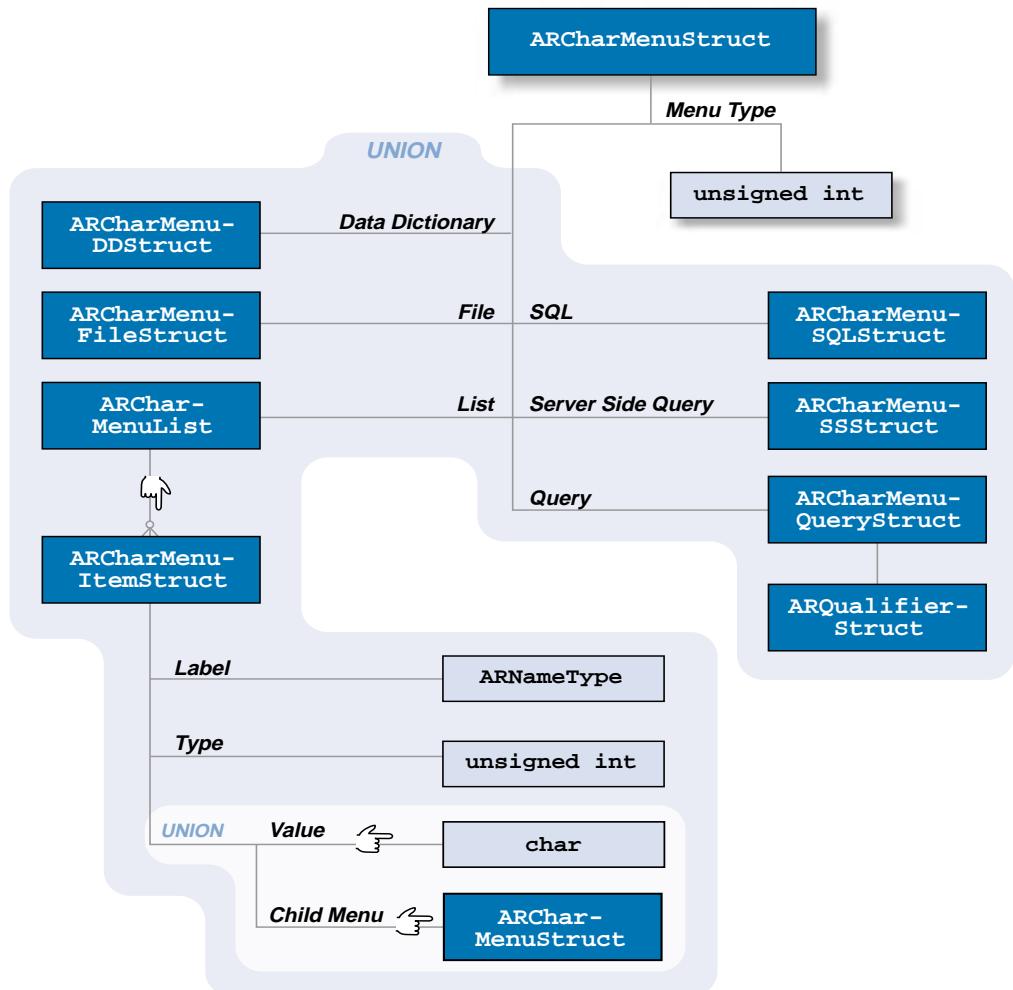
0	AR_COM_PARM_NULL	No value for this parameter.
1	AR_COM_PARM_FIELDID	Pass the specified AR System field as the parameter.
2	AR_COM_PARM_VALUE	Pass the specified value as the parameter.

---

## Character menus

All character menu definitions in the AR System are represented by using the ARCharMenuStruc structure, illustrated in the following figure.

Figure 3-27: Structures used to define character menus



Most of the character menu functions have a `menuDefn` parameter that is a pointer to a structure of this type (for example, see “`ARCreateCharMenu`” on page 134 or “`ARExpandCharMenu`” on page 204).

The ARCharMenuStruct structure defines both the content and organization of the menu and consists of the following elements:

<b>Menu Type</b>	An integer value indicating the type of character menu. The following table describes the possible values.
<b>Menu Definition</b>	The specific menu content (represented by using structures appropriate to the type of menu).

1 AR_CHAR_MENU_LIST	Menu items based on definitions in ARCharMenuList.
2 AR_CHAR_MENU_QUERY	Menu items based on schema query defined in ARCharMenuQueryStruct.
3 AR_CHAR_MENU_FILE	Menu items based on formatted flat file defined in ARCharMenuFileStruct.
4 AR_CHAR_MENU_SQL	Menu items based on SQL query defined in ARCharMenuSQLStruct.
5 AR_CHAR_MENU_SS	Menu items based on server side query defined in ARCharMenuSSStruct.
6 AR_CHAR_MENU_DATA_DICTIONARY	Menu items based on data dictionary query defined in ARCharMenuDDStruct.

The ARCharMenuList structure consists of zero or more ARCharMenuItemStruct items. Each ARCharMenuItemStruct represents an individual menu item. In this context, a menu item can be a value (a low-level item) or another menu (a top- or intermediate-level item).

The ARCharMenuItemStruct structure consists of the following elements:

<b>Label</b>	The label that identifies the menu item.
<b>Type</b>	An integer value indicating the type of menu item. The following table describes the possible values.
<b>Menu Definition</b>	The value associated with the menu item. For leaf-level items, the definition is a string containing the item text. For top- or intermediate-level items, the definition is a pointer to a child menu (represented by using ARCharMenuStruct).

1 AR_MENU_TYPE_VALUE	Leaf-level menu item.
2 AR_MENU_TYPE_MENU	Top- or intermediate-level menu item.

# Containers

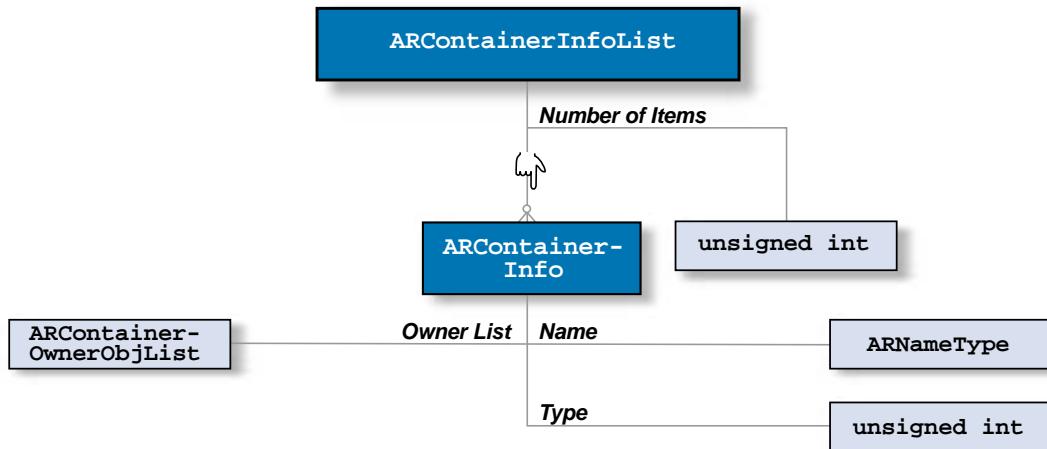
Containers are generic lists of references that are used to define guides and applications. For information about the purpose and use of guides and applications, see the *Workflow Objects* guide.

## Retrieving container lists

The first group of structures relate to retrieving a list of containers. You can retrieve entry lists by using the ARGetListContainer call. The function returns the list of containers that match the specified type, owning object, and modification time stamp.

The ARGetListContainer function uses the ARContainerInfoList structure to specify the containers it returns. The ARContainerInfoList structure consists of zero or more ARContainerInfo structures, as the following figure shows.

Figure 3-28: Structures used to define container lists



The ARContainerInfoList structure consists of the following elements:

- Name** An ARNameType structure that specifies the name of the container.
- Type** An integer value indicating the type of container. The following table describes the possible values.
- Owner List** A list of the structures that limit the containers retrieved, based on the object that owns them.

## Container type values for ARContainerInfo

1	ARCON_GUI_DE	Guide
2	ARCON_APP	Application
3	ARCON_PACK	Packing List
4	ARCON_FILTER_GUI_DE	Filter Guide
5	ARCON_WEBSERVICE	Web Service

## Owner type values for ARContainerInfo

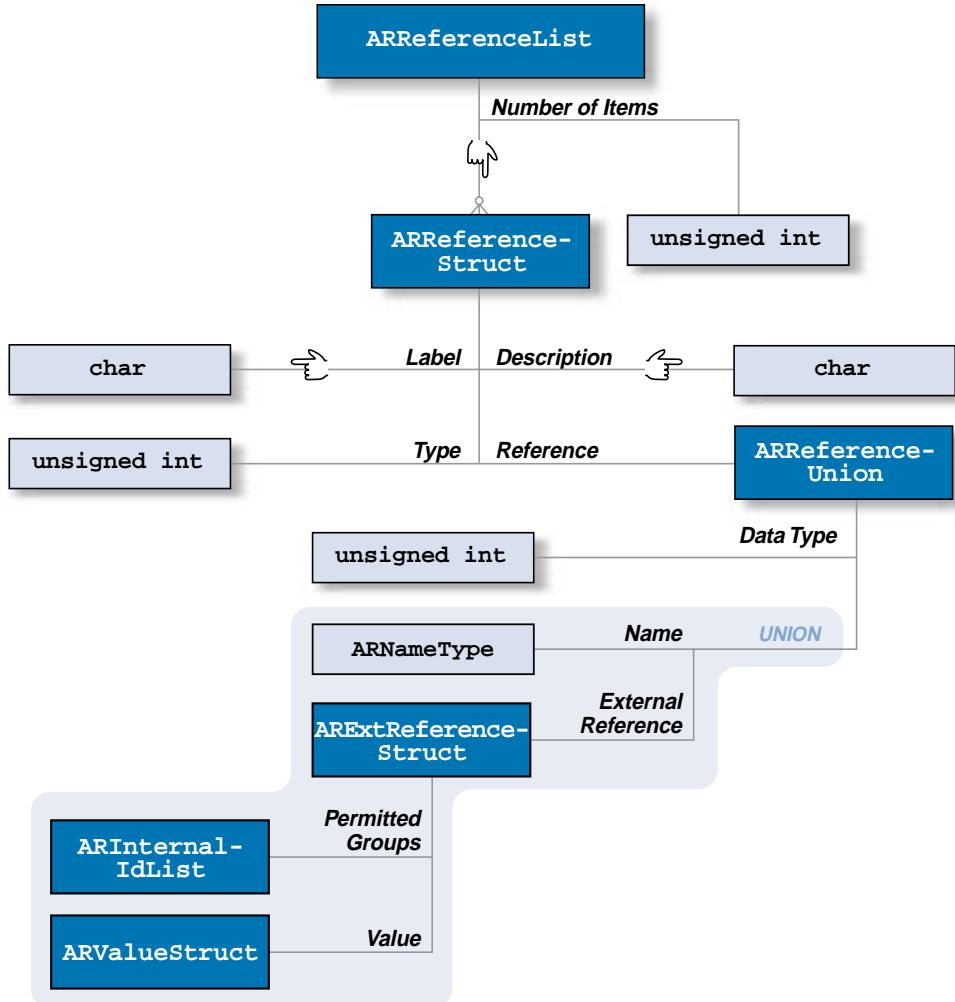
0	ARCONOWNER_NONE	Unowned container (global).
1	ARCONOWNER_ALL	Owned and unowned containers.
2	ARCONOWNER_SCHEMA	The owner object is a schema.

## Manipulating individual containers

The ARGetListContainer function (see page 244) returns a list of containers that match specified criteria. This list contains the names of the matching containers but does not contain their reference lists and other attributes. To retrieve or modify individual containers, you must pass the name as an input argument to the ARGetContainer or ARSetContainer functions (see page 219 and page 381).

Both of these functions, as well as the ARCreateContainer (see page 136) functions, use the ARReferenceList structure to specify the objects referenced by a container. Guides reference active links and applications reference schemas and other objects, both internal and external. The ARReferenceList structure consists of zero or more ARReferenceStruct structures, as the following figure shows.

Figure 3-29: Structures used to define container references



The ARReferenceStruct structure consists of the following elements:

<b>Label</b>	A string specifying a display-only label for the reference. Its length is limited by ARMAX_CON_LABEL_LEN (255 bytes).
<b>Description</b>	A string specifying a display-only description for the reference. Its length is limited by ARMAX_CON_DESCRIPTON_LEN (2000 bytes).
<b>Type</b>	An integer specifying the type of reference represented by the structure. The following table describes the possible values. Values above 32767 define external reference types.
<b>Reference</b>	A union that defines either an internal or external reference depending on its Data Type, which is either 0 (ARREF_DATA_ARSR) or 1 (ARREF_DATA_EXTREF). Internal references (0) are specified by using an ARI internal Id structure. External references (1) are specified by using an ARExtReferenceStruct structure, as Figure 3-29 on page 104 shows.

0	ARREF_NONE	No reference.
1	ARREF_ALL	All reference types.
2	ARREF_SCHEMA	Reference to a schema.
3	ARREF_FILTER	Reference to a filter.
4	ARREF_ESCALATION	Reference to an escalation.
5	ARREF_ACTIVELINK	Reference to an active link.
6	ARREF_CONTAINER	Reference to a container.
7	ARREF_CHARACTERMENU	Reference to a character menu.
32768	ARREF_ICON	Reference to an icon.
32769	ARREF_SMALL_ICON	Reference to a small icon.
32770	ARREF_MAXIMIZE_FORMS	Reference to a Boolean value that specifies whether to maximize the forms of an application.
32771	ARREF_APPLICATIION_FORMS	Specifies that the next reference in the list is a schema. Use this reference in front of each schema reference in your application except the primary schema.
32772	ARREF_ABOUT_BOX_IMAGE	Reference to an About box image.

32773	ARREF_ABOUT_BOX_FORM	Tag specifying that the About box form is the next reference.
32774	ARREF_NULL_STRING	Reference to a null string.
32775	ARREF_APPLICATI ON_HELP_EXT	Reference to a string that specifies the help file name extension.
32776	ARREF_APPLICATI ON_HELP_FILE	Reference to a bytelist that specifies the contents (not the name) of the help file to use with this application.
32777	ARREF_APPLICATI ON_PRI MARY_FORM	Specifies that the next reference in the list is the primary schema for this application. Use this reference in front of only one schema reference in your application.
32778	ARREF_APPLICATI ON_FORM_VUI	Reference to the ID of the schema view to use with the previous schema referenced in the list. Use this reference after each schema reference in your application.
32779	ARREF_APPLICATI ON_DI SABLE_BEGI N_TASK	Reference to a Boolean value that specifies whether the Begin a Task menu item is disabled for this application. The default is false (not disabled).
32780	ARREF_APPLICATI ON_HELP_I NDEX_EXT	Reference to application object help file's index file extension.
32781	ARREF_APPLICATI ON_HELP_I NDEX_FI LE	Reference to application object help file's index file.
32782	ARREF_APPLICATI ON_HELP_FI LE_NAME	Reference to application object help file's name, without the extension.
32783	ARREF_PACKI NGLİ ST_GUI DE	Packing list reference to a guide.
32784	ARREF_PACKI NGLİ ST_APP	Packing list reference to an application.
32785	ARREF_PACKI NGLİ ST_PACK	Packing list reference to a packing list.
32786	ARREF_GROUP_DATA	Packing list reference to data in the group schema.
32787	ARREF_DISTRIMAPPING_DATA	Packing list reference to data in the Distributed Mapping schema.
32788	ARREF_APPLICATI ON_HAS_EXT_HELP	Reference to whether applications use external help (Boolean).

32789	ARREF_APPLI_CATI_ON_SUPPORT_FILES	Reference to application object support file's contents.
32792	ARREF_PACKINGLIST_DSPPOOL	Packing lists reference to data in the Distributed Pool schema.
32793	ARREF_PACKINGLIST_FILTER_GUIDE	Packing lists reference to a filter guide.
32794	ARREF_FLASH_BOARD_DEF	Reference to Flashboard definition.
32795	ARREF_FLASH_DATA_SOURCE_DEF	Reference to Flashboard data source definition.
32796	ARREF_FLASH_VARIABLE_DEF	Reference to Flashboard variable definition.
32797	ARREF_WS_PROPERTIES	XML string that refers to miscellaneous properties of web service.
32798	ARREF_WS_OPERATION	Reference to web service operation information (XML string consisting of name, type, mapping names, and so on).
32799	ARREF_WS_ARXML_MAPPING	Mapping XML document that describes the relationship between AR System model and XML schema.
32800	ARREF_WS_WSDL	Reference to WSDL for web service.
32801	ARREF_PACKINGLIST_WEBSERVICE	Packing lists reference to a web service.
32802	ARREF_WS_PUBLISHING_LOC	Reference to saved URLs for publishing a web service.
32803	ARREF_APPLI_CATI_ON_HELPFILE_NAME2	Reference to application object help file's name, without the extension.
32804	ARREF_APPLI_CATI_ON_HELPFILE_EXT2	Reference to application object help file's extension.
32805	ARREF_APPLI_CATI_ON_HELPFILE2	Reference to application object help file.
32806	ARREF_APPLI_CATI_ON_HELPINDEX_EXT2	Reference to application object help file's index file extension.
32807	ARREF_APPLI_CATI_ON_HELPINDEXFILE2	Reference to application object help file's index file.
32808	ARREF_APPLI_CATI_ON_HELPFILE_NAME3	Reference to application object help file's name, without the extension.
32809	ARREF_APPLI_CATI_ON_HELPFILE_EXT3	Reference to application object help file's extension.

32810	ARREF_APPLI_CATI_ON_HELP_FI LE3	Reference to application object help file.
32811	ARREF_APPLI_CATI_ON_HELP_I_NDEX_EXT3	Reference to application object help file's index file extension.
32812	ARREF_APPLI_CATI_ON_HELP_I_NDEX_FI LE3	Reference to application object help file's index file.
32813	ARREF_APPLI_CATI_ON_HELP_FI LE_NAME4	Reference to application object help file's name, without the extension.
32814	ARREF_APPLI_CATI_ON_HELP_EXT4	Reference to application object help file's extension.
32815	ARREF_APPLI_CATI_ON_HELP_FI LE4	Reference to application object help file.
32816	ARREF_APPLI_CATI_ON_HELP_I_NDEX_EXT4	Reference to application object help file's index file extension.
32817	ARREF_APPLI_CATI_ON_HELP_I_NDEX_FI LE4	Reference to application object help file's index file.
32818	ARREF_APPLI_CATI_ON_HELP_FI LE_NAME5	Reference to application object help file's name, without the extension.
32819	ARREF_APPLI_CATI_ON_HELP_EXT5	Reference to application object help file's extension.
32820	ARREF_APPLI_CATI_ON_HELP_FI LE5	Reference to application object help file.
32821	ARREF_APPLI_CATI_ON_HELP_I_NDEX_EXT5	Reference to application object help file's index file extension.
32822	ARREF_APPLI_CATI_ON_HELP_I_NDEX_FI LE5	Reference to application object help file's index file.
32823	ARREF_APPLI_CATI_ON_HELP_LABEL	Reference to application object help file's label.
32824	ARREF_APPLI_CATI_ON_HELP_LABEL2	Reference to application object help file's label.
32825	ARREF_APPLI_CATI_ON_HELP_LABEL3	Reference to application object help file's label.
32826	ARREF_APPLI_CATI_ON_HELP_LABEL4	Reference to application object help file's label.
32827	ARREF_APPLI_CATI_ON_HELP_LABEL5	Reference to application object help file's label.
32828	ARREF_WS_XML_SCHEMA_LOC	Reference to XML schema location for a web service.

32829	ARREF_ENTRYPOINT_ORDER	Reference to listing order of the entry point.
32830	ARREF_ENTRYPOINT_START_ACTIVELINK	Reference to starting active link for the entry point.
32831	ARREF_APP_AUTOLAYOUT_SS	Reference to style sheet information for auto layout.
32832	ARREF_APP_FORMATATION_FIELDS	Reference to form action fields.
32833	ARREF_ENCAPSULATED_APP_DATA	Reference to application data identifier.
32835	ARREF_APP_DEFAULT_OBJ_PERMS	Reference to default application object permissions.
32836	ARREF_APP_ADD_FORMATATION_FIELDS	Reference to add form action fields.
32837	ARREF_APP_FORMATATION_RESULTS_LIST_FIXED_HEADER	Reference to fixed header property.
32838	ARREF_APP_FORMATATION_PAGE_PROPERTIES	Reference to page property.
32839	ARREF_APP_OBJECT_VERSION	Reference to application object version.
32840	ARREF_APP_PACKING_LISTS	Reference to packing lists in the application.
32841	ARREF_APP_DATA_MERGE_IMPORT_QUAL	Reference to list of field IDs for merge qualifications on import.
32842	ARREF_APP_DATA_MERGE_IMPORT_OPTION	Reference to data import option for merge on application import.

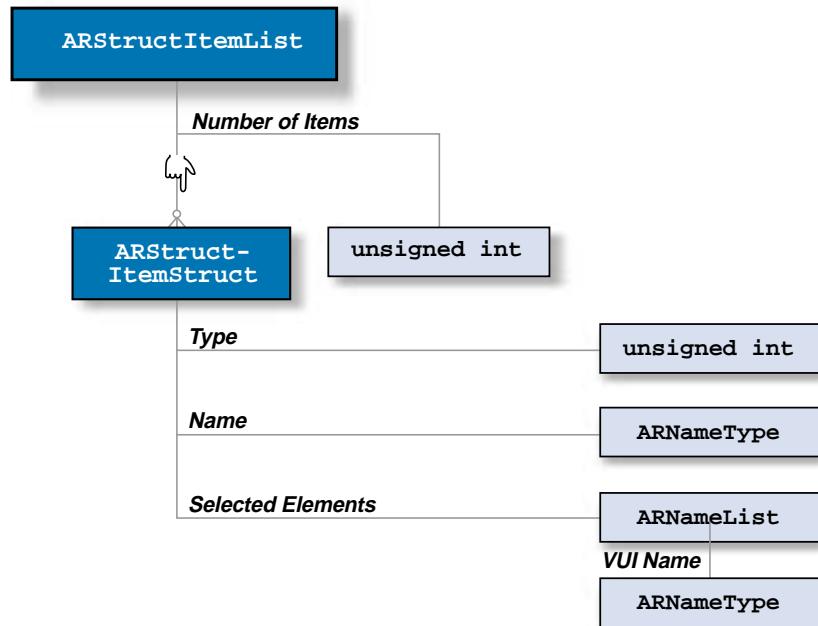
The ARExtReferenceStruct structure in Figure 3-29 on page 104 defines external references. This structure consists of the following elements:

<b>Permitted Groups</b>	An ARInternalList specifying the groups that have access to the referenced object.
<b>Value</b>	An ARValueStruct specifying the external reference.

## Importing and exporting

Both the ARExport and ARImport functions (see page 206 and page 363) use the ARStructItemList structure to represent the AR System objects to export or import. As illustrated in the following graphic, this list contains zero or more ARStructItemStruct items.

Figure 3-30: Structures used to import and export AR System objects



Each ARStructItemStruct represents a particular object to export or import:

<b>Item Type</b>	An integer value indicating the type of AR System object. The following table describes the possible values.
<b>Item Name</b>	The name associated with the AR System object.
<b>Selected Elements</b>	The server only uses this parameter during export and import actions.  During export actions, this parameter is only used when exporting data of the following types:  AR_STRUCT_ITEM_SCHEMA_MAIL - Specifies the VUI mail template to export.  During import actions, this parameter is only used when importing data of one type:  AR_STRUCT_ITEM_VUI - Specifies the VUIs to export and import from the schema. Specify the schema name in the <code>name</code> parameter. If the selected elements list is empty (zero elements), all schema VUIs are exported.  AR_STRUCT_ITEM_VUI - Specifies the VUIs to import from the schema. Specify the schema name in the <code>name</code> parameter. If the selected elements list is empty (zero elements), all schema VUIs are imported.
<b>VUI Name</b>	The name of the view.

1	AR_STRUCT_ITEM_SCHEMA	Schema definition, including views, help text, and change diary information.
2	AR_STRUCT_ITEM_SCHEMA_DEFN	Structure definition for configuring BMC Remedy User cache (special purpose export).
3	AR_STRUCT_ITEM_SCHEMA_VIEW	Display definition for configuring BMC Remedy User cache (special purpose export).
4	AR_STRUCT_ITEM_SCHEMA_MAIL	Email template for submitting requests (special purpose export).
5	AR_STRUCT_ITEM_FILTER	Filter definition.
6	AR_STRUCT_ITEM_ACTIVE_LINK	Active link definition.
7	AR_STRUCT_ITEM_ADMIN_EXT	No longer supported.
8	AR_STRUCT_ITEM_CHARACTER_MENU	Character menu definition.
9	AR_STRUCT_ITEM_ESCALATION	Escalation definition.
10	AR_STRUCT_ITEM_DIST_MAP	Mapping definition for Distributed Server Option.

---

11	AR_STRUCT_I TEM_SCHEMA_VI EW_MI_N	Display definition without bitmaps for BMC Remedy User.
12	AR_STRUCT_I TEM_CONTAINER	Container definition.
13	AR_STRUCT_I TEM_DI ST_POOL	Pool definition for the attachments.
14	AR_STRUCT_I TEM_VUI	View definition.
15	AR_STRUCT_I TEM_FIELD	Field definition. This item is supported only in the XML API.
16	AR_STRUCT_I TEM_APP	Application definition.
30	AR_STRUCT_I TEM_SCHEMA_DATA	Schema data definition.
103	AR_STRUCT_I TEM_SCHEMA_VI EW_2	Definition tag for all schema views instead of just one.

---

**Note:** To import *all* structures in the import buffer, specify `NULL` for the `structItems` parameter or an `ARStructItemList` with zero items.

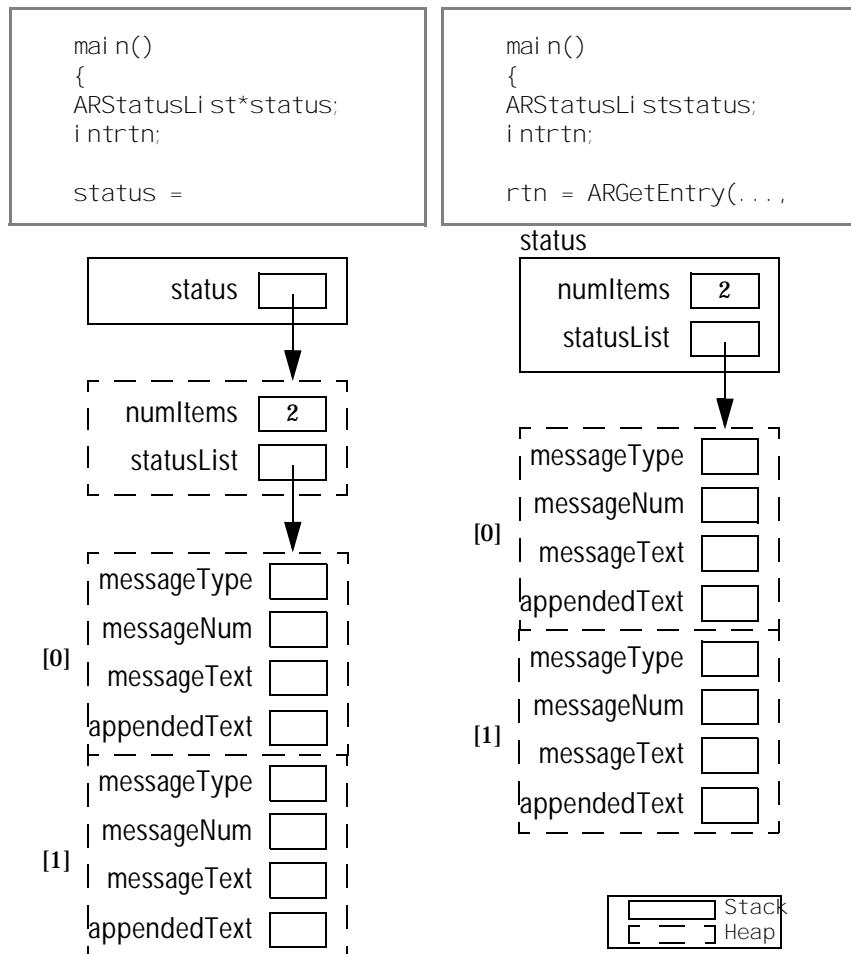
## Freeing allocated memory

Many of the AR System data structures use allocated memory. In the case of **input** parameters, allocate the necessary space by using the `malloc` function. Memory for **output** parameters is allocated dynamically by the system based on their content. You (the caller of the API function) are the only one who knows whether you allocated space for input parameters, and when you are finished with output parameters, you are responsible for freeing *all* allocated memory. To avoid a buildup of allocated space, free memory as soon as you no longer need it.

To ease the process of freeing memory, the API includes a series of `FreeAR` functions, each of which is specialized to free all allocated memory associated with a particular data structure. These functions assume that all structure components are in allocated memory and should not be used if any components of a structure are on the stack. In this case, you must free the memory manually by using the `free` function.

All of the `Free` functions accept two input arguments. The `value` parameter is a pointer to the structure you want to free. The function recursively frees all allocated memory within that structure. If you specify `NULL` for this parameter (or the structure is a list with zero items), the function performs no operations.

The `freeStruct` parameter is a Boolean value that indicates whether you need to free the top level structure. If you allocated memory for the top level structure, specify `TRUE` to free both the structure and its contents. If you used a stack variable for the top level structure, specify `FALSE` to free only the contents of the structure. The following provides an example that illustrates this difference.



When an API function fails (that is, `status` greater than one), the system initializes all output parameters other than `ARStatusList`. As a result, the only `Free` function you *must* call is `FreeARStatusList`. Calling `FreeAR<data structure>` for the other parameters is not necessary in this situation but causes no harm.

---

Note: For additional information about the FreeAR<*data structure*> header files, see the *Installing* guide.

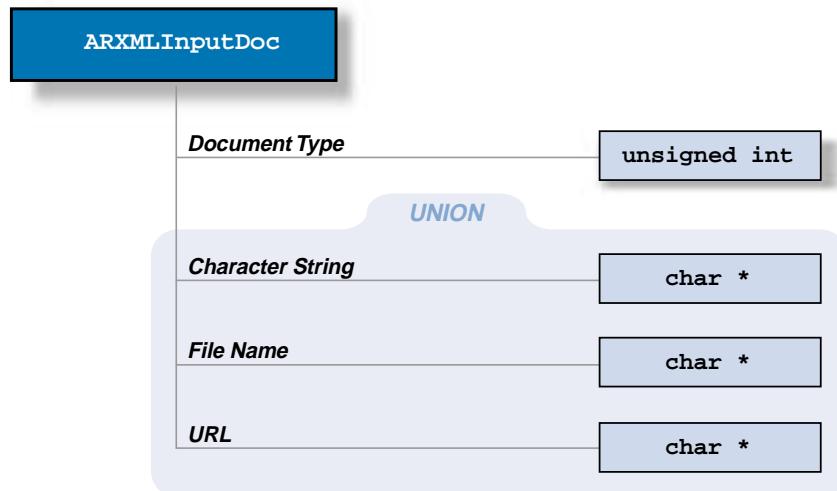
---

## XML formats

To transform XML documents, use the ARXMLInputDoc and ARXMLOutputDoc formats.

The following figure shows the ARXMLInputDoc format.

Figure 3-31: ARXMLInputDoc format

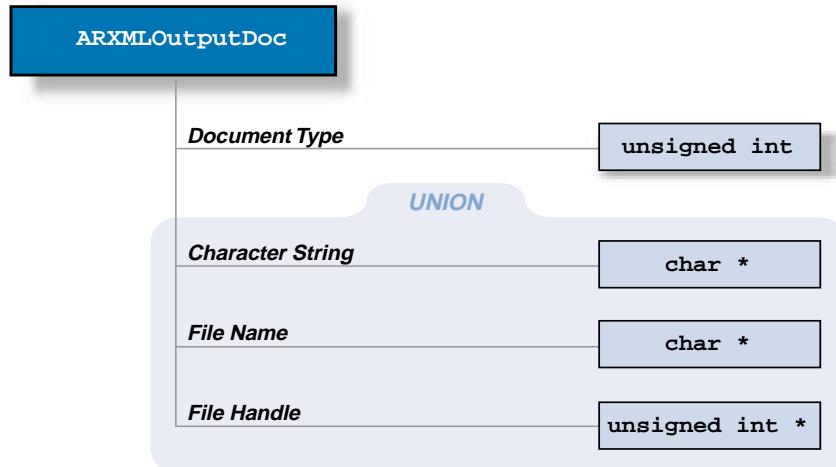


The ARXML InputDoc format, which you use to transform XML documents to AR System structures, consists of the following elements:

- Document Type** The type of XML documents.
- Character String** The null-terminated character string of the XML document (AR\_XML\_DOC\_CHAR\_STR).
- File Name** The file name of the XML document (AR\_XML\_DOC\_FILE\_NAME).
- URL** The URL of the XML document (AR\_XML\_DOC\_URL).

The following figure shows the ARXMLOutputDoc format.

Figure 3-32: ARXMLEOutputDoc format



The ARXMLEOutputDoc format, which you use to transform AR System structures to XML documents, consists of the following elements:

- |                         |   |
|-------------------------|---|
| <b>Document Type</b>    | The type of XML documents.  |
| <b>Character String</b> | The null-terminated character string of the XML document (AR_XML_DOC_CHAR_STR). |
| <b>File Name</b>        | The file name of the XML document (AR_XML_DOC_FILE_NAME).                       |
| <b>File Handle</b>      | The file handle of the XML document (AR_XML_DOC_FILE_HANDLE).                   |



Chapter

# 4

# AR System C API calls

This chapter describes the AR System C API functions.

The following topics are provided:

- Related files (page 121)
- Types of functions (page 121)
- Function descriptions (page 127)
- ARBeginBulkEntryTransaction (page 127)
- ARCreateActiveLink (page 128)
- ARCreateAlertEvent (page 132)
- ARCreateCharMenu (page 134)
- ARCreateContainer (page 136)
- ARCreateEntry (page 139)
- ARCreateEscalation (page 140)
- ARCreateField (page 143)
- ARCreateFilter (page 160)
- ARCreateLicense (page 163)
- ARCreateSchema (page 164)
- ARCreateSupportFile (page 168)
- ARCreateVUI (page 169)
- ARDateToJulianDate (page 175)
- ARDecodeAlertMessage (page 176)

- ARDecodeARAssignStruct (page 179)
- ARDecodeARQualifierStruct (page 180)
- ARDecodeDiary (page 181)
- ARDecodeStatusHistory (page 182)
- ARDeleteActiveLink (page 183)
- ARDeleteCharMenu (page 184)
- ARDeleteContainer (page 185)
- ARDeleteEntry (page 186)
- ARDeleteEscalation (page 187)
- ARDeleteField (page 189)
- ARDeleteFilter (page 190)
- ARDeleteLicense (page 191)
- ARDeleteMultipleFields (page 192)
- ARDeleteSchema (page 193)
- ARDeleteSupportFile (page 195)
- ARDeleteVUI (page 196)
- ARDeregisterForAlerts (page 197)
- AREncodeARAssignStruct (page 198)
- AREncodeARQualifierStruct (page 199)
- AREncodeDiary (page 200)
- AREncodeStatusHistory (page 201)
- AREndBulkEntryTransaction (page 202)
- ARExecuteProcess (page 203)
- ARExpandCharMenu (page 204)
- ARExport (page 206)
- ARExportLicense (page 210)
- ARGetActiveLink (page 211)
- ARGetAlertCount (page 214)
- ARGetApplicationState (page 215)
- ARGetCharMenu (page 216)
- ARGetClientCharSet (page 218)
- ARGetContainer (page 219)
- ARGetCurrencyRatio (page 222)
- ARGetEntry (page 223)

- ARGetEntryBLOB (page 225)
- ARGetEntryBlock (page 227)
- ARGetEntryStatistics (page 228)
- ARGetEscalation (page 230)
- ARGetField (page 233)
- ARGetFilter (page 237)
- ARGetListActiveLink (page 240)
- ARGetListAlertUser (page 241)
- ARGetListApplicationState (page 242)
- ARGetListCharMenu (page 243)
- ARGetListContainer (page 244)
- ARGetListEntry (page 246)
- ARGetListEntryBlocks (page 249)
- ARGetListEntryWithFields (page 252)
- ARGetListEscalation (page 254)
- ARGetListExtSchemaCandidates (page 256)
- ARGetListField (page 257)
- ARGetListFilter (page 258)
- ARGetListGroup (page 260)
- ARGetListLicense (page 261)
- ARGetListRole (page 262)
- ARGetListSchema (page 263)
- ARGetListSchemaWithAlias (page 266)
- ARGetListServer (page 269)
- ARGetListSQL (page 270)
- ARGetListSupportFile (page 271)
- ARGetListUser (page 273)
- ARGetListVUI (page 274)
- ARGetLocalizedValue (page 275)
- ARGetMultipleActiveLinks (page 276)
- ARGetMultipleCharMenus (page 281)
- ARGetMultipleContainers (page 284)
- ARGetMultipleCurrencyRatioSets (page 288)
- ARGetMultipleEntryPoints (page 289)

- ARGetMultipleEntries (page 293)
- ARGetMultipleEscalations (page 295)
- ARGetMultipleExtFieldCandidates (page 298)
- ARGetMultipleFields (page 299)
- ARGetMultipleFilters (page 304)
- ARGetMultipleLocalizedValues (page 308)
- ARGetMultipleSchemas (page 309)
- ARGetMultipleVUIs (page 314)
- ARGetSchema (page 316)
- ARGetServerCharSet (page 320)
- ARGetServerInfo (page 322)
- ARGetSessionConfiguration (page 349)
- ARGetServerStatistics (page 351)
- ARGetSupportFile (page 358)
- ARGetTextForErrorMessage (page 360)
- ARGetVUI (page 360)
- ARImport (page 363)
- ARImportLicense (page 366)
- ARInitialization (page 367)
- ARJulianDateToDate (page 368)
- ARLoadARQualifierStruct (page 369)
- ARMergeEntry (page 370)
- ARRegisterForAlerts (page 372)
- ARSetActiveLink (page 373)
- ARSetApplicationState (page 378)
- ARSetCharMenu (page 379)
- ARSetContainer (page 381)
- ARSetEntry (page 384)
- ARSetEscalation (page 386)
- ARSetField (page 389)
- ARSetFilter (page 393)
- ARSetImpersonatedUser (page 397)
- ARSetLogging (page 397)
- ARSetSchema (page 399)

- ARSetServerInfo (page 403)
- ARSetServerPort (page 405)
- ARSetSessionConfiguration (page 406)
- ARSetSupportFile (page 408)
- ARSetVUI (page 410)
- ARSignal (page 412)
- ARTermination (page 413)
- ARValidateFormCache (page 413)
- ARValidateLicense (page 416)
- ARValidateMultipleLicenses (page 417)
- ARVerifyUser (page 418)
- FreeAR (page 419)

## Related files

The AR System server uses API functions to perform operations on all AR System objects. The `arextern.h` file contains the API function definitions. The `arfree.h` file contains the definitions of associated routines that free allocated memory see “Freeing allocated memory” on page 112).

## Types of functions

The API functions consist of object manipulation functions, notification functions, and general use functions, such as functions that initialize or terminate sessions, authentication functions, and functions that import or export object definitions.

## Object manipulation functions

You can perform five primary operations (*create*, *delete*, *get* (retrieve), *get list* (retrieve a list), and *set* (modify)) for each of the following objects:

- Active Links
- Containers
- Entries
- Escalations

- Fields
- Filters
- Character menus
- Forms (schemas)
- Support files
- VUIs

For example, you can retrieve field properties or create entries for a specified form. Users with administrator capability can create or delete forms or other objects on a specified server. Some server objects can also be manipulated in other ways by additional functions. Specifically, these are the AR System functions associated with each object.

---

**Note:** Functions that are used to perform administrative operations are noted by an asterisk (\*).

---

## Active links

- ARCreateActi veLi nk\*
- ARDel eteActi veLi nk\*
- ARGetActi veLi nk
- ARGetLi stActi veLi nk
- ARGetMul ti pl eActi veLi nks
- ARSetActi veLi nk\*

## Containers

- ARCreateContai ner\*
- ARDel eteContai ner\*
- ARGetContai ner
- ARGetLi stContai ner
- ARGetMul ti pl eContai ners
- ARSetContai ner\*

## Entries

- ARCreateEntry
- ARDeleteEntry\*
- ARGetEntry
- ARGetEntryBLOB
- ARGetEntryBlock
- ARGetEntryStatistics
- ARGetListEntry
- ARGetListEntryBlocks
- ARGetListEntryWithFields
- ARGetMultipleEntries
- ARMergeEntry
- ARSetEntry

## Escalations

- ARCreateEscalation\*
- ARDeleteEscalation\*
- ARGetEscalation
- ARGetListEscalation
- ARGetMultipleEscalations
- ARSetEscalation\*

## Fields

- ARCreateField\*
- ARDeleteField\*
- ARDeleteMultipleFields\*
- ARGetField
- ARGetListEntryWithFields
- ARGetListField
- ARGetMultipleFieldsCandidates
- ARGetMultipleFields
- ARSetField\*

## Filters

- ARCreateFilter\*
- ARDeleteFilter\*
- ARGetFilter
- ARGetListFilter
- ARGetMultipleFilter
- ARSetFilter\*

## Character menus

- ARCreateCharMenu\*
- ARDeleteCharMenu\*
- ARExpandCharMenu
- ARGetCharMenu
- ARGetListCharMenu
- ARGetMultipleCharMenus
- ARSetCharMenu\*

## Forms (schemas)

- ARCreateSchema\*
- ARDeleteSchema\*
- ARGetListExtSchemaCandidates
- ARGetListSchema
- ARGetListSchemaWithAlias
- ARGetMultipleSchemas
- ARGetSchema
- ARSetSchema\*

## Support files

- ARCreateSupportFile\*
- ARDeleteSupportFile\*
- ARGetListSupportFile
- ARGetSupportFile
- ARSetSupportFile\*

## VUIs

- ARCreateVUI \*
- ARDeleteVUI \*
- ARGetListVUI
- ARGetMultipleVUIs
- ARGetVUI
- ARSetVUI \*

## Alert functions

The AR System includes the following alert functions that register users, cancel user registration, enter alert events into the system, and retrieve a list of alert users.

- ARCreateAlertEvent
- ARDecodeAlertMessage
- ARDeregisterForAlerts
- ARGetAlertCount
- ARGetListAlertUser
- ARRegisterForAlerts

## Other functions

The API also includes functions for a variety of other operations, such as localization, data structure, housekeeping, object definition, server processes, and text manipulation.

## Access control

- ARGetListGroup
- ARGetListRole\*
- ARGetListUser
- ARValiateFormCache
- ARVerifyUser

## Bulk entry

- ARBeginBulkEntryTransaction
- AREndBulkEntryTransaction

## Currency

- ARGetCurrencyRatio
- ARGetMultipleCurrencyRatioSets

## Data structure help functions

- ARDateToJulianDate
- ARJulianDateToDate
- ARDecodeARAssignStruct
- ARDecodeARQualifierFileInfoStruct
- ARDecodeDictionary
- ARDecodeStatusHistory
- AREncodeARAssignStruct
- AREncodeARQualifierFileInfoStruct
- AREncodeDictionary
- AREncodeStatusHistory
- ARLoadARQualifierFileInfoStruct

## Housekeeping

- ARInitiation
- ARTermination

## Licensing

- ARCreateLicense
- ARGetLicense
- ARDeleteLicense\*
- ARExportLicense\*
- ARImportLicense\*
- ARValidateDateLicense
- ARValidateMultipleLicenses

## Object definitions and external data manipulation

- ARExport
- ARImport\*
- ARGetListSQL

## Server processes

- ARGetListServer
- ARGetLocalizedValue
- ARGetMultipliedLocalizedValues
- ARGetServerInfo
- ARGetServerStatistics
- ARExecuteProcess\*
- ARSetLogging
- ARSetServerInfo\*
- ARSetServerPort

## Text manipulation

- ARSetFullTextInfo\*
- ARGetTextForErrorMessage
- ARSetImpersonatedUser

# Function descriptions

This section describes each API function call and its components.

---

**Note:** In API function descriptions, when a value is said to be specified, it means the value is found within the synopsis for that particular API function.

---

## ARBeginBulkEntryTransaction

Description	Marks the beginning of a series of entry API function calls that will be grouped together and sent to the AR System server as part of one transaction. All calls related to create, set, delete, and merge operations made between this API call and a trailing <code>AREndBulkEntryTransaction</code> call will not be sent to the server until the trailing call is made.
Privileges	All users.

**Synopsis**

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

int ARBeginBulkEntryTransaction(
    ARControlStruct *control,
    ARStatusList *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**Return values****status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

`AREndBulkEntryTransaction`.

## ARCreateActiveLink

**Description**

Creates a new active link with the indicated name on the specified server. The active link is added to the server immediately and returned to users who request information about active links. Because active links operate on clients, individual clients do not receive the new definition until they reconnect to the form (thus reloading the form from the server).

**Privileges**

AR System administrator.

**Synopsis**

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

int ARCreateActiveLink(
    ARControlStruct *control,
    ARNameType name,
    unsigned int order,
    ARWorkflowConnectStruct *schemaList,
    ARInternalLinkList *groupList,
    unsigned int executeMask,
```

ARIInternalId	*controlField,
ARIInternalId	*focusField,
unsigned int	enable,
ARQualifierFilterStruct	*query,
ARActiveLinkActionList	*actionList,
ARActiveLinkActionList	*elseList,
char	*helpText,
ARAccessNameType	owner,
char	*changeDictionary,
ARPropList	*objectPropList,
ARStatusList	*status)

## control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

## name

The name of the active link to create. The names of all active links on a given server must be unique.

## order

A value between 0 and 1000 (inclusive) that determines the active link execution order. When multiple active links are associated with a form, the value associated with each active link determines the order in which they are processed (from lowest to highest).

## schemaList

The list of form names the active link is linked to. The active link must be associated with a single form or a list of forms that currently exists on the server.

## groupList

A list of zero or more groups who can access this active link. Users can execute an active link if they belong to a group that has access to it. Specifying an empty group list defines an active link accessible by users with administrator capability only. Specifying group ID 0 (Public) provides access to all users.

## executeMask

A bitmask indicating the form operations that trigger the active link.

- Bit 0: Execute the active link when a user selects a button, toolbar button, or menu item specified by the controlField parameter (AR\_EXECUTE\_ON\_BUTTON).
- Bit 1: Execute the active link when a user presses Return in field specified by the focusField parameter (AR\_EXECUTE\_ON\_RETURN).
- Bit 2: Execute the active link when a user submits an entry (*before* data is sent to the AR System server) (AR\_EXECUTE\_ON\_SUBMIT).
- Bit 3: Execute the active link when a user modifies an individual entry (*before* data is sent to the AR System server) (AR\_EXECUTE\_ON\_MODIFY).
- Bit 4: Execute the active link when a user displays an entry (after data is retrieved from the AR System server) (AR\_EXECUTE\_ON\_DISPLAY).
- Bit 7: Execute the active link when a user selects an item from a character menu associated with a field specified by the focusField parameter or selects a row in a table field specified by the focusField parameter (AR\_EXECUTE\_ON\_MENU\_CHOICE).
- Bit 9: Execute the active link when a user sets default values (either manually or through preference settings) (AR\_EXECUTE\_ON\_SET\_DEFAULT).
- Bit 10: Execute the active link when a user retrieves one or more entries (*before* the query is sent to the AR System server) (AR\_EXECUTE\_ON\_QUERY).
- Bit 11: Execute the active link when a user modifies an individual entry (*after* data is committed to the database) (AR\_EXECUTE\_ON\_AFTER MODIFY).
- Bit 12: Execute the active link when a user submits an entry (*after* data is committed to the database) (AR\_EXECUTE\_ON\_AFTER\_SUBMIT).
- Bit 14: Execute the active link when a user opens any form window or changes its mode (AR\_EXECUTE\_ON\_WINDOW\_OPEN).
- Bit 15: Execute the active link when a user closes any form window or changes its mode (AR\_EXECUTE\_ON\_WINDOW\_CLOSE).

## controlField

The ID of the field that represents the button, toolbar button, or menu item associated with executing the active link. This parameter is ignored if you do not specify the AR\_EXECUTE\_ON\_BUTTON condition (see the executeMask parameter on page 130).

## focusField

The ID of the field associated with executing the active link by pressing Return, selecting a character menu item, or gaining or losing focus. You must create another active link to specify a different field for each condition. This parameter is ignored if you do not specify one of the following conditions: AR\_EXECUTE\_ON\_RETURN, AR\_EXECUTE\_ON\_MENU\_CHOICE, AR\_EXECUTE\_ON\_LOSE\_FOCUS, or AR\_EXECUTE\_ON\_GAIN\_FOCUS (see the executeMask parameter on page 130).

## enable

A flag to enable or disable this active link. A value of 0 disables the active link, causing it to be invisible to the user and unavailable for use. A value of 1 enables the active link, causing it to be visible and available for use.

## query

A qualification that determines whether the active link is executed. Specify NULL or assign an operation value of 0 (AR\_COND\_OP\_NONE) to execute the active link unconditionally.

## actionList

The set of actions performed if the condition defined by the query parameter is satisfied. You can specify from 1 to 25 actions in this list (limited by AR\_MAX\_ACTIONS).

## elseList

The set of actions performed if the condition defined by the query parameter is not satisfied. You can specify from 0 to 25 actions in this list (limited by AR\_MAX\_ACTIONS). Specify NULL for this parameter (or zero actions) if you do not want to define any else actions.

## helpText

The help text associated with the active link. This text can be of any length. Specify NULL for this parameter if you do not want to associate help text with this object.

## owner

The owner for the active link. The owner defaults to the user performing the operation if you specify NULL for this parameter.

## changeDiary

The initial change diary associated with the active link. This text can be of any length. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to associate change diary text with this object.

## objPropList

If the `objPropList` parameter is `NULL`, an object properties list with zero properties will be associated with the object, and zero properties are returned when an `ARGetActiveLink` is performed. See “Server object property tags” on page 79 for object property tag names and data types.

### Return values

#### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

### See also

`ARDeleteActiveLink`, `ARDeleteField`, `ARGetActiveLink`, `ARGetField`, `ARGetListActiveLink`, `ARGetListField`, `ARGetMultipleActiveLinks`, `ARSetActiveLink`, `ARSetField`. See `FreeAR` for: `FreeARActiveLinkActionList`, `FreeARInternalIdList`, `FreeARQualifierStruct`, `FreeAR`, `FreeARPropList`, `FreeARWorkflowConnectStruct`.

## ARCreateAlertEvent

**Description** Enters an alert event on the specified server. The AR System server sends an alert to the specified, registered users.

**Privileges** All users.

### Synopsis

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

```
int ARCreateAlertEvent(
    ARControlStruct *control,
    ARAccessType user,
    *alertText,
    priority,
    sourceTag,
    serverName,
```

ARNameType	formName,
char	*obj ectI d,
AREntryIdType	*entryId,
ARStatusList	*status)

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

### user

The user who receives the alert. Specify \* (AR\_REGISTERED\_BROADCAST) to create an alert event for all users that are currently registered to receive alerts with the AR System server. You cannot specify a group name for this argument.

### alertText

The text that the alert contains.

### priority

A relative value that represents the priority for this alert. The range of acceptable values is between 0 and 10.

### sourceTag

A string that identifies the source of the alert. The AR System provides two predefined values for this string:

- AR - alert originated from the AR System
- FB - alert originated from Flashboards

### serverName

The name of the server that is the source of the alert. Use @ to specify the current server. Specify NULL for this parameter if the parameter is not applicable to the type of alert that this call creates.

### formName

The name of the form that is the source of the alert. For Flashboards, this is the name of the Flashboard that generated the alert. Specify NULL for this parameter if the parameter is not applicable to the type of alert that this call creates.

**objectId**

The identifier for the object. For AR System, this value is the Entry ID of the originating request. For Flashboards, this value is the name of the Flashboard alert that the user provides. Specify `NULL` for this parameter if the parameter is not applicable to the type of alert that this call creates. Join forms have multiple entry Ids (`0000000000000001 | 0000000000000002`) separated by vertical bars.

**Return values****entryId**

The unique identifier for the new alert (system-generated).

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** `ARRegisterForAlerts`, `ARDeregisterForAlerts`. See `FreeAR` for: `FreeARStatusList`.

## ARCreateCharMenu

**Description** Creates a new character menu with the indicated name on the specified server.

**Privileges** AR System administrator.

**Synopsis**

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

int ARCreateCharMenu(
    ARControlStruct *control,
    ARNameType name,
    refreshCode,
    *menuDefn,
    *helpText,
    owner,
    *changeDir,
    *objPropList,
    *status)
```

**Input  
arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**name**

The name of the character menu to create. The names of all character menus on a given server must be unique.

**refreshCode**

A value indicating when the menu is refreshed. This parameter enables you to balance menu consistency with performance.

- 1: Refresh only when form opened (`AR_MENU_REFRESH_CONNECT`).
- 2: Refresh every time menu opened (`AR_MENU_REFRESH_OPEN`).
- 3: Refresh first time menu opened and every 15 minutes thereafter (`AR_MENU_REFRESH_INTERVAL`).

**menuDefn**

The definition of the character menu.

**helpText**

The help text associated with the character menu. This text can be of any length. Specify `NULL` for this parameter if you do not want to associate help text with this object.

**owner**

The character menu owner. The owner defaults to the user performing the operation if you specify `NULL` for this parameter.

**changeDiary**

The initial change diary associated with the character menu. This text can be of any length. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to associate change diary text with this object.

## objPropList

If the objPropList parameter is set to NULL, an object properties list with zero properties will be associated with the character menu, and a list of zero properties is returned when an ARGetCharMenu is performed. See “Server object property tags” on page 79 for object property tag names and data types.

### Return values

#### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

### See also

ARDeleteCharMenu, ARExpandCharMenu, ARGetCharMenu, ARGetListCharMenu, ARSetCharMenu. See FreeAR for: FreeARCharMenuStruct, FreeARStatusList, FreeARPropList.

## ARCreateContainer

**Description** Creates a new container with the indicated name on the specified server. Use this function to create applications, active links, active link guides, filter guide, packing lists, guides, and AR System-defined container types. A container can also be a custom type that you define.

**Privileges** AR System administrator.

### Synopsis

```
#i ncl ude "ar.h"
#i ncl ude "arerrno.h"
#i ncl ude "arextern.h"
#i ncl ude "arstruct.h"

int ARCreateContainer(
    ARControlStruct *control,
    ARNameType name,
    *groupList,
    *adminGroupList,
    *ownerObjList,
    *label,
    *description,
    *type,
    *references,
    removeFlag,
    *helpText,
```

ARAccessNameType char ARPropList ARStatusList	owner, *changeDictionary, *objPropList, *status)
--	---

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**name**

The name of the container to create. The names of all containers on a given server must be unique.

**groupList**

A list of zero or more groups who can access this container. Specifying an empty group list defines a container accessible by users with administrator capability only. Specifying group ID 0 (Public) provides access to all users. The permission value you assign for each group determines whether users in that group see the container in the container list.

- 1: Users see the container in the container list (AR\_PERMISSIONS\_VISIBLE).
- 2: Users do not see the container in the container list (AR\_PERMISSIONS\_HIDDEN).

**adminGrpList**

A list of zero or more groups who can administer this container (and the referenced objects). If `ownerObj` is not `NULL`, this parameter is ignored and the Subadministrator group list of the owning form is used instead. Users must belong to both a specified group *and* the Subadministrator group to obtain these privileges. Specifying an empty administrator group list defines a container that can be administered by users with administrator capability only. Specifying group ID 0 (Public) provides subadministrator capability to all members of the Subadministrator group.

**ownerObjList**

A list of schemas that own this container. This parameter can be `NULL` if the container exists globally.

**label**

The label for this container. It can be as many as 255 characters long or `NULL`.

## description

The description for this container. It can be as many as 2000 characters long or `NULL`.

## type

The type for this container—either guide (`ARCON_GUI_DE`), application (`ARCON_APP`), or a custom type you have defined.

## references

A list of pointers to the objects referenced by this container. References can be to internal AR System objects (for example, guides reference active links and applications reference forms) or to external objects such as URLs or file names. Specify `NULL` for this parameter if you do not want to associate any objects with this container.

## removeFlag

A flag specifying how invalid object references are removed when the container is created. If `FALSE`, references to nonexistent AR System objects will be removed with no error generated. If `TRUE`, an error will be generated.

## helpText

The help text associated with the container. This text can be of any length. Specify `NULL` for this parameter if you do not want to associate help text with this object.

## owner

The owner for the container. The owner defaults to the user performing the operation if you specify `NULL` for this parameter.

## changeDiary

The initial change diary associated with the container. This text can be of any length. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to associate change diary text with this object.

## objPropList

If `objPropList` parameter is set to `NULL`, an object properties list with zero properties will be associated with the container, and a list of zero properties is returned when an `ARGetContainer` is performed. See “Server object property tags” on page 79 for object property tag names and data types.

**Return values** [status](#)

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateSchema, ARDeleteContainer, ARGetContainer, ARGetListContainer, ARGetListEntry, ARSetContainer. See FreeAR for: FreeARContainerInfoList, FreeARInternalIdList, FreeARPermissionList, FreeARPropList, FreeARReferenceList, FreeARStatusList.

## ARCreateEntry

**Description** Creates a new entry in the indicated schema (form) on the specified server. You can create entries in base schemas only. To add entries to join forms, create them in one of the underlying base forms.

**Privileges** The system creates data based on the access privileges of the user you specify for the `control` parameter and the `createMode` setting for each field (see “ARCreateField” on page 143). User permissions are verified for each specified field. The system generates an error if the user does not have write permission for a field or a field does not exist.

**Synopsis**

```
#i ncl ude "ar.h"
#i ncl ude "arerrno.h"
#i ncl ude "arextern.h"
#i ncl ude "arstruct.h"

int ARCreateEntry(
    ARControl Struct          *control,
    ARNameType                 schema,
    ARFieldListStruct          *fieldList,
    AREntryIdType              entryId,
    ARStatusListStruct         *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**schema**

The name of the schema to create the entry in.

## fieldList

A list of one or more field/value pairs (specified in any order) that identifies the data for the new entry. You must specify values for all required fields that do not have defined defaults. Values must be of the data type defined for the field or have a data type of 0 (AR\_DATA\_TYPE\_NULL). NULL values can be specified for optional fields only, and assigning NULL overrides any defined field default. An error is generated if a field does not exist or the user specified by the control parameter does not have write permission for a field.

### Return values

#### entryId

The unique identifier for the new entry (system-generated).

#### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARDeleteEntry, ARGetEntry, ARGetListEntry, ARMergeEntry, ARSetEntry. See FreeAR for: FreeARFieldValueList, FreeARStatusList.

## ARCreateEscalation

**Description** Creates a new escalation with the indicated name on the specified server. The escalation condition is checked regularly based on the time structure defined when it is enabled.

**Privileges** AR System administrator.

### Synopsis

```
#i ncl ude "ar. h"
#i ncl ude "arerrno. h"
#i ncl ude "arextern. h"
#i ncl ude "arstruct. h"
```

```
int ARCreateEscalation(
    ARControlStruct *control,
    ARNameType name,
    AREscalationTmStruct *escalationTm,
    *schemaList,
    enable,
    *query,
    *actionList,
    *elseList,
```

```

char *helpText,
ARAccessNameType owner,
char *changeDictionary,
ARPropList *objectPropList,
ARStatusList *status)

```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

### name

The name of the escalation to create. The names of all escalations on a given server must be unique.

### escalationTm

The time specification for evaluating the escalation condition. This parameter can take one of two forms: a time interval that defines how frequently the server checks the escalation condition (in seconds) or a bitmask that defines a particular day (by month or week) and time (hour and minute) for the server to check the condition.

### schemaList

The list of form names the escalation is linked to. The escalation must be associated with a single form or a list of forms that currently exists on the server.

### enable

A flag to enable or disable this escalation. A value of 0 disables the escalation, causing its condition checks and associated actions to not be performed. A value of 1 enables the escalation, causing its conditions to be checked at the specified time interval.

### query

A query operation performed when the escalation is executed that determines the set of entries to which the escalation actions (defined by the `actionList` parameter) are applied. Specify `NULL` or assign an operation value of 0 (`AR_COND_OP_NONE`) to match all schema entries.

### **actionList**

The set of actions performed for each entry that matches the criteria defined by the `query` parameter. You can specify from 1 to 25 actions in this list (limited by `AR_MAX_ACTIONS`).

### **elseList**

The set of actions performed if no entries match the criteria defined by the `query` parameter. These actions are *not* performed for all non-matching entries. You can specify from 0 to 25 actions in this list (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter (or zero actions) if you do not want to define any `else` actions.

### **helpText**

The help text associated with the escalation. This text can be of any length. Specify `NULL` for this parameter if you do not want to associate help text with this object.

### **owner**

The owner for the escalation. The owner defaults to the user performing the operation if you specify `NULL` for this parameter.

### **changeDiary**

The initial change diary associated with the escalation. This text can be of any length. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to associate change diary text with this object.

### **objPropList**

If `objPropList` parameter is set to `NULL`, an object properties list with zero properties will be associated with the escalation, and a list of zero properties is returned when an `ARGetEscalation` is performed. See “Server object property tags” on page 79 for object property tag names and data types.

## **Return values**

### **status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARDeleteEscalation, ARGetEscalation, ARGetListEscalation, ARSetEscalation. See FreeAR for: FreeARFilterActionList, FreeARQualifierStruct, FreeARStatusList, FreeARWorkflowConnectStruct, FreeARPropList.

## ARCreateField

<b>Description</b>	Creates a new field with the indicated name on the specified server. Forms can contain data and nondata fields. Nondata fields serve several purposes. Trim fields enhance the appearance and usability of the form (for example, lines, boxes, or static text). Control fields provide mechanisms for executing active links (for example, menus, buttons, or toolbar buttons). Other nondata fields organize data for viewing (for example, pages and page holders) or show data from another form (for example, tables and columns). An active link can be associated with only one control field (you can, however, choose any combination of screen appearances for that field). A particular control field, however, can be associated with multiple active links. While information about nondata fields is stored on the server, they do not require storage space in the underlying database.
<b>Privileges</b>	AR System administrator.
<b>Synopsis</b>	<pre>#i ncl ude "ar.h" #i ncl ude "arerrno.h" #i ncl ude "arextern.h" #i ncl ude "arstruct.h"  int ARCreateField(     ARControlStruct *control,     ARNameType schema,     *FieldDef d,     reserved1 dOK,     FieldDefName,     *FieldDefMap,     dataType,     option,     createMode,     FieldDefOption,     *defaultVal,     *permissions,     *limit,     *InstanceList,</pre>

```
char *helpText,
ARAccessNameType owner,
char *changeDictionary,
ARStatusList *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

### schema

The name of the form the field is linked to. The field must be associated with a single form that currently exists on the server.

### fieldId

The internal ID of the field to create. The IDs of all fields and form views (VUIs) associated with a given form must be unique. Specify 0 for this parameter if you want the system to generate and return the ID. Otherwise, specify a value between 536870912 and 2147483647 (limited by AR\_MAX\_RESERVED\_FIELD\_ID in arstruct.h). If you specify a reserved ID, the system generates an error unless the reservedIdOK parameter is set to 1 (TRUE) (see the *Form and Application Objects* guide for restrictions on using reserved IDs).

### reservedIdOK

A flag indicating whether you can create the field with a reserved ID. Specify 1 (TRUE) for this parameter to allow reserved IDs. Specify 0 (FALSE) if you want the system to generate an error when you specify a reserved ID for the fieldId parameter.

### fieldName

The name of the field to create. The names of all fields and VUIs associated with a given form must be unique. The system uses this name when it creates the SQL view of the form for report writing purposes. The field name is used to identify a field in workflow more easily. (See the *Form and Application Objects* guide for more information about field names.) The field name is different from the field label in that it is specific to a form, but not specific to a view of the form. See the `instanceList` parameter on page 147 for information about how to specify a field label.

## fieldMap

The underlying form in which to create the field (applicable for join forms only). If you are creating a field in a base form, specify a field type of 1 (AR\_FIELD\_REGULAR). Otherwise, specify a field type of 2 (AR\_FIELD\_JOIN), and identify the member form (primary or secondary) and field ID for the new field. If the member form is also a join form, create fields in all nested join forms until you can map the field to an underlying base form.

## dataType

The data type of the field to create.

- 2: Integer (AR\_DATA\_TYPE\_INTEGER).
- 3: Real (AR\_DATA\_TYPE\_REAL).
- 4: Character (AR\_DATA\_TYPE\_CHAR).
- 5: Diary (AR\_DATA\_TYPE\_DIARY).
- 6: Selection (AR\_DATA\_TYPE\_ENUM).
- 7: Date/time (AR\_DATA\_TYPE\_TIME).
- 10: Fixed-point decimal (AR\_DATA\_TYPE\_DECIMAL). Values must be specified in C locale, for example 1234.56
- 11: Attachment (AR\_DATA\_TYPE\_ATTACHMENT).
- 12: Currency (AR\_DATA\_TYPE\_CURRENCY).
- 31: Trim (AR\_DATA\_TYPE\_TRIM).
- 32: Control (AR\_DATA\_TYPE\_CONTROL).
- 33: Table (AR\_DATA\_TYPE\_TABLE).
- 34: Column (AR\_DATA\_TYPE\_COLUMN).
- 35: Page (AR\_DATA\_TYPE\_PAGE).
- 36: Page holder (AR\_DATA\_TYPE\_PAGE HOLDER).

## option

A flag indicating whether users must enter a value in the field.

- 1: Required (data fields only) (AR\_FIELD\_OPTION\_REQUIRED).
- 2: Optional (data fields only) (AR\_FIELD\_OPTION\_OPTIONAL).
- 4: Display-only (data fields only). Works like an optional field but doesn't write to the database (AR\_FIELD\_OPTION\_DISPLAY).

## createMode

A flag indicating the permission status for the field when users submit entries. This parameter is ignored for display-only fields.

- 1: Any user (including guest users) can enter data in the field when submitting (AR\_FI\_ELD\_OPEN\_AT\_CREATE).
- 2: Only users who have been granted permission can enter data in the field when submitting (AR\_FI\_ELD\_PROTECTED\_AT\_CREATE).

## fieldOption

A bitmask that indicates whether the field should be audited or copied when other fields are audited.

- Bit 0: Audit this field. (AR\_FI\_ELD\_BI\_TOPTI\_ON\_AUDIT)  
Bit 1: Copy this field when other fields in the form are audited. (AR\_FI\_ELD\_BI\_TOPTI\_ON\_COPY)  
Bit 2: Indicates this field is for Log Key 1. (AR\_FI\_ELD\_BI\_TOPTI\_ON\_LOG\_KEY1)  
Bit 3: Indicates this field is for Log Key 2. (AR\_FI\_ELD\_BI\_TOPTI\_ON\_LOG\_KEY2)  
Bit 2 and 3: Indicates this field is for Log Key 3. (AR\_FI\_ELD\_BI\_TOPTI\_ON\_LOG\_KEY3)

## defaultVal

The value to apply if a user submits an entry with no field value (applicable for data fields only). The default value can be as many as 255 bytes in length (limited by AR\_MAX\_DEFAULT\_SIZE) and must be of the same data type as the field. Specify NULL (or AR\_DEFAULT\_VALUE\_NONE) for trim, control, page, and page holder fields or if you do not want to define a default value.

## permissions

A list of zero or more groups who can access this field. Specifying an empty permission list defines a field accessible by users with administrator capability only. Specifying group ID 0 (Public) provides access to all users. The permission value you assign for each group determines whether users in that group can modify the field.

- 1: Users can view but not modify the field (AR\_PERMISSIONS\_VIEW).
- 2: Users can view and modify the field (AR\_PERMISSIONS\_CHANGE).

## limit

The value limits for the field and other properties specific to the field's type. See the ARFieldListStruct definition in the ar.h file to find the contained structure that applies to the type of field you are creating. The limits and properties you define must be of the same data type as the field. Specify `NULL` (or `AR_FIELD_LIST_TYPE_NONE`) for trim and control fields or if you do not want to define any limits or properties.

## dInstanceList

A list of zero or more display properties to associate with the field. You can define both display properties common to all form views (VUIs) and display properties specific to particular views. The system includes the field in each view you specify, regardless of whether you define any display properties for those views. If you do not specify a property for a particular view, the system uses the default value (if defined). Specify `NULL` for this parameter if you do not want to define any display properties.

## helpText

The help text associated with the field. This text can be of any length. Specify `NULL` for this parameter if you do not want to associate help text with this object.

## owner

The owner for the field. The owner defaults to the user performing the operation if you specify `NULL` for this parameter.

## changeDiary

The initial change diary associated with the field. This text can be of any length. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to associate change diary text with this object.

## Return values

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**Display properties** You can specify the following display properties in any order. Avoid specifying a property more than once, because the system then chooses one of your definitions at random.

---

**Note:** When using the driver program to call the functions in your API program, you will need to specify the property number instead of the property name. The property number is listed for each display property name in the ar.h file in the api/include directory. See “Using the driver program” on page 475 for more information.

---

**AR\_DPROP\_ALIGN (unsigned long):**

A value indicating the vertical alignment of text within its bounding box.

- 1: Top-aligned (AR\_DVAL\_ALIGN\_TOP).
- 2: Centered (AR\_DVAL\_ALIGN\_MIDDLE) (default setting).
- 4: Bottom-aligned (AR\_DVAL\_ALIGN\_BOTTOM).

**AR\_DPROP\_AUTO\_REFRESH (unsigned long):**

A value indicating whether automatic refresh is enabled (applicable for table fields only). Specify 0 to disable (AR\_DVAL\_AUTO\_REFRESH\_NONE) and 1 to enable (AR\_DVAL\_AUTO\_REFRESH\_TABLE\_MAX). When automatic refresh is enabled, the contents of the table field will be updated when the entry is displayed in a Modify window. When automatic refresh is disabled, the table appears empty until the user clicks within its borders to refresh it.

**AR\_DPROP\_BACKGROUND\_MODE (unsigned long):**

A value indicating whether the background of a field is opaque or transparent (applicable for trim fields only). Specify 0 for opaque (AR\_DVAL\_BKG\_MODE\_OPAQUE) or 1 for transparent (AR\_DVAL\_BKG\_MODE\_TRANSPARENT). The default value is 0 (opaque).

**AR\_DPROP\_BBOX (ARCoordList):**

A set of coordinates (X,Y) that specify the screen boundaries (bounding box) for the field (applicable for all fields except page and column fields). The first coordinate pair identifies the upper-left corner for the field. The second coordinate pair identifies the lower-right corner (see AR\_DPROP\_COORDS).

**AR\_DPROP\_BUTTON\_2D (unsigned long):**

A flag indicating whether the button appears as a two-dimensional image with no border (applicable for control fields only). Valid values for this property are 1 (TRUE) and 0 (FALSE). The default value is 0 (three-dimensional with borders).

**AR\_DPROP\_BUTTON\_IMAGE\_POSITION (unsigned long):**

A value that determines the position of the button image relative to the button label (applicable for control fields only). The default value is 0 (centered, no label).

- 0: AR\_DVAL\_IMAGE\_CENTER (centered, no label).
- 1: AR\_DVAL\_IMAGE\_LEFT (to the left of the label).
- 2: AR\_DVAL\_IMAGE\_RIGHT (to the right of the label).
- 3: AR\_DVAL\_IMAGE\_ABOVE (above the label).
- 4: AR\_DVAL\_IMAGE\_BELOW (below the label).

**AR\_DPROP\_BUTTON\_MAINTAIN\_RATIO (unsigned long):**

A flag indicating whether a scaled button image will preserve its original width-to-height ratio (applicable for control fields only). Valid values for this property are 1 (TRUE) and 0 (FALSE). The default value is F (do not preserve ratio). This property only affects the image when **AR\_DPROP\_BUTTON\_SCALE\_IMAGE** is TRUE.

**AR\_DPROP\_BUTTON\_SCALE\_IMAGE (unsigned long):**

A flag indicating whether the button image will shrink or grow to fill the space available after allocating space for the button label (applicable for control fields only). Valid values for this property are 1 (TRUE) and 0 (FALSE). The default value is F (leave image at original size).

**AR\_DPROP\_BUTTON\_TEXT (character string):**

The button label for the field (applicable for control fields only). This label can be as many as 30 bytes in length (limited by **AR\_MAX\_NAME\_SIZE**).

AR\_DPROP\_CHARFI ELD\_DI SPLAY\_TYPE:

A value indicating whether the character field is a drop-down list or an editable field.

0: Editable field (AR\_DVAL\_CHARFI ELD\_EDI T).

1: Drop-down field (AR\_DVAL\_CHARFI ELD\_DROPDOWN).

AR\_DPROP\_CNTL\_TYPE (unsigned long):

A bitmask indicating the screen appearance for a control field.

Bit 0: Button (AR\_DVAL\_CNTL\_BUTTON).

Bit 1: Menu item (AR\_DVAL\_CNTL\_MENU).

Bit 2: Toolbar button (AR\_DVAL\_CNTL\_TOOLBAR).

AR\_DPROP\_COLUMN\_ORDER (integer):

The order of a column in a table field. The column with the lowest value will be the left-most displayed.

AR\_DPROP\_COLUMN\_WIDTH (integer):

The width of a column in a table field.

AR\_DPROP\_COORDS (ARCoordList):

A set of coordinates (X,Y) that specify the vertex location (applicable for line- and box-type trim fields only). For line-type trim fields, identify the vertices of the line segment by using two coordinate pairs. For box-type trim fields, identify the corners of the box (clockwise from the upper-left corner) by using four coordinate pairs (see also AR\_DPROP\_BBOX).

To minimize display differences across a variety of operating systems and screen resolutions, the clients standardize all coordinate values before storing them in the database and then map these values to the current environment upon retrieval. When specifying (or retrieving) coordinate data for any property with ARCoordList, you must apply the same translation algorithms in order for BMC Remedy Administrator and BMC Remedy User to display the field correctly. For additional information, see “Using ARCoordList to specify field size and location” on page 67.

**AR\_DPROP\_DATA\_BBOX (ARCoordLi st):**

A set of coordinates (X,Y) that specify the screen boundaries (bounding box) for the field's data area (applicable for data fields only). For example, this property locates the entry box for a character field and the buttons for a radio button selection field. The first coordinate pair identifies the upper-left corner for the data area. The second coordinate pair identifies the lower-right corner. The coordinates specified are relative to the field's location as specified in its AR\_DPROP\_BBOX property, not to the window or page containing it.

**AR\_DPROP\_DATA\_COLS (unsigned long):**

The number of text columns to display for the field (applicable for data fields only). AR\_DPROP\_DATA\_BBOX overrides this value.

**AR\_DPROP\_DATA\_EXPAND\_BBOX (ARCoordLi st):**

A set of coordinates (X,Y) that specify the screen boundaries (bounding box) for the field's text editor button (applicable for diary and long character fields only). The first coordinate pair identifies the upper-left corner for the button. The second coordinate pair identifies the lower-right corner. The coordinates specified are relative to the field's location as specified in its AR\_DPROP\_BBOX property, not to the window.

**AR\_DPROP\_DATA\_MENU (unsigned long):**

A flag indicating whether the field has a drop-down list (applicable for selection fields only). Valid values for this property are 1 (TRUE) and 0 (FALSE). The default value is 0 (no drop-down list).

**AR\_DPROP\_DATA\_MENU\_BBOX (ARCoordLi st):**

A set of coordinates (X,Y) that specify the screen boundaries (bounding box) for the field's menu button (applicable for character fields with an attached menu only). The first coordinate pair identifies the upper-left corner for the button. The second coordinate pair identifies the lower-right corner. The coordinates specified are relative to the field's location as specified in its AR\_DPROP\_BBOX property, not to the window.

**AR\_DPROP\_DATA\_RADIO (unsigned long):**

A flag indicating whether the field uses radio option buttons (applicable for selection fields only). Valid values for this property are 1 (TRUE) and 0 (FALSE). The default value is 0 (does not use option buttons).

**AR\_DPROP\_DATA\_ROWS (unsigned long):**

The number of text rows to display for the field (applicable for data fields only).

**AR\_DPROP\_DATA\_SPI\_N (unsigned long):**

A flag indicating whether the field has a numeric spinner (applicable for integer fields only). Valid values for this property are T (TRUE) and F (FALSE). The default value is F (no numeric spinner).

**AR\_DPROP\_DATETIME\_POPUP (unsigned long):**

A value indicating the display type of a date/time field. The default value is 0 (both date and time).

- 0: Both date and time (AR\_DVAL\_DATETIME\_BOTH) (default setting).
- 1: Time only (AR\_DVAL\_DATETIME\_TIME).
- 2: Date only (AR\_DVAL\_DATETIME\_DATE).

**AR\_DPROP\_DEPTH\_EFFECT (unsigned long):**

A value indicating the three-dimensional style for the field. This property, in conjunction with the AR\_DPROP\_LINE\_WIDTH property, determines the overall look of line- and box-type trim fields.

- 0: No three-dimensional effect (AR\_DVAL\_DEPTH\_EFFECT\_FLAT).
- 1: Field appears to lie above screen surface (AR\_DVAL\_DEPTH\_EFFECT\_RAISED).
- 2: Field appears to lie below screen surface (AR\_DVAL\_DEPTH\_EFFECT\_SUNKEN).
- 4: Field appears to be etched into screen surface (AR\_DVAL\_DEPTH\_EFFECTETCHED).

**AR\_DPROP\_DISPLAY\_PARENT (unsigned long):**

A value indicating the parent field that the field will display in. Specify the field ID of the parent. A page holder field is the parent of the page fields displayed in it, and a page field is the parent of the fields displayed in it. A value of 0 indicates that the field's parent is the form view and not another field. The default value is 0.

**AR\_DPROP\_DRI\_LL\_DOWN (unsigned long):**

A bitmask indicating whether drill-down is enabled for a table field. Specify bit 0 to disable and 1 to enable. When drill-down is enabled and a user double-clicks a row in the table field, a Modify window opens for the entry represented by that row.

**AR\_DPROP\_ENABLE (unsigned long):**

A value identifying the enabled/disabled status of the field. For control fields, this setting applies to all specified screen appearances (see AR\_DPROP\_CNTL\_TYPE).

- 1: View and select only (data); disabled (trim or control) (AR\_DVAL\_ENABLE\_READ\_ONLY).
- 2: View, select, and change (data); enabled (trim or control) (AR\_DVAL\_ENABLE\_READ\_WRITE) (default setting).
- 3: Disabled (data, trim, or control) (AR\_DVAL\_ENABLE\_DISABLE).

**AR\_DPROP\_FIELD\_CUSTOMSTYLE (character string):**

Specifies a custom CSS style for a field.

**AR\_DPROP\_FIELD (character string):**

The text for a text-type trim field that includes a URL. The string must include a standard HTML anchor tag () that encloses the text to be linked and specifies the URL to link to. Text that does not fit within its bounding box causes the form window to scale up to accommodate it (see AR\_DPROP\_BBOX).

**AR\_DPROP\_HTML\_TEXT (character string):**

The text for a text-type trim field that includes a URL. The string must include a standard HTML anchor tag () that encloses the text to be linked and specifies the URL to link to. Text that does not fit within its bounding box causes the form window to scale up to accommodate it (see AR\_DPROP\_BBOX).

**AR\_DPROP\_HTML\_TEXT\_COLOR (character string):**

The color of the linked text for a text-type trim field that includes a URL (see AR\_DPROP\_HTML\_TEXT). Use one of the values listed for AR\_DPROP\_LABEL\_COLOR\_TEXT or specify a custom color as `0xBBGGRR`, a string concatenating the two-digit hexadecimal values for blue, green, and red. The default value is the link color specified in the user's browser preferences.

**AR\_DPROP\_IMAGE (ARByteLi st):**

The icon image for a toolbar-type control field. Specify a byte list type of 1 (AR\_BYTE\_LIST\_WI\_N30\_BI\_TMAP) for images in bitmap (. bmp) and DIB (. dib) format. Specify 0 (AR\_BYTE\_LIST\_SELF\_DEFINED) for all other image formats.

---

**Note:** The only image formats currently supported on toolbar control fields are bitmap and DIB. The only image size currently supported is 16 pixels wide by 15 pixels high. Images of other sizes are not scaled to fit. When creating images, use simple images and the 216-color palette for best results.

---

**AR\_DPROP\_JUSTIFY (unsigned long):**

A value indicating the horizontal justification of text within its bounding box.

- 1: Left-aligned (AR\_DVAL\_JUSTIFY\_LEFT).
- 2: Centered (AR\_DVAL\_JUSTIFY\_CENTER).
- 4: Right-aligned (AR\_DVAL\_JUSTIFY\_RIGHT).

**AR\_DPROP\_LABEL (character string):**

The screen label for the field (applicable for data fields only). This label can be as many as 30 bytes in length (limited by AR\_MAX\_NAME\_SIZE).

**AR\_DPROP\_LABEL\_BBOX (ARCoordLi st):**

A set of coordinates (X,Y) that specify the screen boundaries (bounding box) for the field label (applicable for data fields only). The first coordinate pair identifies the upper-left corner for the label. The second coordinate pair identifies the lower-right corner. The coordinates specified are relative to the field's location as specified in its AR\_DPROP\_BBOX property, not to the window or page containing it.

**AR\_DPROP\_LABEL\_COLOR\_TEXT (character string):**

The color of the field label. Specify the color as #BBGGRR, a string concatenating the two-digit hexadecimal values for blue, green, and red.

**AR\_DPROP\_LABEL\_POS\_ALI\_GN** (unsigned long):

A value indicating the vertical alignment of the label within the sector (see the Note that follows).

- 1: Top-aligned (AR\_DVAL\_ALI\_GN\_TOP).
- 2: Middle-aligned (AR\_DVAL\_ALI\_GN\_MIDDLE).
- 4: Bottom-aligned (AR\_DVAL\_ALI\_GN\_BOTTOM).

---

**Note:** The following two combinations of values for the AR\_DPROP\_LABEL\_POS\_SECTOR, AR\_DPROP\_LABEL\_POS\_JUSTIFY, and AR\_DPROP\_LABEL\_POS\_ALI\_GN properties provide the best-looking views in BMC Remedy User:

---

SECTOR	JUSTIFY	ALIGN	Label position	
north	left	bottom	>	top
west	left	top	>	left

---

**AR\_DPROP\_LABEL\_POS\_JUSTIFY** (unsigned long):

A value indicating the horizontal alignment of the label within the sector (see the Note that follows AR\_DPROP\_LABEL\_POS\_ALI\_GN).

- 1: Left-justified (AR\_DVAL\_JUSTIFY\_LEFT).
- 2: Center-justified (AR\_DVAL\_JUSTIFY\_CENTER).
- 4: Right-justified (AR\_DVAL\_JUSTIFY\_RIGHT).

**AR\_DPROP\_LABEL\_POS\_SECTOR** (BITMASK):

A value indicating the label position relative to the field. This property, in conjunction with the AR\_DPROP\_LABEL\_POS\_JUSTIFY and AR\_DPROP\_LABEL\_POS\_ALI\_GN properties, determines the specific placement and alignment of the screen label (see the Note that follows AR\_DPROP\_LABEL\_POS\_ALI\_GN).

- 0: Suppress label display (AR\_DVAL\_SECTOR\_NONE).
- 2: Above the field (AR\_DVAL\_SECTOR\_NORTH).
- 16: Left of the field (AR\_DVAL\_SECTOR\_WEST) (default setting).

**AR\_DPROP\_LI\_NE\_WI\_DTH (unsigned long):**

The line width for the field (applicable for line- and box-type trim fields only). The default value is 3 pixels.

**AR\_DPROP\_MENU\_HELP (character string):**

The help text for a menu-type control field. This text can be as many as 30 bytes in length (limited by AR\_MAX\_NAME\_SIZE).

**AR\_DPROP\_MENU\_MODE (unsigned long):**

A value indicating the type of menu item (applicable for control fields only).

- 0: Leaf-level menu item (AR\_DVAL\_CNTL\_ITEM).
- 2: Separator item (AR\_DVAL\_CNTL\_SEPARATOR).
- 5: Top- or intermediate-level menu item (AR\_DVAL\_CNTL\_A\_MENU).

**AR\_DPROP\_MENU\_PARENT (unsigned long):**

A value identifying the parent menu item for the field (applicable for control fields only). Specify the field ID of the parent control field for leaf- or intermediate-level menu items. Specify 0 for top-level menu items.

**AR\_DPROP\_MENU\_POS (unsigned long):**

A value that determines the menu position for the field (applicable for control fields only). For leaf- or intermediate-level menu items, the value associated with each field in a view (VUI) determines the top-to-bottom order of the items in the menu (from lowest to highest, starting with one). For top-level menu items, this value determines the left-to-right order of the items in the menu bar.

**AR\_DPROP\_MENU\_TEXT (character string):**

The menu item text for the field (applicable for control fields only). This text can be as many as 30 bytes in length (limited by AR\_MAX\_NAME\_SIZE).

**AR\_DPROP\_NAVBAR\_INITIAL\_SELECTED\_ITEM (field ID):**

Specifies that when you open a form, the navigation bar is initialized with the selected item.

---

**AR\_DPROP\_NAVBAR\_WORKFLOW\_ON\_SELECTED\_ITEM (integer):**

Specifies whether workflow is executed when the item is selected.  
Values are 1 (TRUE, execute workflow on selection) or 0 (FALSE, select the item but do not execute workflow).

**AR\_DPROP\_NAVBAR\_SELECT\_ITEM\_ON\_CLICK (integer):**

Specifies whether to move selection to an item when that item is clicked. Values are 1 (TRUE, move selection on click) or 0 (FALSE, do nothing on click).

**AR\_DPROP\_NONE:**

Indicates the NULL display property. You can specify no display properties by passing an empty property list. This property will be ignored if the list also contains other display properties.

**AR\_DPROP\_PUSH\_BUTTON\_IMAGE (ARByteList):**

The icon image for a button. Specify a byte list type of 1 (AR\_BYTE\_LIST\_WIN30\_BITMAP) for images in bitmap (.bmp) and DIB (.dib) format. Specify 2 (AR\_BYTE\_LIST\_JPEG) for JPEG (.jpg or .jpeg) format.

---

**Note:** The only image formats currently supported on button fields are JPEG, bitmap, and DIB. There is no limit to the pixel dimensions or file size of a button image, but large images can slow down your system's performance. When creating images, use simple images and the 216-color Web palette for best results.

---

**AR\_DPROP\_SORT\_DIR (integer):**

A value indicating the sort direction of a column in a table field. Specify 0 for ascending (AR\_DVAL\_SORT\_DIR\_ASCENDING) and 1 for descending (AR\_DVAL\_SORT\_DIR\_DESCENDING).

**AR\_DPROP\_SORT\_SEQ (integer):**

The sort order of a column in a table field. The system will sort based on the column with the lowest value first. To keep from sorting based on a column, specify 0 (the default).

**AR\_DPROP\_TAB\_ORDER (unsigned long):**

A value that determines the field tab order (from lowest to highest).

**AR\_DPROP\_TABLE\_DI\_SPLAY\_TYPE (integer):**

Dictates the appearance of a table field in a browser.

- 0: Standard table (AR\_DVAL\_TABLE\_DI\_SPLAY\_TABLE).
- 1: Results list table (AR\_DVAL\_TABLE\_DI\_SPLAY\_RESULTS\_LIST).
- 2: Notification alert table (AR\_DVAL\_TABLE\_DI\_SPLAY\_NOTIFICATION).
- 3: Display from a single source table (AR\_DVAL\_TABLE\_DI\_SPLAY\_SINGLE\_TABLE\_TREE).

**AR\_DPROP\_TEXT (character string):**

The text for a text-type trim field. Text that does not fit within its bounding box causes the form window to scale up to accommodate it (see AR\_DPROP\_BBOX).

**AR\_DPROP\_TEXT\_FONT\_STYLE (character string):**

The name of the font style to use for a text-type trim field.

Editor:	Font used for data fields.
Optional:	Font used for optional data field labels.
PushButton:	Font used for button labels.
System:	Font used for system-generated data field labels.
RadioButton:	Font used for option button choices.
Required:	Font used for required data field labels.
Header1:	Font used for titles.
Header2:	Font used for major headings.
Header3:	Font used for minor headings.
Note:	Font used for notes.
Detail:	Font used for detail information.

**AR\_DPROP\_TOOLBAR\_MODE (unsigned long):**

A value indicating the type of toolbar item (applicable for control fields only).

- 0: Toolbar button (AR\_DVAL\_CNTL\_ITEM).
- 2: Toolbar separator (AR\_DVAL\_CNTL\_SEPARATOR).

**AR\_DPROP\_TOOLBAR\_POS (unsigned long):**

A value that determines the toolbar position for the field (applicable for control fields only). The value associated with each field in a view (VUI) determines the left-to-right order of the icons in the toolbar (from lowest to highest, starting with one).

**AR\_DPROP\_TOOLTIP (character string):**

The tooltip text for a toolbar-type control field. This text can be as many as 30 bytes in length (limited by AR\_MAX\_NAME\_SIZE).

**AR\_DPROP\_TRIM\_TYPE (unsigned long):**

A value indicating the type of trim field.

- 1: Horizontal or vertical line (AR\_DVAL\_TRIM\_LINE).
- 2: Box (rectangle) (AR\_DVAL\_TRIM\_SHAPE).
- 3: One or more rows of text (AR\_DVAL\_TRIM\_TEXT).

**AR\_DPROP\_VIEWFIELD\_ELD\_BORDERS**

A value indicating whether borders are hidden or shown for view fields.

- 0: For native clients, the borders are on and for web clients, the borders are shown based on the display (AR\_DVAL\_VIEWFIELD\_ELD\_BORDERS\_DEFAULT).
- 1: No borders on view fields (AR\_DVAL\_VIEWFIELD\_ELD\_BORDERS\_NONE).
- 2: Borders on view fields (AR\_DVAL\_VIEWFIELD\_ELD\_BORDERS\_ENABLE).

**AR\_DPROP\_VI\_EWFI\_ELD\_SCROLLBARS**

A value indicating whether scroll bars are hidden or shown for view fields.

- 0: Add scroll bars if needed. (AR\_DVAL\_VI\_EWFI\_ELD\_SCROLLBARS\_AUTO).
- 1: Always show scroll bars (AR\_DVAL\_VI\_EWFI\_ELD\_SCROLLBARS\_ON).
- 2: Hide scroll bars; content might be clipped. (AR\_DVAL\_VI\_EWFI\_ELD\_SCROLLBARS\_HIDDEN).

**AR\_DPROP\_VI\_SI\_BLE (ENUM):**

A flag indicating the visibility of the field. Valid values for this property are 0 (hidden) and 1 (visible). The default value is 1 (visible). For control fields, this setting applies to all specified screen appearances (see AR\_DPROP\_CNTL\_TYPE).

**AR\_DPROP\_Z\_ORDER (unsigned long):**

A value that determines the field drawing order. The value associated with each field in a view (VUI) determines the back-to-front layering of the fields on the screen (from lowest to highest). The Z-order values of all fields associated with a given view must be unique. If two or more fields have duplicate Z-order values, the system will order them at random.

---

**Note:** BMC Remedy User will display table fields, page holder fields, and buttons with images in front of other fields regardless of their Z-order.

---

**See also** ARCreateVUI, ARDeleteField, ARGetField, ARGetMultipleFields, ARGetListField, ARSetField. See FreeAR for: FreeARFieldLimitList, FreeARPermissionList, FreeARStatusList, FreeARValueStruct.

## ARCreateFilter

**Description** Creates a new filter with the indicated name on the specified server. The filter takes effect immediately and remains in effect until changed or deleted.

**Privileges** AR System administrator.

**Synopsis**

```
#i ncl ude "ar. h"
#i ncl ude "arerrno. h"
#i ncl ude "arextern. h"
#i ncl ude "arstruct. h"

int ARCreateFilter(
    ARControlStruct *control,
    ARNameType name,
    unsigned int order,
    ARWorkflowConnectStruct *schemaList,
    unsigned int opSet,
    unsigned int enable,
    ARQualifierStruct *query,
    ARFilterActionList *actionList,
    ARFilterActionList *elseList,
    char *helpText,
    ARAccessNameType owner,
    *changeDictionary,
```

ARPropLi st ARStatusLi st	*obj PropLi st, *status)
------------------------------	-----------------------------

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

### name

The name of the filter to create. The names of all filters on a given server must be unique.

### order

A value between 0 and 1000 (inclusive) that determines the filter execution order. When multiple filters are associated with a form, the value associated with each filter determines the order in which they are processed (from lowest to highest).

### schemaList

The list of form names the filter is linked to. The filter must be associated with a single form or a list of forms that currently exist on the server.

### opSet

A bitmask indicating the form operations that trigger the filter.

- Bit 0: Execute the filter on get operations (AR\_OPERATION\_GET).
- Bit 1: Execute the filter on set operations (AR\_OPERATION\_SET).
- Bit 2: Execute the filter on create operations (AR\_OPERATION\_CREATE).
- Bit 3: Execute the filter on delete operations (AR\_OPERATION\_DELETE).
- Bit 4: Execute the filter on merge operations (AR\_OPERATION\_MERGE).

### enable

A flag to enable or disable this filter. A value of 0 disables the filter, causing its condition checks and associated actions to not be performed. A value of 1 enables the filter, causing its conditions to be checked for each form operation specified by the `opSet` parameter.

## query

A qualification that determines whether the filter is executed. Specify `NULL` or assign an operation value of 0 (`AR_COND_OP_NONE`) to execute the filter unconditionally.

## actionList

The set of actions performed if the condition defined by the `query` parameter is satisfied. You can specify from 1 to 25 actions in this list (limited by `AR_MAX_ACTIONS`).

## elseList

The set of actions performed if the condition defined by the `query` parameter is not satisfied. You can specify from 0 to 25 actions in this list (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter (or zero actions) if you do not want to define any else actions.

## helpText

The help text associated with the filter. This text can be of any length. Specify `NULL` for this parameter if you do not want to associate help text with this object.

## owner

The owner for the filter. The owner defaults to the user performing the operation if you specify `NULL` for this parameter.

## changeDiary

The initial change diary associated with the filter. This text can be of any length. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to associate change diary text with this object.

## objPropList

If the `objPropList` parameter is set to `NULL`, an object properties list with zero properties will be associated with the object, and when the filter definition is retrieved, zero properties are returned. See “Server object property tags” on page 79 for object property tag names and data types.

## Return values

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARDeleteFilter, ARGetFilter, ARGetListFilter, ARSetFilter. See FreeAR for: FreeARFilterActionList, FreeARPropList, FreeARQualifierStruct, FreeARStatusList.

## ARCreateLicense

**Description** This call adds an entry to the license file for the current server.

**Privileges** All users.

**Synopsis**

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

int ARCreateLicense(
    ARControlStruct *control,
    ARLicenseInfoStruct *licenseInfo,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**licenseInfo**

New license entry information, such as license key, license type, expiration date, and the license host for the new license.

**Return values**

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

ARGetListLicense, ARDeleteLicense, ARValidateLicense, ARValidateMultipleLicenses. See FreeAR for: FreeARStatusList, FreeARLicenseInfoStruct.

# ARCreateSchema

**Description** Creates a new form with the indicated name on the specified server. The nine required core fields are automatically associated with the new form.

**Privileges** AR System administrator.

**Synopsis**

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

int ARCreateSchema(
    ARControlStruct *control,
    const char *name,
    ARCompoundSchema *schema,
    ARSchemaList *schemas,
    ARPermList *groupList,
    ARInternalList *adminGroupList,
    AREntryList *fields,
    ARSortList *sortList,
    ARIndexList *indexList,
    ARArchivedInfoStruct *archivedInfo,
    ARAuditInfoStruct *auditInfo,
    ARNameType *defaultVui,
    const char *helpText,
    ARAccessNameType *owner,
    ARPropList *objProps,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**name**

The name of the form to create. The names of all forms on a given server must be unique.

## schema

The type of form to create. The information contained in this definition depends on the form type you specify.

- 1: Indicates a base form (AR\_SCHEMA\_REGULAR). All other structure information is ignored if you specify this type.
- 2: Indicates a join form (AR\_SCHEMA\_JOIN). If you specify this type you must identify the underlying member forms and how to join them.
- 3: Indicates a view form (AR\_SCHEMA\_VIEW). You must identify the underlying table and key field names if you specify this type.
- 4: Indicates a display-only form (AR\_SCHEMA\_DISPLAY). The `getListFields`, `sortList`, and `indexList` parameters are ignored if you specify this type.
- 5: Indicates a vendor form (AR\_SCHEMA\_VENDOR). You must identify the underlying vendor and table names if you specify this type.

## schemaInheritanceList

This is reserved for future use. Specify `NULL`.

## groupList

A list of zero or more groups who can access this form. Specifying an empty group list defines a form accessible by users with administrator capability only. Specifying group ID 0 (Public) provides access to all users. The permission value you assign for each group determines whether users in that group see the form in the form list:

- 1: Users see the form in the form list (AR\_PERMISSIONS\_VISIBLE).
- 2: Users do not see the form in the form list (AR\_PERMISSIONS\_HIDDEN).

## adminingrpList

A list of zero or more groups who can administer this form (and the associated filters, escalations, and active links). Users must belong to both a specified group *and* the Subadministrator group to obtain these privileges. Specifying an empty administrator group list defines a form that can be administered by users with administrator capability only. Specifying group ID 0 (Public) provides subadministrator capability to all members of the Subadministrator group.

## getListFields

A list of zero or more fields that identifies the default query list data for retrieving form entries. The list can include any data fields except diary fields and long character fields. The combined length of all specified fields, including separator characters, can be as many as 128 bytes (limited by AR\_MAX\_SDESC\_SIZE). The query list displays the Short-Description core field if you specify `NULL` for this parameter (or zero fields). Specifying a `getListFields` argument when calling the `ARGetListEntry` function overrides the default query list data.

## sortList

A list of zero or more fields that identifies the default sort order for retrieving form entries. Specifying a `sortList` argument when calling the `ARGetListEntry` function overrides the default sort order.

## indexList

The set of zero or more indexes to create for the form. You can specify from 1 to 16 fields for each index (limited by AR\_MAX\_INDEX\_FIELDS). Diary fields and character fields larger than 255 bytes cannot be indexed.

## archiveInfo

If a form is to be archived, this is the archive information for the form. Specify `NULL` for this parameter if you do not want to create or set the archive information. These are the values for archive type:

- 0: Deletes the archive settings for the selected form. The selected form will not be archived (AR\_ARCHIVE\_NONE).
- 1: Entries on the selected form are copied to the archive form (AR\_ARCHIVE\_FORM).
- 2: Entries on the selected form are deleted from the source form (AR\_ARCHIVE\_DELETE).
- 32: Attachment fields are not copied to the archive form. (AR\_ARCHIVE\_NO\_ATTACHMENTS).
- 64: Diary fields are not copied to the archive form. (AR\_ARCHIVE\_NO\_DIARY).

In addition to the archive type, `archiveInfo` also stores the form name (if archive type is AR\_ARCHIVE\_FORM), time to trigger the archive, and the archive qualification criteria. For an archive form, only the name of the source form is maintained, and none of this archive information can be set.

## auditInfo

If a form is to be audited, this is the audit information for the form. Specify `NULL` for this parameter if you do not want to create or set the audit information. These are the values for audit type:

- 0: The selected form is not to be audited. (`AR_AUDIT_NONE`).
- 1: Audit fields and copy fields on the selected form are copied to the audit shadow form (`AR_AUDIT_COPY`).
- 2: Audit fields and copy fields on the selected form are copied to the audit log form (`AR_AUDIT_LOG`).

In addition to the audit type, `auditInfo` also stores the form name (if audit type is `AR_AUDIT_COPY` or `AR_AUDIT_LOG`) and the audit qualification criteria.

## defaultVui

The label for the default view.

## helpText

The help text associated with the form. This text can be of any length. Specify `NULL` for this parameter if you do not want to associate help text with this object.

## owner

The owner for the form. The owner defaults to the user performing the operation if you specify `NULL` for this parameter.

## changeDiary

The initial change diary associated with the form. This text can be of any length. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to associate change diary text with this object.

## objPropList

If `objPropList` parameter is set to `NULL`, an object properties list with zero properties will be associated with the form, and a list of zero properties is returned when an `ARGetSchema` is performed. See “Server object property tags” on page 79 for object property tag names and data types.

**Return values** [status](#)

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateField, ARDeleteSchema, ARGetSchema, ARGetListEntry, ARGetListAlertUser, ARSetField, ARSetSchema. See FreeAR for: FreeARCompoundSchema, FreeAREntryListFieldList, FreeARIndexList, FreeARInternalIdList, FreeARPermissionList, FreeARPropList, FreeARSortList, FreeARStatusList.

## ARCreateSupportFile

**Description** Creates a file that clients can retrieve by using the AR System. Such files are commonly used for reports (to store them separately from the active link that calls them, preventing large downloads of unneeded information), but this function can store any file on the server. Each support file is associated with a server object.

**Privileges** AR System administrator.

**Synopsis**

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

int ARCreateSupportFile(
    ARControlStruct          *control,
    unsigned int               fileType,
    ARNameType                name,
    ARIinternalId             id2,
    ARIinternalId             fileId,
    FILE                      *filePtr,
    ARStatusList              *status)
```

**Input arguments**

**control**

The control record for the operations. It contains information about the user requesting the operation, where the operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

## fileType

The numerical value for the type of file, and the type of object the file is related to. Specify 1 (AR\_SUPPORT\_FILE\_EXTERNAL\_REPORT) for an external report file associated with an active link.

## name

The name of the object the file is associated with, usually a form.

## id2

The ID of the field or VUI, if the object is a form. If the object is not a form, set this parameter to 0.

## fileId

The unique identifier of a file within its object.

## filePtr

A pointer to the support file to be created in the system. If using Windows, you must open the file in binary mode.

## Return values

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## See also

ARCreatAlertEvent, ARDeleteActiveLink, ARDeleteSupportFile, ARGetActiveLink, ARGetListSupportFile, ARGetSupportFile, ARSetActiveLink, ARSetSupportFile.

# ARCreatVUI

Description	Creates a new form view (VUI) with the indicated name on the specified server.
Privileges	AR System administrator.

**Synopsis**

```
#i ncl ude "ar.h"
#i ncl ude "arerrno.h"
#i ncl ude "arextern.h"
#i ncl ude "arstruct.h"

int ARCreateVUI (
    ARControlStruct *control,
    ARNameType schema,
    ARInternalId *vuiId,
    ARNameType vuiName,
    ARLocalEntityType locale,
    unsigned int vuiType,
    ARPropList *dPropList,
    char *helpText,
    ARAccessNameType owner,
    char *changeDictionary,
    ARStatusList *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**schema**

The name of the form the VUI is linked to. The VUI must be associated with a single form that currently exists on the server.

**vuid**

The internal ID of the VUI to create. The IDs of all VUIs and fields associated with a given form must be unique. Specify 0 for this parameter if you want the system to generate and return the ID. Otherwise, specify a value between 536870912 and 2147483647 (limited by AR\_MAX\_RESERVED\_FIELD\_ID in arstruct.h).

**vuiName**

The name of the VUI to create. The names of all VUIs and fields associated with a given form must be unique.

**locale**

The locale of the VUI.

## vuiType

The type of VUI. Specify `NULL` for this parameter if you do not want to retrieve the VUI type.

- 0: No VUI type (`AR_VUI_TYPE_NONE`).
- 1: Windows type (`AR_VUI_TYPE_WINDOWS`) - fields in BMC Remedy User.
- 2: Web relative type (`AR_VUI_TYPE_WEB`) - fields in view can be adjusted.
- 3: Web absolute type (`AR_VUI_TYPE_WEB_ABS_POS`) - fields in view are fixed.

## dPropList

A list of zero or more display properties to associate with the VUI (see the Display Properties section that follows). Specify `NULL` for this parameter if you do not want to define any display properties.

## helpText

The help text associated with the VUI. This text can be of any length. Specify `NULL` for this parameter if you do not want to associate help text with this object.

## owner

The owner for the VUI. The owner defaults to the user performing the operation if you specify `NULL` for this parameter.

## changeDiary

The initial change diary associated with the VUI. This text can be of any length. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to associate change diary text with this object.

## Return values

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## Display properties

### AR\_DPROP\_VUI\_DEFAULT (unsigned long):

A flag identifying the default view (VUI) for the form. A value of 1 identifies the default VUI (with all other VUIs having a value of 0). The system uses the first VUI returned by the `ARGetListVUI` function if you do not specify a default view.

**AR\_DPROP\_PANE\_LAYOUT (integer):**

A value indicating how the panes of the VUI are arranged. This property also specifies whether the layout is locked to prevent the user from changing it.

- 1: Results pane left of details, prompt pane on top, locked.
- 2: Details pane left of results, prompt pane on top, locked.
- 3: Results pane above details, prompt pane on top, locked.
- 4: Details pane above results, prompt pane on top, locked.
- 5: Results pane left of details, prompt pane on bottom, locked.
- 6: Details pane left of results, prompt pane on bottom, locked.
- 7: Results pane above details, prompt pane on bottom, locked.
- 8: Details pane above results, prompt pane on bottom, locked.
- 1: Results pane left of details, prompt pane on top, unlocked.
- 2: Details pane left of results, prompt pane on top, unlocked.
- 3: Results pane above details, prompt pane on top, unlocked.
- 4: Details pane above results, prompt pane on top, unlocked.
- 5: Results pane left of details, prompt pane on bottom, unlocked.
- 6: Details pane left of results, prompt pane on bottom, unlocked.
- 7: Results pane above details, prompt pane on bottom, unlocked.
- 8: Details pane above results, prompt pane on bottom, unlocked.

**AR\_DPROP\_DETAI L\_PANE\_COLOR (character string):**

The background color of the details pane. Specify the color as #BBGGRR, a string concatenating the two-digit hexadecimal values for blue, green, and red.

**AR\_DPROP\_DETAI L\_PANE\_I MAGE (ARByteLi st):**

The background image for the details pane. Specify a byte list type of 1 (AR\_BYTE\_LI ST\_WI N30\_BI TMAP) for images in bitmap (. bmp) and DIB (. dib) format. Specify 2 (AR\_BYTE\_LI ST\_JPEG) for JPEG (. jpg or . jpeg) format (see AR\_DPROP\_PUSH\_BUTTON\_I MAGE).

**AR\_DPROP\_IMAGE\_ALI GN (unsigned long):**

A value indicating the vertical alignment of the background image for the details pane.

- 1: Top-aligned (AR\_DVAL\_ALI GN\_TOP).
- 2: Centered (AR\_DVAL\_ALI GN\_MIDDLE) (default setting).
- 3: Expand to fill (AR\_DVAL\_ALI GN\_FILL).
- 4: Bottom-aligned (AR\_DVAL\_ALI GN\_BOTTOM).
- 5: Tile to fill (AR\_DVAL\_ALI GN\_TILE).

**AR\_DPROP\_IMAGE\_JUSTIFY (unsigned long):**

A value indicating the horizontal justification of the background image for the details pane.

- 1: Left-aligned (AR\_DVAL\_JUSTIFY\_LEFT).
- 2: Centered (AR\_DVAL\_JUSTIFY\_CENTER).
- 3: Expand to fill (AR\_DVAL\_JUSTIFY\_FILL).
- 4: Right-aligned (AR\_DVAL\_JUSTIFY\_RIGHT).
- 5: Tile to fill (AR\_DVAL\_JUSTIFY\_TILE).

**AR\_DPROP\_TITLE\_BAR\_ICON\_IMAGE (ARByteList):**

The icon image for the title bar. Specify a byte list type of 1 (AR\_BYTE\_LIST\_WIN30\_BITMAP) for images in bitmap (.bmp) and DIB (.dib) format. Specify 2 (AR\_BYTE\_LIST\_JPEG) for JPEG (.jpg or .jpeg) format.

**AR\_DPROP\_DETAIL\_PANE\_WIDTH (integer):**

The width of the details pane, in form coordinates. Negative values are reserved for future use.

**AR\_DPROP\_DETAIL\_PANE\_HEIGHT (integer):**

The height of the details pane, in form coordinates.

**AR\_DPROP\_DETAIL\_PANE\_VISIBLE (unsigned long):**

A value indicating whether the details pane is visible. Specify 0 for invisible, 1 for visible.

**AR\_DPROP\_RESULT\_BANNER\_VISIBLE (unsigned long):**

A value indicating whether the results pane is visible. Specify 0 for invisible, 1 for visible.

**AR\_DPROP\_ALI\_AS\_SI\_NGULAR (character string):**

The singular form of the name for the logical items represented by entries in the form this view is associated with (such as help desk calls). BMC Remedy User displays this name instead of the default “request” when referring to form entries.

**AR\_DPROP\_ALI\_AS\_PLURAL (character string):**

The plural form of the name for the logical items represented by entries in the form this view is associated with (see AR\_DPROP\_ALI\_AS\_SI\_NGULAR).

**AR\_DPROP\_ALI\_AS\_SHORT\_SI\_NGULAR (character string):**

A short, singular form of the name for the logical items represented by entries in the form this view is associated with (see AR\_DPROP\_ALI\_AS\_SI\_NGULAR).

**AR\_DPROP\_ALI\_AS\_SHORT\_PLURAL (character string):**

A short, plural form of the name for the logical items represented by entries in the form this view is associated with (see AR\_DPROP\_ALI\_AS\_SI\_NGULAR).

**AR\_DPROP\_ALI\_AS\_ABBREV\_SI\_NGULAR (character string):**

A very short, singular form of the name for the logical items represented by entries in the form this view is associated with (see AR\_DPROP\_ALI\_AS\_SI\_NGULAR).

**AR\_DPROP\_ALI\_AS\_ABBREV\_PLURAL (character string):**

A very short, plural form of the name for the logical items represented by entries in the form this view is associated with (see AR\_DPROP\_ALI\_AS\_SI\_NGULAR).

**AR\_DPROP\_NAMED\_SEARCHES (character string):**

A list of administrator-defined searches. It is encoded in the same format as qualifications in .arf files. This is not supported for customer use.

**AR\_DPROP\_MENU\_ACCESS (character string):**

A list of client interface items an administrator can control. It is encoded in the same format as qualifications in .arf files. This is not supported for customer use.

AR\_DPROP\_QUERY\_LIST\_COLOR (character string):

The name of the field whose value for a given entry controls the display color of that entry in a query results list, and a list of field values and their corresponding colors. This is not supported for customer use.

**See also** ARCreateField, ARDeleteVUI, ARGetVUI, ARGetListVUI, ARSetVUI. See FreeAR for: FreeARPropList, FreeARStatusList.

## ARDateToJulianDate

**Description** Converts a year, month, and day value to a Julian date. The Julian date is the number of days since noon, Universal Time, on January 1, 4713 BCE (on the Julian calendar). The changeover from the Julian calendar to the Gregorian calendar occurred in October, 1582. The Julian calendar is used for dates on or before October 4, 1582. The Gregorian calendar is used for dates on or after October 15, 1582.

**Privileges** All users.

**Synopsis**

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

int ARDateToJulianDate(
    ARControlStruct *control,
    ARDateStruct *date,
    int *jd,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**date**

The date structure holding the year, month, day value to convert.

**Return values**

**jd**

The resulting julian date value.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

See also ARJulianDateToDate.

# ARDecodeAlertMessage

**Description** Decodes a formatted alert message and returns the component parts of the message to the alert client.

**Privileges** All users.

### Synopsis

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

```
int ARDecodeAlertMessage(
    ARControlStruct *control,
    unsigned char *message,
    unsigned int messageLen,
    ARTimestamp *timestamp,
    *sourceType,
    *priority,
    **alertText,
    **sourceTag,
    **serverName,
    **serverAddr,
    **formName,
    **objectID,
    *status)
```

**Input arguments**

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

## message

An alert message buffer received by the client via its sockets communication with the server. The buffer is not null-terminated.

The method for receiving the message from the socket is as follows:

Read at least the first 8 bytes that comprise the message header. The first 4 bytes are a message header identifier and the second 4 bytes are the length of the entire message in network byte order. Using the `ntohl()` system function, convert the 4-byte message length into the host integer format. Using the length, determine if the entire message has been received. If it has not, continue to receive data on the socket until the entire length of the message has arrived. Pass the entire message buffer, including the header, as this parameter.

## messageLen

The byte length of the alert message buffer supplied in the message parameter.

## Return values

### timestamp

The time that the alert event or indicator was created. Specify `NULL` for this parameter if you do not want to retrieve the time stamp.

### sourceType

A value that identifies the source of the alert. The possible values are:

- 1: General purpose alert event (`AR_NOTIF_SOURCE_GP`).
- 2: BMC Remedy Alert event (`AR_NOTIF_SOURCE_AR`).
- 3: First contact indicator (`AR_NOTIF_SOURCE_FIRST`).
- 4: Accessibility check indicator (`AR_NOTIF_SOURCE_CHECK`).
- 5: Dashboards alert event (`AR_NOTIF_SOURCE_FB`).

Specify `NULL` if you do not want to return the source type.

### priority

A relative value that represents the priority for this alert. The range of acceptable values is between 0 and 10. Specify `NULL` if you do not want to return the priority.

### alertText

The text that the alert contains. Specify `NULL` if you do not want to return the text. If you do not specify `NULL`, you must free the returned memory buffer.

### sourceTag

A string that identifies the source of the alert. Specify `NULL` if you do not want to return the source tag. If you do not specify `NULL`, you must free the returned memory buffer.

### serverName

The name of the server that is the source of the alert. A value of `@` indicates the current server. Specify `NULL` if you do not want to return the server name. If you do not specify `NULL`, you must free the returned memory buffer.

### serverAddr

The server IP address that the client expects. This is usually the IP address of the server named in the `serverName` parameter, except in the case where a load balancer works with the AR System server or there are multiple servers accessing the same database. Specify `NULL` if you do not want to return the server address. If you do not specify `NULL`, you must free the returned memory buffer.

### formName

The name of the form that is the source of the alert. For Flashboards, this is the name of the Flashboard that generated the alert. Specify `NULL` if you do not want to return this value. If you do not specify `NULL`, you must free the returned memory buffer.

### objectId

The identifier for the object. For AR System, this value is a special Entry ID format. For a join form, the AR System server concatenates all entry IDs and precedes them with a 2 character count (02000000000012345000000000023456).

For non-join forms, the AR System server precedes the entry IDs with a 2 character count (0100000000098765). For Flashboards, this value is the name of the Flashboard alert. Specify `NULL` if you do not want to return this value. If you do not specify `NULL`, you must free the returned memory buffer.

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateAlertEvent. See FreeAR for: FreeARStatusList.

# ARDecodeARAssignStruct

**Description** Converts a serialized assign string in a DEF file into an ARAssignStruct structure to facilitate string import.

**Privileges** All users.

**Synopsis**

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

```
int ARDecodeARAssignStruct(
    ARControlStruct *control,
    char *assignText,
    ARAssignStruct *assignStruct,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is performed, and what session performs it. The user and server fields are required.

**assignText**

The serialized assign string to be converted.

**Return values**

**assignStruct**

The resulting ARAssignStruct structure.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

ARDecodeARQualifierStruct, ARDecodeDiary, ARDecodeStatusHistory, AREncodeARAssignStruct, AREncodeARQualifierStruct, AREncodeDiary, AREncodeStatusHistory. See FreeAR for: FreeARStatusList.

# ARDecodeARQualifierStruct

**Description** Converts a serialized qualifier string into an ARQualifierStruct structure.

**Privileges** All users.

**Synopsis**

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

int ARDecodeARQualifierStruct(
    ARControlStruct *control,
    char *qualText,
    ARQualifierStruct *qualStruct,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is performed, and what session performs it. The user and server fields are required.

**qualText**

The serialized qualification string to be converted.

**Return values**

**qualStruct**

The resulting ARQualifierStruct structure.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

ARDecodeARAssignStruct, AREncodeARAssignStruct,  
 AREncodeARQualifierStruct, ARDecodeDiary, AREncodeDiary,  
 ARDecodeStatusHistory, AREncodeStatusHistory. See FreeAR for:  
 FreeARStatusList.

# ARDecodeDiary

<b>Description</b>	Parses any diary field (including the changeDiary associated with every AR System object) into user, time stamp, and text components. The function takes the formatted string returned for all diary fields and decodes it into an array of diary entries.
<b>Privileges</b>	All users.
<b>Synopsis</b>	<pre>#include "ar.h" #include "arerrno.h" #include "arextern.h" #include "arstruct.h"  int ARDecodeDiary(     ARControlStruct *control,     char *diaryString,     ARDiaryList *diaryList,     ARStatusList *status)</pre>
<b>Input arguments</b>	<p><b>control</b>  The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.</p> <p><b>diaryString</b>  A formatted diary string returned by ARGetEntry or any of the ARGet&lt;object&gt; API functions.</p>
<b>Return values</b>	<p><b>diaryList</b>  An array of decoded diary entries.</p> <p><b>status</b>  A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the <i>Error Messages</i> guide.</p>
<b>See also</b>	ARDecodeStatusHistory, ARGetActiveLink, ARGetCharMenu, ARGetEntry, ARGetField, ARGetFilter, ARGetSchema, ARGetVUI. See FreeAR for: FreeARDiaryList, FreeARStatusList, AREncodeDiary.

# ARDecodeStatusHistory

**Description** Parses the Status History core field into user and time stamp components. The function takes the formatted string returned for this field and decodes it into an array of status history entries.

**Privileges** All users.

## Synopsis

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

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

**Input arguments**

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

### statHistString

A formatted status history string returned by ARGetEntry.

**Return values**

### statHistList

An array of decoded status history entries.

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

ARDecodeDiary, ARGetEntry. See FreeAR for: FreeARStatusList, FreeARStatusHistoryList.

# ARDeleteActiveLink

**Description** Deletes the active link with the indicated name from the specified server and deletes any container references to the active link. The active link is deleted from the server immediately and is not returned to users who request information about active links. Because active links operate on clients, individual clients can continue using the active link until they reconnect to the form (thus reloading the form from the server).

**Privileges** AR System administrator.

**Synopsis**

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

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

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**name**

The name of the active link to delete.

**deleteOption**

A bitmask indicating the action to take.

Bit 0: Server object default delete option. For locked objects, fails the delete attempt (AR\_DEFAULT\_DELETE\_OPTION).

Bit 2: Delete active link and all objects in the block that are locked with the same key. This is applicable only for locked objects (AR\_LOCK\_BLOCK\_DELETE).

**Return values** [status](#)

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateAlertEvent, ARDeleteSchema, ARGetActiveLink, ARGetListActiveLink, ARGetMultipleActiveLinks, ARSetActiveLink. See FreeAR for: FreeARStatusList.

## ARDeleteCharMenu

**Description** Deletes the character menu with the indicated name from the specified server. The character menu is deleted from the server immediately and is not returned to users who request information about character menus. Because character menus operate on clients, individual clients can continue using the character menu until they reconnect to the form (thus reloading the form from the server).

**Privileges** AR System administrator.

**Synopsis**

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

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

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**name**

The name of the character menu to delete.

## deleteOption

A bitmask indicating the action to take.

- Bit 0: Server object default delete option. For locked objects, fails the delete attempt (AR\_DEFAULT\_DELETE\_OPTION).
- Bit 2: Delete character menu and all objects in the block (AR\_LOCK\_BLOCK\_DELETE).

### Return values

#### **status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

### See also

ARCreatCharMenu, ARGetCharMenu, ARGetListCharMenu, ARSetCharMenu. See FreeAR for: FreeARStatusList, ARExpandCharMenu.

## ARDelContainer

**Description** Deletes the container with the indicated name from the specified server and deletes any references to the container from other containers. Objects referenced by the container are not deleted.

**Privileges** AR System administrator.

### Synopsis

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

int ARDeleteContainer(
    ARControlStruct *control,
    ARNameType name,
    unsigned int deleteoption,
    ARStatusList *status)
```

**Input arguments**

### **control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**name**

The name of the container to delete.

**deleteOption**

A bitmask indicating the action to take.

Bit 0: Server object default delete option. For locked objects, fails the delete attempt (AR\_DEFAULT\_DELETE\_OPTION).

Bit 2: Delete container and all objects in the block (AR\_LOCK\_BLOCK\_DELETE).

**Return values****status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateContainer, ARGetContainer, ARGetListContainer, ARSetContainer. See FreeAR for: FreeARStatusList.

## ARDeleteEntry

**Description** Deletes the form entry with the indicated ID from the specified server. You can delete entries from base forms only. To remove entries from join forms, delete them from the underlying base forms.

**Privileges** AR System administrator.

**Synopsis**

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

int ARDeleteEntry(
    ARControlStruct          *control,
    ARNameType                schema,
    AREntryIdList             *entryId,
    unsigned int               option,
    ARStatusList               *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**schema**

The name of the base form containing the entry to delete.

**entryId**

The ID of the entry to delete.

**Note:** The system identifies entries in join forms by concatenating the entry IDs from the member forms. As a result, an entry ID can consist of one or more values of type `AREntryIdType` and, therefore, is represented by using the `AREntryIdList` structure.

**option**

Specify 0 for this parameter (reserved for future use).

**Return values****status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

`ARCreateEntry`, `ARDeleteSchema`, `ARGetEntry`, `ARGetListEntry`, `ARSetEntry`. See `FreeAR` for: `FreeAREntryIdList`, `FreeARStatusList`.

## ARDeleteEscalation

**Description**

Deletes the escalation with the indicated name from the specified server and deletes any container references to the escalation. The escalation is deleted from the server immediately and is not returned to users who request information about escalations.

**Privileges**

AR System administrator.

**Synopsis**

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

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

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**name**

The name of the escalation to delete.

**deleteOption**

A bitmask indicating the action to take.

Bit 0: Server object default delete option. For locked objects, fails the delete attempt (`AR_DEFAULT_DELETE_OPTION`).

Bit 2: Delete escalation and all objects in the block (`AR_LOCK_BLOCK_DELETE`).

**Return values****status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

`ARCreateEscalation`, `ARDeleteSchema`, `ARGetEscalation`, `ARGetListEscalation`, `ARSetEscalation`. See `FreeAR` for: `FreeARStatusList`.

# ARDeleteField

**Description** Deletes the form field with the indicated ID from the specified server. Depending on the value you specify for the `deleteOpti` parameter, the field is deleted from the server immediately and is not returned to users who request information about fields. When a parent field is deleted, its child fields might also be deleted. For example, deleting a page holder field will delete all pages within it. All fields within those pages are removed from the current view but not deleted. Deleting a table field will delete all columns within it.

**Privileges** AR System administrator.

**Synopsis**

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

int ARDeleteField(
    ARControlStruct *control,
    ARNameType schema,
    ARInternalId controlId,
    unsigned int deleteOpti,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**schema**

The name of the form containing the field to delete.

**fieldId**

The internal ID of the field to delete.

## deleteOption

A value indicating the action to take if the specified field contains data (applicable for base forms only) or is associated with a join form.

- 0: Do not delete the field (AR\_FI\_ELD\_CLEAN\_DELETE).
- 1: Delete if the field contains data but not if it is associated with a join form (AR\_FI\_ELD\_DATA\_DELETE).
- 2: Delete the field, all join form fields that map to it, and all join forms dependent on it for join qualification (AR\_FI\_ELD\_FORCE\_DELETE).

### Return values

#### [status](#)

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

### See also

[ARCreatField](#), [ARDeleteMultipleFields](#), [ARDeleteSchema](#), [ARGetField](#), [ARGetListField](#), [ARGetMultipleFields](#), [ARSetField](#). See [FreeAR](#) for: [FreeARStatusList](#).

## ARDeleteFilter

**Description** Deletes the filter with the indicated name from the specified server and deletes any container references to the filter. The filter is deleted from the server immediately and is not returned to users who request information about filters.

**Privileges** AR System administrator.

### Synopsis

```
#i ncl ude "ar. h"
#i ncl ude "arerrno. h"
#i ncl ude "arextern. h"
#i ncl ude "arstruct. h"

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

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**name**

The name of the filter to delete.

**deleteOption**

A bitmask indicating the action to take.

- Bit 0: Server object default delete option. For locked objects, fails the delete attempt (AR\_DEFAULT\_DELETE\_OPTION).
- Bit 2: Delete filter and all objects in the block (AR\_LOCK\_BLOCK\_DELETE).

**Return values****status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateFilter, ARDeleteSchema, ARGetFilter, ARGetListFilter, ARSetFilter. See FreeAR for: FreeARStatusList.

## ARDeleteLicense

**Description** Deletes an entry from the license file for the current server.

**Privileges** AR System administrator.

**Synopsis**

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

int ARDeleteLicense(
    ARControlStruct *control,
    ARLicenseNameType licenseeNameType,
    ARLicenseKeyType licenseeKeyType,
    ARStatusList *status)
```

<b>Input arguments</b>	<b>control</b>
	The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.
	<b>licenseType</b>
	The type of license to be deleted. See the arstruct.h file for license types.
	<b>licenseKey</b>
	The unique license key to be deleted.
<b>Return values</b>	<b>status</b>
	A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the <i>Error Messages</i> guide.
<b>See also</b>	ARCreateLicense, ARGetListLicense, ARValidateLicense, ARValidateMultipleLicenses. See FreeAR for: FreeARStatusList.

## ARDelteMultipleFields

**Description** Deletes the form fields with the indicated IDs from the specified server. Depending on the value you specify for the del eteOpti on parameter, the fields are deleted from the server immediately and are not returned to users who request information about fields. This function performs the same action as ARDel eteFi el d but is easier to use and more efficient than deleting multiple entries one by one.

**Privileges** AR System administrator.

### Synopsis

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

```
int ARDel eteMul ti pl eFi el ds(
```

ARControl Struct	*control ,
ARNameType	schema,
ARI nternal l dLi st	*fi el dLi st,
unsi gned i nt	del eteOpti on,
ARStatusLi st	*status)

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**schema**

The name of the form containing the fields to delete.

**fieldList**

The internal IDs of the fields to delete.

**deleteOption**

A value indicating the action to take if one of the specified fields contains data (applicable for base forms only) or are associated with a join form.

- 0: Do not delete the field (`AR_FI_ELD_CLEAN_DELETE`).
- 1: Delete if the field contains data but not if it is associated with a join form (`AR_FI_ELD_DATA_DELETE`).
- 2: Delete the field, all join form fields that map to it, and all join forms dependent on it for join qualification (`AR_FI_ELD_FORCE_DELETE`).

**Return values****status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

`ARCreatField`, `ARDeleteField`, `ARDeleteSchema`, `ARGetField`, `ARGetListField`, `ARGetMultipleFields`, `ARSetField`. See `FreeAR` for: `FreeARInternalIdList`, `FreeARStatusList`.

## ARDeleteSchema

**Description**

Deletes the form with the indicated name from the specified server and deletes any container references to the form. Depending on the value you specify for the `deleteOpti`n parameter, the form is deleted from the server immediately and is not returned to users who request information about forms. In addition, the system deletes all entries, fields, views (VUIs), active links, escalations, and filters associated with the form and all containers owned by the form.

**Privileges** AR System administrator.

**Synopsis**

```
#i ncl ude "ar. h"
#i ncl ude "arerrno. h"
#i ncl ude "arextern. h"
#i ncl ude "arstruct. h"

int ARDeleteSchema(
    ARControlStruct *control,
    ARNameType name,
    unsigned int deleteOption,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**name**

The name of the form to delete.

**deleteOption**

A bitmask indicating the action to take if the specified form contains entries (applicable for base forms only) or is a member of a join form.

Bit 0: Delete schema only when there is no dependency and there is no data in the schema, set the bit to zero (0). This is ignored in the case of a join form (AR\_SCHEMA\_CLEAN\_DELETE).

To delete schema even if there is data, set the bit to one (1). This is only applicable for regular schemas (AR\_SCHEMA\_DATA\_DELETE).

Bit 1: Delete schema even if there is a dependency. All the specified schemas will be deleted. This option overrides AR\_SCHEMA\_DATA\_DELETE (AR\_SCHEMA\_FORCE\_DELETE).

Bit 2: Delete the given object and all objects that are locked with the same key. This is only applicable for locked objects (AR\_LOCK\_BLOCK\_DELETE).

Bit 3: Delete the archive form even if archive is enabled or the archive form is part of a lock block (AR\_SCHEMA\_SHADOW\_DELETE).

**Return values**

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateSchema, ARGetSchema, ARGetListAlertUser, ARSetSchema. See FreeAR for: FreeARStatusList.

## ARDeleteSupportFile

**Description** Deletes a support file in the AR System.

**Privileges** AR System administrator.

**Synopsis**

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

int ARDeleteSupportFile(
    ARControlStruct *control,
    unsigned int fileType,
    ARNameType name,
    ARInternalId id2,
    ARInternalId fileId,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**fileType**

The numerical value for the type of file, and the type of object the file is related to. Specify 1 (`AR_SUPPORT_FILE_EXTERNAL_REPORT`) for an external report file associated with an active link.

**name**

The name of the object the file is associated with, usually a form.

**id2**

The ID of the field or VUI, if the object is a form. If the object is not a form, set this parameter to 0.

**fileId**

The unique identifier of a file within its object.

**Return values** [status](#)

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateAlertEvent, ARCreateSupportFile, ARDeleteActiveLink, ARGetActiveLink, ARGetListSupportFile, ARGetSupportFile, ARSetActiveLink, ARSetSupportFile. See FreeAR for: FreeARStatusList.

## ARDeleteVUI

**Description** Deletes the form view (VUI) with the indicated ID from the specified server.

**Privileges** AR System administrator.

**Synopsis**

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

int ARDeleteVUI (
    ARControlStruct *control,
    ARNameType schema,
    ARInternalID vui_id,
    ARStatusList *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**schema**

The name of the form containing the VUI to delete.

**vuid**

The internal ID of the VUI to delete.

**Return values** [status](#)

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateVUI, ARDeleteSchema, ARGetVUI, ARGetListVUI, ARSetVUI.  
See FreeAR for: FreeARStatusList.

## ARDeregisterForAlerts

**Description** Cancels registration for the specified user on the specified AR System server and port. The system deletes the user from the active list and stops delivering alerts.

**Privileges** All users.

**Synopsis**

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

```
int ARDeregisterForAlerts(
    ARControlStruct *control,
    int clientPort,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**clientPort**

The TCP port that the client uses. Client programs can use different port numbers to register with the same server multiple times, but must deregister individually.

**Return values**

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

ARRegisterForAlerts, ARCreateAlertEvent. See FreeAR for: FreeARStatusList.

# AREncodeARAssignStruct

<b>Description</b>	Converts an ARAssi gnStruct structure into a serialized assignment string. The resulting string is stored in a DEF file and is used for exporting definitions of ARServer objects.
<b>Privileges</b>	All users.
<b>Synopsis</b>	<pre>#i ncl ude "ar.h" #i ncl ude "arerrno.h" #i ncl ude "arextern.h" #i ncl ude "arstruct.h"  int (AREncodeARAssi gnStruct       ARControl Struct *control,       ARAssi gnStruct *assi gnStruct,       **assi gnText,       ARStatusLi st *status)</pre>
<b>Input arguments</b>	<p><b>control</b>  The control record for the operation. It contains information about the user requesting the operation, where that operation is performed, and what session performs it. The user and server fields are required.</p> <p><b>assignStruct</b>  The ARAssi gnStruct structure to convert.</p>
<b>Return values</b>	<p><b>assignText</b>  A serialized representation of the assignment structure.</p> <p><b>status</b>  A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the <i>Error Messages</i> guide.</p>
<b>See also</b>	ARDecodeARQualifierStruct, AREncodeARAssignStruct, AREncodeARQualifierStruct, ARDecodeDiary, AREncodeDiary, ARDecodeStatusHistory, AREncodeStatusHistory. See FreeAR for: FreeARStatusList.

# AREncodeARQualifierStruct

**Description** Converts an ARQualifierStruct into a serialized qualification string. The resulting string is stored in a DEF file and is used for exporting definitions of ARServer objects.

**Privileges** All users.

## Synopsis

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

int AREncodeARQualifierStruct(
    ARControlStruct *control,
    ARQualifierStruct *qualStruct,
    char **qualText,
    ARStatusList *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

### qualStruct

The ARQualifierStruct structure to convert.

## Return values

### qualText

A serialized representation of the qualification structure.

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## See also

ARDecodeARQualifierStruct, AREncodeARAssignStruct, AREncodeARQualifierStruct, ARDecodeDiary, AREncodeDiary, ARDecodeStatusHistory, AREncodeStatusHistory. See FreeAR for: FreeARStatusList.

# AREncodeDiary

**Description** Converts an ARDi aryLi st into a diary string. The resulting string is stored in a DEF file and is used for exporting definitions of ARServer objects.

**Privileges** All users.

**Synopsis**

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

```
int AREncodeDiary(
    ARControlStruct *control,
    ARDi aryLi st,
    char **diaryString,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**diaryList**

The ARDi aryLi st to convert.

**Return values**

**diaryString**

The resulting diary string.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

ARDecodeARQualifierStruct, AREncodeARAssignStruct, AREncodeARQualifierStruct, ARDecodeDiary, AREncodeDiary, ARDecodeStatusHistory, AREncodeStatusHistory. See FreeAR for: FreeARStatusList.

# AREncodeStatusHistory

**Description** Converts an ARStatusHi storyList into a status history string. The resulting string is stored in a DEF file and is used for exporting definitions of ARServer objects.

**Privileges** All users.

## Synopsis

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

int AREncodeStatusHiStory(
    ARControlStruct *control,
    ARStatusHiStoryList *statHistList,
    char **statHistString,
    ARStatusList *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

### statHistList

The ARStatusHiStoryList structure to convert.

## Return values

### statHistString

The resulting status history string.

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## See also

ARDecodeARQualifierStruct, AREncodeARAssignStruct, AREncodeARQualifierStruct, ARDecodeDiary, AREncodeDiary, ARDecodeStatusHistory, AREncodeStatusHistory. See FreeAR for: FreeARStatusList.

# AREndBulkEntryTransaction

**Description** Marks the ending of a series of entry API function calls that are grouped together and sent to the AR System server as part of one transaction. All calls related to create, set, delete, and merge operations made before this API call and after the preceding `ARBeginBulkEntryTransaction` call will be sent to the server when this call is issued and executed within a single database transaction.

**Privileges** All users.

**Synopsis**

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

```
int AREndBulkEntryTransaction(
    ARControlStruct *control,
    unsigned int actionType,
    ARBulkEntryReturnList *bulkEntryReturnList,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**actionType**

The action to take. Use `AR_BULK_ENTRY_ACTION_SEND` to transmit the queued entry calls to the server and end the transaction. Use `AR_BULK_ENTRY_ACTION_CANCEL` to remove the queued entry calls and end the transaction.

**Return values**

**bulkEntryReturnList**

A list of return structures corresponding to the individual operations executed within the bulk entry transaction. This parameter can be `NULL`. If it is not `NULL`, this parameter will return information regardless of whether the `status` parameter indicates an error occurred. The status of the individual calls is returned in the order the calls were issued, up to and including the last call executed on the server. If any of the individual calls fail, the list terminates with the return structure for the failed call. Any return information from previous calls that were successful can be ignored.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See the *Error Messages* guide for a description of all possible values. If any of the individual entry calls fail during the execution of the API call, the error code 9713 (AR\_ERROR\_BULK\_ENTRY\_CALL\_FAILED) will be returned and the transaction will be rolled back. The status of each individual entry call will be returned in the `kEntryReturnList`.

**See also** ARBeginBulkEntryTransaction.

# ARExecuteProcess

**Description** Performs the indicated command on the specified server. Depending on the values you specify for the `returnStatus` and `returnString` parameters, you can execute the command as an independent process or wait for the process to complete and return the result to the client. The system executes the command based on the access privileges of the user who launched the AR System server.

**Privileges** AR System administrator.

**Synopsis**

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

int ARExecuteProcess(
    ARControlStruct *control,
    char *command,
    int *returnStatus,
    char **returnString,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**command**

The command to execute (must be a valid command on the specified server).

**Return values****[returnStatus](#)**

An integer identifying the status of the operation. A value of 0 generally indicates success. Any other value generally indicates a failure. Specify `NULL` for this parameter and the `returnString` parameter if you want the system to launch an independent process and not wait for it to complete. Otherwise, specify a value for this parameter if you want the system to wait for the process to complete before returning. If the process does not finish within the time-out interval, adjust the filter process time-out interval to prevent server blocking. For information about the process time-out interval, see the *Configuring* guide.

**[returnString](#)**

A string containing the process output. Depending on the outcome of the operation, this string contains either result data or an error message. Specify `NULL` for this parameter and the `returnStatus` parameter if you want the system to launch an independent process and not wait for it to complete. Otherwise, specify a value for this parameter if you want the system to wait for the process to complete before returning. If the process does not finish within the time-out interval, adjust the filter process time-out interval to prevent server blocking (configurable from 1 to 20 seconds).

**[status](#)**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** [ARGetListSQL](#). See [FreeAR](#) for: `FreeARStatusList`.

## **ARExpandCharMenu**

**Description** Expands the references for the specified menu definition and returns a character menu with list-type items only.

**Privileges** The system returns information based on the access privileges of the user you specify for the `control` parameter. All query items, therefore, are limited to fields the user can access.

**Synopsis**

```
#i ncl ude "ar.h"
#i ncl ude "arerrno.h"
#i ncl ude "arextern.h"
#i ncl ude "arstruct.h"

int ARExpandCharMenu(
    ARControl Struct           *control ,
    ARCharMenuStruct           *menuIn,
    ARCharMenuStruct           *menuOut,
    ARStatusList                *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**menuIn**

The menu definition to expand.

**Return values****menuOut**

The expanded character menu. The menu definition contains list-type items only.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

`ARCCreateCharMenu`, `ARDeleteCharMenu`, `ARGetCharMenu`, `ARGetListCharMenu`, `ARSetCharMenu`. See `FreeAR` for: `FreeARCharMenuStruct`, `FreeARStatusList`.

# ARExport

<b>Description</b>	Exports the indicated structure definitions from the specified server. Use this function to copy structure definitions from one AR System server to another (see “ARImport” on page 363).
--------------------	---

---

**Note:** Form exports do not work the same way with ARExport as they do in BMC Remedy Administrator. Other than views, you cannot automatically export related items along with a form. You must explicitly specify the workflow items you want to export. Also, ARExport cannot export a form without embedding the server name in the export file (something you can do with the “Server-Independent” option in BMC Remedy Administrator).

---

<b>Privileges</b>	For filters, escalations, character menus, and distributed mappings, this operation can be performed by users with AR System administrator privileges only. For forms, containers, and active links, this operation can be performed by users with access permission for the specified structure. Access to groupList information for these structures is limited to users with AR System administrator privileges only.
-------------------	--

## Synopsis

```
#i ncl ude "ar.h"
#i ncl ude "arerrno.h"
#i ncl ude "arextern.h"
#i ncl ude "arstruct.h"

int ARExport(
    ARControlStruct *control,
    ARStructList tems,
    ARNameType di spl ayTag,
    unsi gned int vui Type,
    ARWorkfl owLockStruct *I ocki nfo,
    char **exportBuf,
    ARStatusList *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

## structItems

A list of zero or more structure items to export (identified by type and name).

### AR Format:

- 1: **Form (includes all associated views)** (AR\_STRUCT\_ITEM\_SCHEMA).
- 2: **Partial form (fields only)** (AR\_STRUCT\_ITEM\_SCHEMA\_DEFN).
- 3: **Partial form (specified view only)** (AR\_STRUCT\_ITEM\_SCHEMA\_VIEW).
- 4: **Partial form (email template only)** (AR\_STRUCT\_ITEM\_SCHEMA\_MAIL).
- 5: **Filter** (AR\_STRUCT\_ITEM\_FILTER).
- 6: **Active link** (AR\_STRUCT\_ITEM\_ACTIVELINK).
- 8: **Character menu** (AR\_STRUCT\_ITEM\_CHAR\_MENU).
- 9: **Escalation** (AR\_STRUCT\_ITEM\_ESCALATION).
- 10: **Distributed mapping** (AR\_STRUCT\_ITEM\_DIST\_MAP).
- 11: **Partial form (specified view only, without button images)** (AR\_STRUCT\_ITEM\_SCHEMA\_VIEW\_MINOR).
- 12: **Container** (AR\_STRUCT\_ITEM\_CONTAINER).
- 16: **Application** (AR\_STRUCT\_ITEM\_APP).
- 30: **Schema data** (AR\_STRUCT\_ITEM\_XML\_SCHEMA\_DATA).

**XML Format:**

```
1073741825 Form (includes all associated views)
:
(AR_STRUCT_I TEM_XML_SCHEMA).

1073741829 Filter (AR_STRUCT_I TEM_XML_FILTER).
:
1073741830 Active link (AR_STRUCT_I TEM_XML_ACTIVE_LINK).

1073741832 Character menu (AR_STRUCT_I TEM_XML_CHAR_MENU).
:
1073741833 Escalation (AR_STRUCT_I TEM_XML_ESCALATION).
:
1073741834 Distributed mapping (AR_STRUCT_I TEM_XML_DIST_MAP).
:
1073741836 Container (AR_STRUCT_I TEM_XML_CONTAINER).
:
1073741837 Distributed pool (AR_STRUCT_I TEM_XML_DIST_POOL).
:
1073741838 View (AR_STRUCT_I TEM_XML_VUI).
:
1073741839 Field (AR_STRUCT_I TEM_XML_FIELD).
:
1073741840 Application (AR_STRUCT_I TEM_XML_APP).
:
1073741854 Schema data (AR_STRUCT_I TEM_XML_SCHEMA_DATA).
:
```

You must specify AR\_STRUCT\_I TEM\_SCHEMA to export a form that you intend to import to another server. The three partial form types do not contain complete form definitions and are used for caching purposes only.

The AR\_STRUCT\_XML\_OFFSET ( $1 << 30$ ) bitmask is used to derive the XML structure item types. The XML type is the logical-OR of the non-XML type and the XML offset. For example, AR\_STRUCT\_I TEM\_XML\_SCHEMA is defined in the ar.h file as (AR\_STRUCT\_XML\_OFFSET | AR\_STRUCT\_I TEM\_SCHEMA).

**displayTag**

The label of the form view (VUI) to export. You must specify this parameter to export a particular view (AR\_STRUCT\_I TEM\_SCHEMA\_VIEW). If the specified view does not exist (or you specify NULL for this value), the system exports the default view. The system exports the first view in the list if the form does not have a default view.

## vuiType

The type of VUI to export.

- 0: No VUI type (AR\_VUI\_TYPE\_NONE).
- 1: Windows type (AR\_VUI\_TYPE\_WINDOWS) - fields in BMC Remedy User.
- 2: Web relative type (AR\_VUI\_TYPE\_WEB) - fields in view can be adjusted.
- 3: Web absolute type (AR\_VUI\_TYPE\_WEB\_ABS\_POS) - fields in view are fixed.

## lockInfo

To export objects as *locked*, you must supply this parameter. If exporting in unlocked mode, you can set this to NULL.

---

**Note:** The locking feature is for protecting your intellectual property. You can now export workflow as locked (either read-only lock or hidden-lock) and deliver it to your customers. When your customer imports the workflow, the system will seamlessly run that workflow but restrict the clients from getting the definition data, thus preventing human administrators or API clients from viewing or modifying the definition.

---

The values consist of a lock type and lock key. Valid values for the lock type are:

- 0: Export the objects as unlocked (AR\_LOCK\_TYPE\_NONE).
- 1: Export the objects with read-only lock. When these objects are imported, they are readable but not modifiable (AR\_LOCK\_TYPE\_READONLY).
- 2: Export the objects with hidden-lock. When these objects are imported, they are neither readable nor modifiable (AR\_LOCK\_TYPE\_HIDDEN).

Lock key is a string that is to be used as a key (or a password) to enforce the locking.

## Return values

### exportBuf

A buffer that contains the definition text for the items specified for the structl tems parameter. The system returns error messages for items not exported due to error.

## **status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARImport. See FreeAR for: FreeARStatusList, FreeARStructItemList.

# **ARExportLicense**

**Description** Specifies a pointer that is set to `malloced` space and contains the full contents of the license file currently on the server. This buffer can be written to a file to produce an exact replication of the license file, including all checksums and encryption.

**Privileges** AR System administrator

### **Synopsis**

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

int ARExportLicense(
    ARControlStruct *control,
    char **exportBuf,
    ARStatusList *status)
```

### **Input arguments**

**control** The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

### **exportBuf**

The image of the license file.

### **Return values**

### **status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARImportLicense.

# ARGetActiveLink

**Description** Retrieves information about the active link with the indicated name on the specified server.

**Privileges** This operation can be performed by users with access permission for the active link. Access to groupList information is limited to users with AR System administrator privileges only.

## Synopsis

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

int ARGetActiveLink(
    ARControlStruct *control,
    ARNameType name,
    *order,
    *schemaList,
    *groupList,
    *executeMask,
    *controlField,
    *focusField,
    *enable,
    *query,
    *actionList,
    *elementList,
    **helpText,
    *timestamp,
    owner,
    lastChanged,
    **changeDir,
    *objPropList,
    *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

### name

The name of the active link to retrieve.

**Return values****order**

A value between 0 and 1000 (inclusive) that determines the active link execution order. When multiple active links are associated with a form, the value associated with each active link determines the order in which they are processed (from lowest to highest). Specify `NULL` for this parameter if you do not want to retrieve this value.

**schemaList**

The list of form names the active link is linked to. The active link must be associated with a single form or a list of forms that currently exists on the server. Specify `NULL` for this parameter if you do not want to retrieve the list.

**groupList**

A list of zero or more groups who can access this active link. Access to this information is limited to users with AR System administrator privileges only. Specify `NULL` for this parameter if you do not want to retrieve this value.

**executeMask**

A bitmask indicating the form operations that trigger the active link. See “[ARCreateActiveLink](#)” on page 128 for a description of the possible values. Specify `NULL` for this parameter if you do not want to retrieve this value.

**controlField**

The ID of the field that represents the button, toolbar button, or menu item associated with executing the active link. The system returns zero if the `executeMask` does not include the `AR_EXECUTE_ON_BUTTON` condition. Specify `NULL` for this parameter if you do not want to retrieve this value.

**focusField**

The ID of the field associated with executing the active link by pressing Return or selecting a character menu item. The system returns zero if the `executeMask` does not include the `AR_EXECUTE_ON_RETURN` or `AR_EXECUTE_ON_MENU_CHOICE` conditions. Specify `NULL` for this parameter if you do not want to retrieve this value.

**enable**

A flag identifying whether the active link is disabled (0) or enabled (1). Specify `NULL` for this parameter if you do not want to retrieve this value.

## query

A qualification that determines whether the active link is executed. The system returns zero (AR\_COND\_OP\_NONE) if the active link has no qualification. Specify `NULL` for this parameter if you do not want to retrieve this value.

## actionList

The set of actions performed if the condition defined by the `query` parameter is satisfied. This list can contain from 1 to 25 actions (limited by AR\_MAX\_ACTIONS). Specify `NULL` for this parameter if you do not want to retrieve this value.

## elseList

The set of actions performed if the condition defined by the `query` parameter is not satisfied. This list can contain from 0 to 25 actions (limited by AR\_MAX\_ACTIONS). Specify `NULL` for this parameter if you do not want to retrieve this value.

## helpText

The help text associated with the active link. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

## timestamp

A time stamp identifying the last change to the active link. Specify `NULL` for this parameter if you do not want to retrieve this value.

## owner

The active link owner. Specify `NULL` for this parameter if you do not want to retrieve this value.

## lastChanged

The user who made the last change to the active link. Specify `NULL` for this parameter if you do not want to retrieve this value.

## changeDiary

The change diary associated with the active link. Use ARDecodeDiary to parse the change diary into user, time stamp, and text components. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to retrieve the change diary (which is useful if you are calling this function to verify whether an instance of this object exists).

## objPropList

If the objPropList parameter is set to NULL, no object properties list will be returned. See “Server object property tags” on page 79 for object property tag names and data types.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateAlertEvent, ARDecodeDiary, ARDeleteActiveLink, ARGetListActiveLink, ARGetMultipleActiveLinks, ARSetActiveLink. See FreeAR for: FreeARActiveLinkActionList, FreeARInternalIdList, FreeARPropList, FreeARQualifierStruct, FreeARStatusList.

# ARGetAlertCount

**Description** Retrieves the count of qualifying alert events located on the server that the control structure specifies.

**Privileges** All users.

### Synopsis

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

int ARGetAlertCount(
    ARControlStruct *control,
    ARQualifierStruct *qualifier,
    unsigned int *count,
    ARStatusList *status)
```

**Input arguments**

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

### qualifier

The qualifying criteria for the alert events that this call counts. Specify NULL or assign an operation value of 0 (AR\_COND\_OP\_NONE) to count all alert events.

**Return values****count**

The number of alert events that meet the criteria that the `qualifier` argument specifies.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

`ARRegisterForAlerts`, `ARCreateAlertEvent`. See `FreeAR` for: `FreeARStatusList`.

## ARGetApplicationState

**Description**

Retrieves the application state: maintenance (admin only), test, or production.

**Privileges**

All users.

**Synopsis**

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

```
int ARGetApplicationState(
    ARControlStruct *control,
    ARNameType appli cati onName,
    ARNameType currentStateName,
    ARStatusList *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**applicationName**

The name of the application.

**Return values****currentStateName**

The current value for the state field, `currentStateName`, on the AR System Application State form.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARSetApplicationState, ARGetListApplicationState.

# ARGetCharMenu

**Description** Retrieves information about the character menu with the indicated name on the specified server.

**Privileges** All users.

### Synopsis

```
#i ncl ude " ar. h"
#i ncl ude " arerrno. h"
#i ncl ude " arextern. h"
#i ncl ude " arstruct. h"

int ARGetCharMenu(
    ARControlStruct *control,
    ARNameType      name,
    *refreshCode,
    *menuDefn,
    **helpText,
    *timestamp,
    owner,
    lastChanged,
    **changeDir,
    *objPropList,
    *status)
```

**Input arguments**

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

### name

The name of the character menu to retrieve.

**Return values****refreshCode**

A value indicating when the menu is refreshed. See “ARCreateCharMenu” on page 134 for a description of the possible values. Specify `NULL` for this parameter if you do not want to retrieve this value.

**menuDefn**

The definition of the character menu. Specify `NULL` for this parameter if you do not want to retrieve this value.

**helpText**

The help text associated with the character menu. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

**timestamp**

A time stamp identifying the last change to the character menu. Specify `NULL` for this parameter if you do not want to retrieve this value.

**owner**

The owning user for the character menu. Specify `NULL` for this parameter if you do not want to retrieve this value.

**lastChanged**

The user who made the last change to the character menu. Specify `NULL` for this parameter if you do not want to retrieve this value.

**changeDiary**

The change diary associated with the character menu. Use `ARDecodeDi ary` to parse the change diary into user, time stamp, and text components. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to retrieve the change diary (which is useful if you are calling this function to verify whether an instance of this object exists).

**objPropList**

If the `objPropList` parameter is set to `NULL`, an object properties list with zero properties will be associated with the object, and when an `ARGetCharMenu` action is performed, zero properties are returned. See “Server object property tags” on page 79 for object property tag names and data types.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateCharMenu, ARDecodeDiary, ARDeleteCharMenu, ARExpandCharMenu, ARGetListCharMenu, ARGetMultipleCharMenus, ARSetCharMenu. See FreeAR for: FreeARCharMenuStruct, FreeARPropList, FreeARStatusList.

# ARGetClientCharSet

**Description** Retrieves a string that represents the name of the character set the client is using. The API assumes that all character data the client passes it is encoded in this character set, and returns all character data encoded in this character set.

Use the ARControl Struct `localInfo.charset` field to set the client character set as described under `ARInitialization()`.

**Privileges** All users.

### Synopsis

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

int ARGetClientCharSet(
    ARControlStruct          *control,
    char                      *charSet,
    ARStatusList              *status)
```

**Input arguments**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `server` field is required.

### Return values

#### charSet

A string that names the character set the client is using. `charSet` is a buffer whose size is at least `AR_MAX_LANG_SIZE+1`.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARGetServerCharSet

# ARGetContainer

**Description** Retrieves the contents of the container with the indicated name on the specified server. It can return references of a single, specified type, of all types, or of an exclude reference type. The system returns information for accessible references and does nothing for references for which the user does not have access.

**Privileges** This operation can be performed by users with access permission for the container. Access to `groupList` and `adminGrpList` information is limited to users with AR System administrator privileges only.

**Synopsis**

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

int ARGetContainer(
    ARControlStruct *control,
    ARNameType name,
    *refTypes,
    *groupList,
    *adminGrpList,
    *ownerObjList,
    **label,
    **description,
    *type,
    *references,
    **helpText,
    owner,
    *timestamp,
    lastChanged,
    **changeDir,
    *objPropList,
    *status)
```

Input arguments	<b>control</b>
	The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.
	<b>name</b>
	The name of the container to retrieve.
	<b>refTypes</b>
	A list of the types of references (for example, forms and filters) to retrieve. You can specify individual types of references to retrieve, specify that all (ARREF_ALL) or none (ARREF_NONE) of the references be retrieved, or specify negative numbers to treat the types of references as exclude reference types. The exclude reference types take precedence over the include list; this means that if you specify a type as positive as well as negative, then all references of that type are excluded.
Return values	<b>groupList</b>
	A list of zero or more groups who can access this container. Access to this information is limited to users with AR System administrator privileges only. Specify <code>NULL</code> for this parameter if you do not want to retrieve this value.
	<b>adminingrpList</b>
	A list of zero or more groups who can administer this container (and the referenced objects). If <code>ownerObj</code> does not return <code>NULL</code> , this list is the Subadministrator group list of the owning form. Users must belong to both a specified group <i>and</i> the Subadministrator group to obtain these privileges. Specify <code>NULL</code> for this parameter if you do not want to retrieve this value.
	<b>ownerObjList</b>
	A list of the schemas that own this container. Specify <code>NULL</code> for this parameter if you do not want to retrieve this value. If this parameter returns <code>NULL</code> , the container exists globally.
	<b>label</b>
	The label for this container. Specify <code>NULL</code> for this parameter if you do not want to retrieve this value.
	<b>description</b>
	The description for this container. Specify <code>NULL</code> for this parameter if you do not want to retrieve this value.

## type

The type for this container. Specify `NULL` for this parameter if you do not want to retrieve this value.

## references

Pointers to the objects (for example, forms or filters) referenced by this container. Specify `NULL` for this parameter if you do not want to retrieve this value.

## helpText

The help text associated with the container. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

## owner

The owning user for the container. Specify `NULL` for this parameter if you do not want to retrieve this value.

## timestamp

A time stamp identifying the last change to the container. Specify `NULL` for this parameter if you do not want to retrieve this value.

## lastChanged

The user who made the last change to the container. Specify `NULL` for this parameter if you do not want to retrieve this value.

## changeDiary

The change diary associated with the container. The server adds the user making the change and a time stamp when it saves the change diary. Use `ARDecodeDiary` to parse the change diary into user, time stamp, and text components. Specify `NULL` for this parameter if you do not want to retrieve the change diary (which is useful if you are calling this function to verify whether an instance of this object exists).

## objPropList

If the `objPropList` parameter is set to `NULL`, an object properties list with zero properties will be associated with the object. See “Server object property tags” on page 79 for object property tag names and data types.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateContainer, ARDecodeDiary, ARDeleteContainer, ARGetListContainer, ARGetSchema, ARSetContainer. See FreeAR for: FreeARContainerInfoList, FreeARInternalIdList, FreeARPermissionList, FreeARPropList, FreeARReferenceList, FreeARStatusList.

# ARGetCurrencyRatio

**Description** Retrieves a selected currency ratio from a set of ratios returned when the client program makes a call to ARGetMul ti pl eCurrencyRati oSets.

**Privileges** All users.

**Synopsis**

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

int ARGetCurrencyRatio(
    ARControlStruct *control,
    char *currencyRatios,
    ARCurrencyCodeType fromCurrencyCode,
    ARCurrencyCodeType toCurrencyCode,
    ARValueStruct *currencyRatio,
    ARStatusList *status)
```

**Input arguments**

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and session fields are required.

### currencyRatios

A formatted currency ratio character string that the ARGetMul ti pl eCurrencyRati oSets call returns.

### fromCurrencyCode

The source currency code for the conversion ratio.

## toCurrencyCode

The target currency code for the conversion ratio

### Return values

#### currencyRatio

The conversion ratio for the specified source and target currency codes. The call returns a value structure of type AR\_DATA\_TYPE\_NULL if there is no conversion ratio for the specified currency code combination. The call returns a value structure of type AR\_DATA\_TYPE\_DECIIMAL if there is a conversion ratio. You must free the AR\_DATA\_TYPE\_DECIIMAL value when it is no longer needed.

#### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARGetMultipleCurrencyRatioSets. See FreeAR for: FreeARValueStruct.

## ARGetEntry

**Description** Retrieves the form entry with the indicated ID on the specified server. You can retrieve data for specific fields, all (accessible) fields, or no fields (which is useful to verify whether a form has any entries). This function only returns the name, size, and compressed size of attachment fields. Use ARGetEntryBLOB to retrieve the contents of the attachment.

**Privileges** The system returns data based on the access privileges of the user you specify for the control parameter. User permissions are verified for each specified field. The system returns values for accessible fields and warning messages for fields the user cannot access.

### Synopsis

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

int ARGetEntry(
    ARControlStruct *control,
    ARNameType schema,
    AREntryIdList *entryId,
    ARInternalList *idList,
```

ARFi el dVal ueLi st  
ARStatusLi st

\*fi el dLi st,  
\*status)

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

### schema

The name of the form containing the entry to retrieve.

entryId

The ID of the entry to retrieve.

---

**Note:** The system identifies entries in join forms by concatenating the entry IDs from the member forms. As a result, an entry ID can consist of one or more values of type AREntryIdType and, therefore, is represented by using the AREntryIdList structure.

---

## idList

A list of zero or more internal IDs specifying the fields to retrieve. Specify NULL for this parameter (or zero fields) to retrieve all (accessible) fields.

Specify NULL for both this parameter and the fieldList parameter if you do not want to retrieve any fields. To minimize network traffic, specify only the fields you need if you do not require the data for all fields. If an attachment field is specified in the list, only its name, size, and compressed size will be returned. Use ARGetEntryBLOB to retrieve the contents of the attachment.

## Return values

### fieldList

A list of zero or more field/value pairs that identifies the data for the specified entry. The fields are returned in the order specified by idList. If the user does not have permission for a specified field or the field does not exist, the system does not return a value for the field/value pair. Specify NULL for this parameter if you do not want to retrieve any field data.

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateEntry, ARDecodeDiary, ARDecodeStatusHistory, ARDeleteEntry, ARGetEntryBLOB, ARGetListEntry, ARSetEntry. See FreeAR for: FreeAREntryIdList, FreeARInternalIdList, FreeARFieldValueList, FreeARStatusList.

## ARGetEntryBLOB

**Description** Retrieves the attachment, or binary large object (BLOB), stored for the attachment field with the indicated ID on the specified server. The BLOB can be placed in a buffer or a file.

**Privileges** The system returns data based on the access privileges of the user you specify for the control parameter. User permissions are verified for the specified field. If the user cannot access the field, the system returns an error message.

### Synopsis

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

int ARGetEntryBLOB(
    ARControlStruct          *control,
    ARNameType                schema,
    AREntryIdList             *entryId,
    ARInternalId               id,
    ARLocStruct                *loc,
    ARStatusList               *status)
```

### Input arguments

#### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

#### schema

The name of the form containing the entry to retrieve.

**entryId**

The ID of the entry to retrieve.

---

**Note:** The system identifies entries in join forms by concatenating the entry IDs from the member forms. As a result, an entry ID can consist of one or more values of type `AREntryIdType` and, therefore, is represented by using the `AREntryIdList` structure.

---

**id**

The ID specifying the field to retrieve.

**loc**

A pointer to an `ARLocStruct` structure specifying how you want the contents of the blob returned: in a file (`AR_LOC_FILENAME`) or a data buffer (`AR_LOC_BUFFER`). The structure also contains the name of the file or buffer to be used.

**Return values****status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

`ARGetEntry`, `ARCreateEntry`, `ARDecodeDiary`, `ARDecodeStatusHistory`, `ARDeleteEntry`, `ARGetListEntry`, `ARSetEntry`. See `FreeAR` for: `FreeAREntryIdList`, `FreeARInternalIdList`, `FreeARFieldValueList`, `FreeARStatusList`.

# ARGetEntryBlock

**Description** Retrieves a list of entries contained in a block of entries retrieved using ARGetListEntryBlocks.

**Privileges** The system returns data based on the access privileges of the user you specify for the `control` parameter. User permissions are verified for each specified field. The system returns values for accessible fields and warning messages for fields the user cannot access.

## Synopsis

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

int ARGetEntryBlock(
    AREntryBlockList *entryBlockList,
    unsigned int blockNumber,
    AREntryList *entryList,
    ARStatusList *status)
```

## Input arguments

### [entryBlockList](#)

A list of entry blocks retrieved by ARGetListEntryBlocks.

### [blockNumber](#)

The number of the block of entries for which you want to retrieve a list. A value of 0 represents the first block.

## Return values

### [entryList](#)

A list of entries contained in the specified block.

### [status](#)

A list of zero or more notes, warnings, or errors generated from this function call.

## See also

[ARGetListEntryBlocks](#).

# ARGetEntryStatistics

<b>Description</b>	Computes the indicated statistic for the form entries that match the conditions specified by the <code>qualifier</code> parameter and returns the values sorted in ascending order by the <code>groupByList</code> .
<b>Privileges</b>	The system returns information based on the access privileges of the user you specify for the <code>control</code> parameter. All statistics, therefore, are limited to entries the user can access (users must have permission for the <code>entryId</code> field to access and retrieve entries).
<b>Synopsis</b>	<pre>#include "ar.h" #include "arerrno.h" #include "arextern.h" #include "arstruct.h"  int ARGetEntryStatistics(     ARControlStruct *control,     ARNameType schema,     ARQualifierStruct *qualifier,     ARFieldValuerOrArrayListStruct *target,     ARStatisticsList *statistics,     ARStatisticsResultList *results,     ARStatusList *status)</pre>
<b>Input arguments</b>	<p><b>control</b>  The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The <code>user</code> and <code>server</code> fields are required.</p> <p><b>schema</b>  The name of the form to compute entry statistics for.</p> <p><b>qualifier</b>  A query that determines the set of entries to use. The qualification can include one or more fields and any combination of conditional, relational, and arithmetic (numeric data types only) operations. The system generates an error if the user does not have read permission for a field or a field does not exist. Specify <code>NULL</code> or assign an operation value of 0 (<code>AR_COND_OP_NONE</code>) to match all form entries.</p>

## target

The arithmetic operation that defines the statistic to compute. The statistic can include one or more fields and any combination of arithmetic operations. The system generates an error if the user does not have read permission for a field or a field does not exist. If you specify AR\_STAT\_OP\_COUNT for the statistic parameter, assign a tag value of 0 to omit this parameter.

## statistic

A value indicating the statistic type.

- 1: The total number of matching entries (AR\_STAT\_OP\_COUNT).
- 2: The sum of values for each group (AR\_STAT\_OP\_SUM).
- 3: The average value for each group (AR\_STAT\_OP\_AVERAGE).
- 4: The minimum value for each group (AR\_STAT\_OP\_MINIMUM).
- 5: The maximum value for each group (AR\_STAT\_OP\_MAXIMUM).

## groupByList

A list of zero or more fields to group the results by. The system computes a result for each group of entries having the same value in the specified field. Specifying more than one field creates groups within groups, each of which returns a separate statistic. Specify `NULL` for this parameter (or zero fields) to compute a single result for all matching entries.

## Return values

### results

A list of zero or more results sorted in ascending order. If you specify one or more fields for the `groupByList` parameter, each item in the list represents a group. Each result structure contains the field values that define the group and the statistic for that group.

For example, if the query was to find a count of bugs grouped by submitter and then by status, the results would return data sorted by submitter and then sorted by status, as in the following list:

```
Abe Low 1
Abe Medium 3
Dani Low 11
Dani Medium 10
Dani High 5
Sarah Low 8
Sarah High 2
```

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateEntry, ARDeleteEntry, ARGetEntry, ARGetListEntry, ARLoadARQualifierStruct, ARMergeEntry, ARSetEntry. See FreeAR for: FreeARFieldValueOrArithStruct, FreeARInternalIdList, FreeARQualifierStruct, FreeARStatisticsResultList, FreeARStatusList.

## ARGetEscalation

**Description** Retrieves information about the escalation with the indicated name on the specified server.

**Privileges** AR System administrator.

**Synopsis**

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

int ARGetEscalation(
    ARControlStruct *control,
    ARNameType name,
    *escalationTm,
    *schemaList,
    *enable,
    *query,
    *actionList,
    *elseList,
    **helpText,
    *timestamp,
    owner,
    lastChanged,
    **changeDictionary,
    *objPropList,
    *status)
```

Input arguments	<b>control</b>
	The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.
Return values	<b>name</b>
	The name of the escalation to retrieve.
	<b>escalationTm</b>
	The time specification for evaluating the escalation condition. This parameter can take one of two forms: a time interval that defines how frequently the server checks the escalation condition (in seconds) or a bitmask that defines a particular day (by month or week) and time (hour and minute) for the server to check the condition. Specify <code>NULL</code> for this parameter if you do not want to retrieve this value.
	<b>schemaList</b>
	The list of form names the escalation is linked to. The escalation must be associated with a single form or a list of forms that currently exists on the server. Specify <code>NULL</code> for this parameter if you do not want to retrieve the list.
	<b>enable</b>
	A flag identifying whether the escalation is disabled (0) or enabled (1). Specify <code>NULL</code> for this parameter if you do not want to retrieve this value.
	<b>query</b>
	A query operation performed when the escalation is executed that determines the set of entries to which the escalation actions (defined by the <code>actionList</code> parameter) are applied. The system returns 0 ( <code>AR_COND_OP_NONE</code> ) if the escalation has no qualification. Specify <code>NULL</code> for this parameter if you do not want to retrieve this value.
	<b>actionList</b>
	The set of actions performed for each entry that matches the criteria defined by the <code>query</code> parameter. This list can contain from 1 to 25 actions (limited by <code>AR_MAX_ACTIONS</code> ). Specify <code>NULL</code> for this parameter if you do not want to retrieve this value.

### elseList

The set of actions performed if no entries match the criteria defined by the query parameter. This list can contain from zero to 25 actions (limited by AR\_MAX\_ACTIONS). Specify NULL for this parameter if you do not want to retrieve this value.

### helpText

The help text associated with the escalation. Specify NULL for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

### timestamp

A time stamp identifying the last change to the escalation. Specify NULL for this parameter if you do not want to retrieve this value.

### owner

The owning user for the escalation. Specify NULL for this parameter if you do not want to retrieve this value.

### lastChanged

The user who made the last change to the escalation. Specify NULL for this parameter if you do not want to retrieve this value.

### changeDiary

The change diary associated with the escalation. The server adds the user making the change and a time stamp when it saves the change diary. Use ARDecodeDi ary to parse the change diary into user, time stamp, and text components. Specify NULL for this parameter if you do not want to retrieve the change diary (which is useful if you are calling this function to verify whether an instance of this object exists).

### objPropList

If the obj PropLi st parameter is set to NULL, no object properties list will be returned. See “Server object property tags” on page 79 for object property tag names and data types.

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateEscalation, ARDecodeDiary, ARDeleteEscalation, ARGetListEscalation, ARGetMultipleEscalations, ARSetEscalation. See FreeAR for: FreeARFilterActionList, FreeARPropList, FreeARQualifierStruct, FreeARStatusList.

## ARGetField

<b>Description</b>	Retrieves information about the form field with the indicated ID on the specified server.
<b>Privileges</b>	This operation can be performed by users with access permission for the field's parent form. Access to permissions information is limited to users with AR System administrator privileges only.

### Synopsis

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

int ARGetField(
    ARControlStruct *control,
    ARNameType schema,
    ARInternalId fieldId,
    ARNameType fieldDName,
    ARFieldMappingStruct *fieldMap,
    *dataType,
    *option,
    *createMode,
    *fieldOption,
    *defaultVal,
    *permissions,
    *limit,
    *instanceList,
    **helpText,
    *timestamp,
    owner,
    lastChanged,
    **changeDir,
    *status)
```

Input arguments	<b>control</b>
	The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The <code>user</code> and <code>server</code> fields are required.
	<b>schema</b>
	The name of the form containing the field to retrieve.
Return values	<b>fieldId</b>
	The internal ID of the field to retrieve.
	<b>fieldName</b>
	The name of the field. Specify <code>NULL</code> for this parameter if you do not want to retrieve the field name.
	<b>fieldMap</b>
	The underlying form from which to retrieve the field (applicable for join forms only). Specify <code>NULL</code> for this parameter if you do not want to retrieve the field mapping.
	<b>dataType</b>
	The data type of the field. See “ <code>ARCreateField</code> ” on page 143 for a description of the possible values. Specify <code>NULL</code> for this parameter if you do not want to retrieve the data type.
	<b>option</b>
	A flag indicating whether users must enter a value in the field. See “ <code>ARCreateField</code> ” on page 143 for a description of the possible values. Specify <code>NULL</code> for this parameter if you do not want to retrieve this flag.
	<b>createMode</b>
	A flag indicating the permission status for the field when users submit entries. See “ <code>ARCreateField</code> ” on page 143 for a description of the possible values. Specify <code>NULL</code> for this parameter if you do not want to retrieve this flag.

## fieldOption

A bitmask that indicates whether the field should be audited or copied when other fields are audited.

- Bit 0: Audit this field. (AR\_FIELD\_BITOPTION\_AUDIT)
- Bit 1: Copy this field when other fields in the form are audited. (AR\_FIELD\_BITOPTION\_COPY)
- Bit 2: Indicates this field is for Log Key 1. (AR\_FIELD\_BITOPTION\_LOG\_KEY1)
- Bit 3: Indicates this field is for Log Key 2. (AR\_FIELD\_BITOPTION\_LOG\_KEY2)
- Bit 2 and 3: Indicates this field is for Log Key 3. (AR\_FIELD\_BITOPTION\_LOG\_KEY3)

## defaultVal

The value to apply if a user submits an entry with no field value (applicable for data fields only). The system returns 0 (AR\_DEFAULT\_VALUE\_NONE) if the field has no default. Specify `NULL` for this parameter if you do not want to retrieve the default value.

## permissions

A list of zero or more groups who can access this field. Access to this information is limited to users with AR System administrator privileges only. Specify `NULL` for this parameter if you do not want to retrieve the permissions.

## limit

The value limits for the field and other properties specific to the field's type. See the `ARFieldListStruct` definition in the `ar.h` file to find the contained structure that applies to the type of field you are creating. Specify `NULL` (or `AR_FIELD_LIST_TYPE_NONE`) for trim and control fields or if you do not want to retrieve the limits and properties.

## dInstanceList

A list of zero or more display properties associated with the field. See “[ARCreateField](#)” on page 143 for a description of the possible values. The system returns 0 (AR\_DPROP\_NONE) if the field has no display properties. Specify `NULL` for this parameter if you do not want to retrieve this list.

## helpText

The help text associated with the field. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

**timestamp**

A time stamp identifying the last change to the field. Specify `NULL` for this parameter if you do not want to retrieve this value.

**owner**

The owning user for the field. Specify `NULL` for this parameter if you do not want to retrieve this value.

**lastChanged**

The user who made the last change to the field. Specify `NULL` for this parameter if you do not want to retrieve this value.

**changeDiary**

The change diary associated with the field. The server adds the user making the change and a time stamp when it saves the change diary. Use `ARDecodeDiary` to parse the change diary into user, time stamp, and text components. Specify `NULL` for this parameter if you do not want to retrieve the change diary (which is useful if you are calling this function to verify whether an instance of this object exists).

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

`ARCreatField`, `ARDecodeDiary`, `ARDeleteField`,  
`ARGetMultipleFields`, `ARGetSchema`, `ARGetListField`, `ARSetField`. See  
FreeAR for: `FreeARDisplayInstanceList`, `FreeARFieldLimitList`,  
`FreeARPermissionList`, `FreeARStatusList`, `FreeARValueStruct`.

# ARGetFilter

**Description** Retrieves information about the specified filter on the specified server.

**Privileges** AR System administrator.

**Synopsis**

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

int ARGetFilter(
    ARControlStruct *control,
    ARNameType name,
    *order,
    *schemaList,
    *opSet,
    *enable,
    *query,
    *actionList,
    *elementList,
    **helpText,
    *timestamp,
    owner,
    lastChanged,
    **changeDir,
    *objPropList,
    *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**name**

The name of the filter to retrieve.

**Return values****order**

A value between 0 and 1000 (inclusive) that determines the filter execution order. When multiple filters are associated with a form, the value associated with each filter determines the order in which they are processed (from lowest to highest). Specify `NULL` for this parameter if you do not want to retrieve this value.

**schemaList**

The list of form names the filter is linked to. The filter must be associated with a single form or a list of forms that currently exists on the server. Specify `NULL` for this parameter if you do not want to retrieve the list.

**opSet**

A bitmask indicating the form operations that trigger the filter. See “[ARCreateFilter](#)” on page 160 for a description of the possible values. Specify `NULL` for this parameter if you do not want to retrieve the operation set.

**enable**

A flag identifying whether the filter is disabled (0) or enabled (1). Specify `NULL` for this parameter if you do not want to retrieve this flag.

**query**

A qualification that determines whether the filter is executed. The system returns 0 (`AR_COND_OP_NONE`) if the filter has no qualification. Specify `NULL` for this parameter if you do not want to retrieve the query.

**actionList**

The set of actions performed if the condition defined by the `query` parameter is satisfied. This list can contain from 1 to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve the action list.

**elseList**

The set of actions performed if the condition defined by the `query` parameter is not satisfied. This list can contain from 0 to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve the else list.

## helpText

The help text associated with the filter. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

## timestamp

A time stamp identifying the last change to the filter. Specify `NULL` for this parameter if you do not want to retrieve this value.

## owner

The owning user for the filter. Specify `NULL` for this parameter if you do not want to retrieve this value.

## lastChanged

The user who made the last change to the filter. Specify `NULL` for this parameter if you do not want to retrieve this value.

## changeDiary

The change diary associated with the filter. The server adds the user making the change and a time stamp when it saves the change diary. Use `ARDecodeDiary` to parse the change diary into user, time stamp, and text components. Specify `NULL` for this parameter if you do not want to retrieve the change diary (which is useful if you are calling this function to verify whether an instance of this object exists).

## objPropList

If the `objPropList` parameter is set to `NULL`, no object properties list will be returned. See “Server object property tags” on page 79 for object property tag names and data types.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

### See also

`ARCreatFilter`, `ARDecodeDiary`, `ARDeleteFilter`, `ARGetListFilter`, `ARGetMultipleFilters`, `ARSetFilter`. See `FreeAR` for: `FreeARFilterActionList`, `FreeARPropList`, `FreeARQualifierStruct`, `FreeARStatusList`.

# ARGetListActiveLink

<b>Description</b>	Retrieves a list of active links on the specified server. You can retrieve all (accessible) active links or limit the list to active links associated with a particular form or modified after a specified time.
<b>Privileges</b>	The system returns information based on the access privileges of the user you specify for the control parameter. All lists, therefore, are limited to active links the user can access.

## Synopsis

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

int ARGetListActiveLink(
    ARControlStruct *control,
    ARNameType schema,
    ARTimestamp changedSince,
    ARPropList *objPropList,
    ARNameList *nameList,
    ARStatusList *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

### schema

The name of the form to retrieve active links for. Specify `NULL` for this parameter to retrieve active links for all forms.

### changedSince

A time stamp that limits the active links retrieved to those modified after the specified time. Specify `0` for this parameter to retrieve active links with any modification time stamp.

### objPropList

List of object properties to search for. Returns all active links that match the object properties, for example, the application owner.

**Return values****nameList**

A list of zero or more (accessible) active links that match the criteria in the `changedSince` and `objPropList` arguments. The system returns a list with zero items if no active links match the specified criteria.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

`ARCreateAlertEvent`, `ARDeleteActiveLink`, `ARDeleteSchema`, `ARGetActiveLink`, `ARGetMultipleActiveLinks`, `ARSetActiveLink`. See `FreeAR` for: `FreeARNameList`, `FreeARStatusList`.

## ARGetListAlertUser

**Description** Retrieves a list of all users that are registered for alerts on the specified server.

**Privileges** All users.

**Synopsis**

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

int ARGetListAlertUser(
    ARControl Struct          *control,
    ARAccessNameList           *userList,
    ARStatusList                *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where the operation is performed, and which session the operation is performed in. The `user` and `server` fields are required.

**Return values****userList**

A list of zero or more user names registered to receive alerts on the specified server. The list returns only one entry for users that register under multiple addresses.

## **status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARRegisterForAlerts. See FreeAR for: FreeARAccessNameList, FreeARStatusList.

# **ARGetListApplicationState**

**Description** Retrieves the list of application states (maintenance, test, or production) that an application on this server can assume. This list is server-dependent.

**Privileges** All users.

### **Synopsis**

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

int ARGetListApplicationState(
    ARControlStruct *control,
    ARNameList *stateNameList,
    ARStatusList *status)
```

**Input arguments**

### **control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**Return values**

### **stateNameList**

The list of states an application on this server can assume. There is one entry in the list for every field name on the AR System Role Mapping from where the field ID resides in the Application State reserved range (2000-2999).

## **status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARSetApplicationState, ARGetApplicationState.

# ARGetListCharMenu

**Description** Retrieves a list of character menus on the specified server. You can retrieve all character menus or limit the list to character menus modified after a specified time.

**Privileges** All users.

## Synopsis

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

```
int ARGetListCharMenu(
    ARControlStruct *control,
    ARTimestamp changedSince,
    ARNameList *formList,
    ARNameList *actLinkList,
    ARPropList *objPropList,
    ARNameList *nameList,
    ARStatusList *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

### changedSince

A time stamp that limits the character menus retrieved to those modified after the specified time. Specify 0 for this parameter to retrieve character menus with any modification time stamp.

### formList

A form name list that limits the character menus retrieved to the ones that are referenced by fields in the form.

### actLinkList

An active link list that limits the character menus retrieved to the ones that have a change field action with the character menus.

**objPropList**

List of object properties to search for. Returns all character menus that match the object properties, for example, the application owner.

**Return values****nameList**

A list of zero or more character menus that match the criteria defined in the changedSi nce, formLi st, actLi nkLi st, and obj PropLi st parameters. The system returns a list with zero items if no character menus match the specified criteria.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

`ARCreateCharMenu`, `ARDeleteCharMenu`, `ARExpandCharMenu`, `ARGetCharMenu`, `ARSetCharMenu`. See `FreeAR` for: `FreeARNameList`, `FreeARStatusList`.

## ARGetListContainer

**Description** Retrieves a list of containers on the specified server. You can retrieve all (accessible) containers or limit the list to containers of a particular type, containers owned by a specified form, or containers modified after a specified time.

**Privileges** The system returns information based on the access privileges of the user you specify for the `control` parameter. All lists, therefore, are limited to containers the user can access.

**Synopsis**

```
#i ncl ude "ar. h"
#i ncl ude "arerrno. h"
#i ncl ude "arextern. h"
#i ncl ude "arstruct. h"

int ARGetLi stContai ner(
    ARControl Struct *control,
    ARTi mestamp changedSi nce,
    ARContai nerTypeLi st *contai nerTypes,
    unsi gned int attri butes,
    ARContai nerOwnerObj Li st *ownerObj Li st,
    ARPropLi st *obj PropLi st,
```

```
ARContainerInfoList           *conList,
ARStatusList                  *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

### changedSince

A time stamp that limits the containers retrieved to those modified after the specified time. Specify 0 for this parameter to retrieve containers with any modification time stamp.

### containerTypes

A list of values indicating the container types to retrieve.

- 0: Retrieve all container types (ARCON\_ALL).
- 1: Retrieve all guide containers (ARCON\_GUIDE).
- 2: Retrieve all application containers (ARCON\_APP).
- 3: Retrieve all packing list containers (ARCON\_PACK).
- 4: Retrieve all filter guide containers (ARCON\_FILTER\_GUIDE).

### attributes

Specify `AR_HIDDEN_INCREMENT` for this parameter to retrieve both visible and hidden containers. Specify `NULL` for this parameter to retrieve only visible containers.

### ownerObjList

A list of the structures that limit the containers retrieved, based on the object that owns them. Specify `NULL` for this parameter to reference all containers.

- 0: Retrieve all globally owned containers (ARCONOWNER\_NONE).
- 1: Retrieve all containers (ARCONOWNER\_ALL).
- 2: Retrieve all containers owned by the specified form (ARCONOWNER\_SCHEMA).

### objPropList

List of object properties to search for. Returns all containers that match the object properties, for example, the application owner.

**Return values****[conList](#)**

A list of zero or more (accessible) containers that match the criteria defined by the `containerTypes`, `ownerObj`, `changedSince`, and `objPropList` parameters. The system returns a list with zero items if no containers match the specified criteria.

**[status](#)**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

`ARCreateContainer`, `ARDeleteContainer`, `ARGetContainer`, `ARGetListAlertUser`, `ARSetContainer`. See `FreeAR` for: `FreeARNameList`, `FreeARStatusList`.

## ARGetListEntry

**Description**

Retrieves a list of form entries on the specified server. The AR System Server returns data from each entry as a string containing the concatenated values of selected fields. In the returned data, the combined length of all specified fields, including separator characters, can be up to 128 bytes (limited by `AR_MAX_SDESC_SIZE`). You can limit the list to entries that match particular conditions by specifying the `qualifier` parameter. `ARGetListEntryWithFields` can be used to return a qualified list of entries formatted as field/value pairs, without the 128 byte limitation.

**Privileges**

The system returns information based on the access privileges of the user you specify for the `control` parameter. All lists, therefore, are limited to entries the user can access (users must have permission for the `entryId` field to access and retrieve entries).

**Synopsis**

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

int ARGetListEntry(
    ARControlStruct *control,
    ARNameType schema,
    ARQualifierStruct *qualifier,
    AREntryListFieldsList *getListFields,
    ARSortList *

```

```

    unsigned int          firstRetrieved,
    unsignd int           maxRetrieved,
    ARBool enable,
    AREntryList list,
    unsignd int           numMatches,
    ARStatusList status)

```

## Input arguments

### **control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

### **schema**

The name of the form to retrieve entries for.

### **qualifier**

A query that determines the set of entries to retrieve. The qualification can include one or more fields and any combination of conditional, relational, and arithmetic (numeric data types only) operations. The system generates an error if the user does not have read permission for a field or a field does not exist. Specify `NULL` or assign an operation value of 0 (`AR_COND_OP_NONE`) to match all form entries.

### **getListFields**

A list of zero or more fields that identifies the data to display in the query list. The list can include any data fields except diary fields and long character fields. The system checks the permissions for each specified field and returns only those fields for which you have read access. The combined length of all specified fields, including separator characters, can be as many as 128 bytes (limited by `AR_MAX_SDESC_SIZE`). Specify `NULL` for this parameter (or zero fields) to return the default query list data for the form (see “[ARCreateSchema](#)” on page 164). The system returns the Short-Description core field if the form has no default query list data.

### **sortList**

A list of zero or more fields that identifies the entry sort order. The system generates an error if you do not have read access on all specified fields. Specify `NULL` for this parameter (or zero fields) to use the default sort order for the form (see “[ARCreateSchema](#)” on page 164). The system sorts the entries in ascending order by `entryId` if the form has no default sort order.

### firstRetrieve

The first entry to retrieve. A value of 0 (AR\_START\_WITH\_FIRST\_ENTRY) represents the first entry and is the default value if no value is set.

### maxRetrieve

The maximum number of entries to retrieve. Use this parameter to limit the amount of data returned if the qualification does not sufficiently narrow the list. Specify 0 (AR\_NO\_MAX\_LIST\_RETRIEVE) to assign no maximum.

### useLocale

A flag indicating whether to search for entries based on the locale. If you specify 1 (TRUE) and the Localize Server option is selected, entries are searched using the locale specified in

AR\_RESERV\_LOCALE\_LOCALIZED\_SCHEMA. If no matches are found for the specified locale, the search becomes less restrictive until a match is found. If you specify 0 (FALSE) or the Localize Server option is cleared, all entries are searched. For information about the Localize Server option, see the *Configuring* guide.

## Return values

### entryList

A list of zero or more (accessible) entries that match the criteria defined by the qualifier parameter. The system returns a list with zero items if no entries match the specified criteria.

### numMatches

The total number of (accessible) entries that match the qualification criteria. This value does not represent the number of entries returned unless the number of matching entries is less than or equal to the maxRetrieve value. Specify NULL for this parameter if you do not want to retrieve this value.

---

**Note:** Performing this count requires additional search time if the number of matching entries is more than the maxRetrieve value. In this case, the cost of completing the search diminishes the performance benefits of retrieving fewer entries.

---

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARGetListEntryWithFields, ARCreateEntry, ARDeleteEntry, ARGetEntry, ARGetEntryStatistics, ARLoadARQualifierStruct, ARMergeEntry, ARSetEntry. See FreeAR for: FreeAREntryListFieldList, FreeAREntryListList, FreeARQualifierStruct, FreeARSortList, FreeARStatusList.

## ARGetListEntryBlocks

**Description** Retrieves a list of blocks of entries from the specified server. Data is returned as a data structure, AREntryListBlock. Entries are encapsulated in the AREntryListBlock data structure and divided into blocks of entries. You call ARGetEntryBlock with a block number to return a list of entries for that block.

**Privileges** The system returns information based on the access privileges of the user you specify for the control parameter. All lists, therefore, are limited to entries the user can access (users must have permission for the entryId field to access and retrieve entries).

**Synopsis**

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

int ARGetListEntryBlock(
    ARControlStruct *control,
    ARNameType schema,
    ARQualifierStruct *qualifier,
    AREntryListFilter *getListFilters,
    ARSortList *sortList,
    unsigned int numRowsPerBlock,
    firstRetrieved,
    maxRetrieved,
    useLocal,
    entryBlockList,
    *numReturnedRows,
    *numMatches,
    *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

## schema

The name of the form to retrieve entries for.

## qualifier

A query that determines the set of entries to retrieve. The qualification can include one or more fields and any combination of conditional, relational, and arithmetic (numeric data types only) operations. The system generates an error if the user does not have read permission for a field or a field does not exist. Specify `NULL` or assign an operation value of 0 (`AR_COND_OP_NONE`) to match all form entries.

## getListFields

A list of zero or more fields to be retrieved with each entry. The list can include any data fields except diary fields and long character fields. The system checks the permissions for each specified field and returns only those fields for which you have read access.

## sortList

A list of zero or more fields that identifies the entry sort order. The system generates an error if you do not have read access on all specified fields. Specify `NULL` for this parameter (or zero fields) to use the default sort order for the form. The system sorts the entries in ascending order by `entryId` if the form has no default sort order.

## numRowsPerBlock

The number of rows per block of data retrieved.

## firstRetrieve

The first entry to retrieve. A value of 0 (`AR_START_WITH_FIRST_ENTRY`) represents the first entry and is the default value if no value is set.

## maxRetrieve

The maximum number of entries to retrieve. Use this parameter to limit the amount of data returned if the qualification does not sufficiently narrow the list. Specify 0 (`AR_NO_MAX_LIST_RETRIEVE`) to assign no maximum.

## useLocale

A flag indicating whether to search for entries based on the locale. If you specify 1 (TRUE) and the Localize Server option is selected, entries are searched using the locale specified in

AR\_RESERV\_LOCALE\_LOCALIZED\_SCHEMA. If no matches are found for the specified locale, the search becomes less restrictive until a match is found. If you specify 0 (FALSE) or the Localize Server option is cleared, all entries are searched. For information about the Localize Server option, see the *Configuring* guide.

### Return values

#### entryBlockList

A list of zero or more (accessible) entries that match the criteria defined by the `qualification` parameter. The system returns a list with zero items if no entries match the specified criteria.

#### numReturnedRows

The total number of returned rows in the block.

#### numMatches

The total number of (accessible) entries that match the qualification criteria. This value does not represent the number of entries returned unless the number of matching entries is less than or equal to the `maxRetrieved` value. Specify `NULL` for this parameter if you do not want to retrieve this value.

---

**Note:** Performing this count requires additional search time if the number of matching entries is more than the `maxRetrieved` value. In this case, the cost of completing the search diminishes the performance benefits of retrieving fewer entries.

---

#### status

A list of zero or more notes, warnings, or errors generated from this function call.

See also ARGetEntryBlock.

# ARGetListEntryWithFields

**Description** Retrieves a list of form entries on the specified server. Data from each entry is returned as field/value pairs for all fields. You can limit the list to entries that match particular conditions by specifying the `qualifier` parameter. `ARGetListEntry` also returns a qualified list of entries, but as an unformatted string with a maximum length of 128 bytes for each entry containing the concatenated values of selected fields.

**Privileges** The system returns information based on the access privileges of the user you specify for the `control` parameter. All lists, therefore, are limited to entries the user can access (users must have permission for the `entryId` field to access and retrieve entries).

## Synopsis

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

int ARGetListEntryWithFields(
    ARControlStruct *control,
    ARNameType schema,
    ARQualifierStruct *qualifier,
    ARGetListFields *getListFields,
    ARSortList *sortList,
    firstRetriever,
    maxRetriever,
    useLocal,
    *entryList,
    *numMatches,
    *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

### schema

The name of the form to retrieve entries for.

## qualifier

A query that determines the set of entries to retrieve. The qualification can include one or more fields and any combination of conditional, relational, and arithmetic (numeric data types only) operations. The system generates an error if the user does not have read permission for a field or a field does not exist. Specify `NULL` or assign an operation value of 0 (`AR_COND_OP_NONE`) to match all form entries.

## getListFields

A list of zero or more fields to be retrieved with each entry. The list can include any data fields except diary fields and long character fields. The system checks the permissions for each specified field and returns only those fields for which you have read access.

## sortList

A list of zero or more fields that identifies the entry sort order. The system generates an error if you do not have read access on all specified fields. Specify `NULL` for this parameter (or zero fields) to use the default sort order for the form (see “`ARCreateSchema`” on page 164). The system sorts the entries in ascending order by `entryId` if the form has no default sort order.

## firstRetrieve

The first entry to retrieve. A value of 0 (`AR_START_WI TH_FI RST_ENTRY`) represents the first entry and is the default value if no value is set.

## useLocale

A flag indicating whether to search for entries based on the locale. If you specify 1 (`TRUE`) and the Localize Server option is selected, entries are searched using the locale specified in

`AR_RESERV_LOCALE_LOCALIZED_SCHEMA`. If no matches are found for the specified locale, the search becomes less restrictive until a match is found. If you specify 0 (`FALSE`) or the Localize Server option is cleared, all entries are searched. For information about the Localize Server option, see the *Configuring* guide.

## maxRetrieve

The maximum number of entries to retrieve. Use this parameter to limit the amount of data returned if the qualification does not sufficiently narrow the list. Specify 0 (`AR_NO_MAX_LIST_RETRIEVE`) to assign no maximum.

**Return values****[entryList](#)**

A list of zero or more (accessible) entries that match the criteria defined by the `qualifier` parameter. The system returns a list with zero items if no entries match the specified criteria.

**[numMatches](#)**

The total number of (accessible) entries that match the qualification criteria. This value does not represent the number of entries returned unless the number of matching entries is less than or equal to the `maxRetrieved` value. Specify `NULL` for this parameter if you do not want to retrieve this value.

---

**Note:** Performing this count requires additional search time if the number of matching entries is more than the `maxRetrieved` value. In this case, the cost of completing the search diminishes the performance benefits of retrieving fewer entries.

---

**[status](#)**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

[ARGetListEntry](#), [ARCreateEntry](#), [ARDeleteEntry](#), [ARGetEntry](#), [ARGetEntryStatistics](#), [ARLoadARQualifierStruct](#), [ARMergeEntry](#), [ARSetEntry](#). See [FreeAR](#) for: [FreeAREntryListFieldList](#), [FreeAREntryListList](#), [FreeARQualifierStruct](#), [FreeARSortList](#), [FreeARStatusList](#).

## ARGetListEscalation

**Description**

Retrieves a list of escalations on the specified server. You can retrieve all escalations or limit the list to escalations associated with a particular form. The call returns all escalations modified on or after the timestamp.

**Privileges**

AR System administrator.

**Synopsis**

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

int ARGetListEscalation(
    ARControlStruct *control,
    ARNameType schema,
    ARTimestamp changedSince,
    *objPropList,
    ARNameList *nameList,
    *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**schema**

The name of the form to retrieve escalations for. Specify `NULL` for this parameter to retrieve escalations for all forms.

**changedSince**

A time stamp that limits the escalation retrieved to those modified after the specified time. Specify 0 for this parameter to retrieve escalations with any modification time stamp.

**objPropList**

List of object properties to search for. Returns all escalations that match the object properties, for example, the application owner.

**Return values****nameList**

A list of zero or more escalations that match the criteria defined in the `changedSince` and `objPropList` arguments. The system returns a list with zero items if no forms match the specified criteria.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateEscalation, ARDeleteEscalation, ARDeleteSchema, ARGetEscalation, ARSetEscalation. See FreeAR for: FreeARNameList, FreeARStatusList.

## ARGetListExtSchemaCandidates

<b>Description</b>	Retrieves a list of all available external data source tables (schema candidates). Users choose fields from these candidates to populate the vendor form.
<b>Privileges</b>	The system returns information based on the access privileges of the user you specify for the <code>control</code> parameter. All lists, therefore, are limited to forms the user can access.

### Synopsis

```
#i ncl ude "ar. h"
#i ncl ude "arerrno. h"
#i ncl ude "arextern. h"
#i ncl ude "arstruct. h"

int ARGetListExtSchemaCandidates(
    ARControlStruct *control,
    unsigned int schemaType,
    ARCompoundSchemaList *schemaList,
    ARStatusList *status)
```

### Input arguments

#### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

#### schemaType

A value indicating the form types to retrieve.

- Bit 7: Retrieve all view schemas (AR\_SCHEMA\_VIEW).
- Bit 8: Retrieve all vendor schemas (AR\_SCHEMA\_VENDOR).

To retrieve both visible and hidden forms, add 1024 (AR\_HIDDEN\_INCREMENT) to the form type. For example, specify AR\_SCHEMA\_VENDOR | AR\_HIDDEN\_INCREMENT for this parameter.

**Return values****[schemaList](#)**

A list of zero or more (accessible) forms that match the criteria in the schemaType parameter. Specify `NULL` for this parameter if you do not want to retrieve the list.

**[status](#)**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

`ARGetMultipleExtFieldCandidates`. See `FreeAR` for: `FreeARCompuondSchemaList`, `AFreeARStatusList`.

## ARGetListField

**Description** Retrieves a list of fields for a particular form on the specified server. You can retrieve all fields or limit the list to fields of a particular type or fields modified after a specified time.

**Privileges** This operation can be performed by users with access permission for the specified form.

**Synopsis**

```
#i ncl ude "ar.h"
#i ncl ude "arerrno.h"
#i ncl ude "arextern.h"
#i ncl ude "arstruct.h"

int ARGetListField(
    ARControl Struct *control,
    ARNameType schema,
    fi el dType,
    changedSi nce,
    *i dLi st,
    *status)
```

**Input arguments**

**[control](#)**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**[schema](#)**

The name of the form to retrieve fields for.

## fieldType

A bitmask indicating the field types to retrieve.

- Bit 0: Retrieve data fields (AR\_FIELD\_TYPE\_DATA).
- Bit 1: Retrieve trim fields (AR\_FIELD\_TYPE\_TRIM).
- Bit 2: Retrieve control fields (AR\_FIELD\_TYPE\_CONTROL).
- Bit 3: Retrieve page fields (AR\_FIELD\_TYPE\_PAGE).
- Bit 4: Retrieve page holder fields (AR\_FIELD\_TYPE\_PAGE HOLDER).
- Bit 5: Retrieve table fields (AR\_FIELD\_TYPE\_TABLE).
- Bit 6: Retrieve column fields (AR\_FIELD\_TYPE\_COLUMN).
- Bit 8: Retrieve vendor type fields (AR\_FIELD\_TYPE\_VENDOR).
- Bit 128: Retrieve attachment type fields (AR\_FIELD\_TYPE\_ATTACH).
- Bit 256: Retrieve attachment pool type fields (AR\_FIELD\_TYPE\_ATTACH\_POOL).

## changedSince

A time stamp that limits the fields retrieved to those modified after the specified time. Specify 0 for this parameter to retrieve fields with any modification time stamp.

## Return values

### idList

A list of zero or more fields that match the criteria defined by the field type and changedSince parameters. The system returns a list with zero items if no fields match the specified criteria.

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateField, ARDeleteField, ARGetField, ARGetMultipleFields, ARSetField. See FreeAR for: FreeARInternalIdList, FreeARStatusList.

# ARGetListFilter

Description	Retrieves a list of filters on the specified server. You can retrieve all filters or limit the list to filters associated with a particular form or modified after a specified time.
-------------	--

**Privileges** AR System administrator.

### Synopsis

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

int ARGetListFilter(
    ARControlStruct *control,
    ARNameType schema,
    ARTimestamp changedSince,
    *objPropList,
    ARNameList *nameList,
    *status)
```

### Input arguments

#### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

#### schema

The name of the form to retrieve filters for. Specify `NULL` for this parameter to retrieve filters for all forms.

#### changedSince

A time stamp that limits the filters retrieved to those modified after the specified time. Specify 0 for this parameter to retrieve filters with any modification time stamp.

#### objPropList

List of object properties to search for. Returns all filters that match the object properties, for example, the application owner.

### Return values

#### nameList

A list of zero or more (accessible) filters that match the criteria defined by the `changedSince` and `objPropList` parameters. The system returns a list with zero items if no filters match the specified criteria.

#### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

See also ARCreateFilter, ARDeleteFilter, ARDeleteSchema, ARGetFilter, ARSetFilter.  
See FreeAR for: FreeARNameList, FreeARStatusList.

## ARGetListGroup

**Description** Retrieves a list of access control groups on the specified server. You can retrieve all groups or limit the list to groups associated with a particular user.

**Privileges** Group information for the current user can be retrieved by all users. Access to group information for other users is limited to users with AR System administrator privileges only.

### Synopsis

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

int ARGetListGroup(
    ARControlStruct *control,
    ARAccessNameType,
    ARAccessNameType,
    ARGroupInfoList *groupList,
    ARStatusList *status)
```

### Input arguments

#### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

#### userName

The name of the user for which the system retrieves group information. Specify `NULL` for this parameter to retrieve all groups on the server.

#### password

The password of the user for whom the system retrieves the list. If this parameter is `NULL`, the call returns the group list for the first user whose name matches the name passed in, not necessarily the user who requested the list. If this parameter specifies a password, the server returns the group list for the user whose name and password match those in the parameter list.

**Return values****groupList**

A list of zero or more groups that match the criteria defined by the `userName` parameter. Each item in the list contains a group ID, the name associated with that ID, group category, and the computed group qualification string. The system returns a list with zero items if no groups match the specified criteria.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARGetListUser. See FreeAR for: FreeARGroupInfoList, FreeARStatusList.

## ARGetListLicense

**Description** Retrieves a list of entries from the license file for the correct server. The list contains license information for all the types of licenses, including DSO, flashboards, SMU applications, and servers.

**Privileges** AR System administrator.

**Synopsis**

```
#i ncl ude "ar.h"
#i ncl ude "arerrno.h"
#i ncl ude "arextern.h"
#i ncl ude "arstruct.h"

int ARGetListLicense(
    ARControl Struct *control,
    ARLicenseNameType licenseType,
    ARLicenseInfoList *licenseInfoList,
    ARStatusList *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**licenseType**

The type of license to retrieve. If the value of this argument is `NULL`, the call retrieves all the license types that are available.

**Return values****[licenseInfoList](#)**

A list of licenses that fit the designated criteria and information about those licenses, such as license key, license type, expiration date, and the license host for the new license.

**[status](#)**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

[ARCreateLicense](#), [ARDeleteLicense](#), [ARValidateLicense](#), [ARValidateMultipleLicenses](#).

## ARGetListRole

**Description** Retrieves a list of roles for a deployable application or returns a list of roles for a user for a deployable application.

**Privileges** Role information for the current user can be retrieved by all users. Access to role information for other users is limited to users with AR System administrator privileges only.

**Synopsis**

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

int ARGetListRole(
    ARControlStruct *control,
    ARNameType applicationName,
    ARAccessNameType userName,
    ARAccessNameType password,
    ARRoleList *roleList,
    ARStatusList *status)
```

**Input arguments**

**[control](#)**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**[applicationName](#)**

The name of the specific application to get roles from.

**userName**

The name of the specific user for which the system retrieves roles information. Specify `NULL` to retrieve all roles for the application.

**password**

The password of the specific user for whom the system retrieves the list. If this parameter is `NULL`, the call returns the role list for the first user whose name matches the name passed in, not necessarily the user who requested the list. If this parameter specifies a password, the server returns the role list for the user whose name and password match those in the parameter list.

**Return values****roleList**

A list of zero or more roles that match the criteria defined by the `userName` parameter. Each item in the list contains a role ID, the name associated with that ID, and the group mapping. The system returns a list with zero items if no roles match the specified criteria.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARGetListGroup.

## ARGetListSchema

**Description** Retrieves a list of forms on the specified server. You can retrieve all (accessible) forms or limit the list to forms of a particular type or forms modified after a specified time.

**Privileges** The system returns information based on the access privileges of the user you specify for the `control` parameter. All lists, therefore, are limited to forms the user can access.

**Synopsis**

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

int ARGetListSchema(
    ARControlStruct *control,
    ARTimestamp changedSince,
```

```

unsi gned i nt          schemaType,
ARNameType               name,
ARI nternal I dLi st    *fi el dI dLi st,
ARPropLi st              *obj PropLi st,
ARNameLi st              *nameLi st,
ARStatusLi st            *status)

```

## Input arguments

### **control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

### **changedSince**

A time stamp that limits the forms retrieved to those modified after the specified time. Specify 0 for this parameter to retrieve forms with any modification time stamp.

### **schemaType**

A value indicating the form types to retrieve (some types are mutually exclusive).

- 0: Retrieve all form types (`AR_LIST_SCHEMA_ALL`).
- 1: Retrieve all base forms (`AR_LIST_SCHEMA_REGULAR`).
- 2: Retrieve all join forms (`AR_LIST_SCHEMA_JOIN`).
- 3: Retrieve all view forms (`AR_LIST_SCHEMA_VIEW`).
- 4: Retrieve all join forms that depend on the form specified by the `name` parameter (`AR_LIST_SCHEMA_UPLINK`).
- 5: Retrieve all base or join forms that the form specified by the `name` parameter depends on (`AR_LIST_SCHEMA_DOWNLINK`).
- 6: Retrieve all display-only forms (`AR_LIST_SCHEMA_DISPLAY`).
- 7: Retrieve all forms with data fields (`AR_LIST_SCHEMA_ALL_WITH_DATA`).
- 8: Retrieve all vendor schemas (`AR_LIST_SCHEMA_VENDOR`).

To retrieve both visible and hidden forms, add 1024 (`AR_HIDDEN_INCREMENT`) to the form type. For example, specify `AR_LIST_SCHEMA_ALL | AR_HIDDEN_INCREMENT` for this parameter.

## name

If the schemaType is AR\_LIST\_SCHEMA\_UPLINK, this parameter specifies the form upon which the retrieved forms depend. If the schemaType is AR\_LIST\_SCHEMA\_DOWNLINK, this parameter specifies the form that depends on the retrieved forms. This parameter is ignored if you do not specify AR\_LIST\_SCHEMA\_UPLINK or AR\_LIST\_SCHEMA\_DOWNLINK as the schemaType.

## fieldIdList

List of form fields. The system only returns the forms that contain all the fields in this list. Specify `NULL` for this parameter (or zero fields) if you do not want to qualify the forms returned by the fields they contain. For example, specify the four reserved server event fields (800, 801, 802, 803) in this list to retrieve the server event schema.

---

**Note:** Archive forms are not returned if you specify a list of form fields. You must specify `NULL` to return archive forms.

---

## objPropList

List of object properties to search for. Returns all schemas that match the object properties, for example, the application owner.

## Return values

### nameList

A list of zero or more (accessible) forms that match the criteria in the schemaType, changedSince, fieldIdList, and objPropList parameters. The system returns a list with zero items if no forms match the specified criteria.

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## See also

ARCCreateSchema, ARDeleteSchema, ARGetListSchemaWithAlias, ARGetSchema, ARSetSchema. See FreeAR for: FreeARNameList, FreeARStatusList, FreeARInternalIDList.

# ARGetListSchemaWithAlias

<b>Description</b>	Retrieves a list of form definitions and their corresponding aliases on the specified server. You can retrieve all (accessible) forms or limit the list to forms of a particular type or forms modified after a specified time.
<b>Privileges</b>	The system returns information based on the access privileges of the user you specify for the <code>control</code> parameter. All lists, therefore, are limited to forms the user can access.

## Synopsis

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

int ARGetListSchemaWithAlias(
    ARControlStruct *control,
    ARTimestamp changedSince,
    SchemaType schemaType,
    const char *name,
    *fileList,
    const char *vuiLabel,
    *objPropList,
    *nameList,
    *aliasList,
    *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

### changedSince

A time stamp that limits the forms retrieved to those modified after the specified time. Specify 0 for this parameter to retrieve schemas with any modification time stamp.

## schemaType

A value indicating the form types to retrieve (some types are mutually exclusive).

- 0: Retrieve all form types (AR\_LIST\_SCHEMA\_ALL).
- 1: Retrieve all base forms (AR\_LIST\_SCHEMA\_REGULAR).
- 2: Retrieve all join forms (AR\_LIST\_SCHEMA\_JOIN).
- 3: Retrieve all view forms (AR\_LIST\_SCHEMA\_VIEW).
- 4: Retrieve all join forms that depend on the form specified by the name parameter (AR\_LIST\_SCHEMA\_UPLINK).
- 5: Retrieve all base or join forms that the form specified by the name parameter depends on (AR\_LIST\_SCHEMA\_DOWNLINK).
- 6: Retrieve all display-only forms (AR\_LIST\_SCHEMA\_DISPLAY).
- 7: Retrieve all forms with data fields (AR\_LIST\_SCHEMA\_ALL\_WITH\_DATA).
- 8: Retrieve all vendor schemas (AR\_SCHEMA\_VENDOR).

To retrieve both visible and hidden forms, add 1024 (AR\_INCREMENT) to the form type. For example, specify AR\_LIST\_SCHEMA\_ALL | AR\_INCREMENT for this parameter.

## name

If the schemaType is AR\_LIST\_SCHEMA\_UPLINK, this parameter specifies the form upon which the retrieved forms depend. If the schemaType is AR\_LIST\_SCHEMA\_DOWNLINK, this parameter specifies the form that depends on the retrieved forms. This parameter is ignored if you do not specify AR\_LIST\_SCHEMA\_UPLINK or AR\_LIST\_SCHEMA\_DOWNLINK as the schemaType.

## fieldIdList

A list of zero or more internal IDs specifying the fields to retrieve. Specify NULL for this parameter (or zero fields) if you do not want to qualify the forms returned by the fields they contain. For example, specify the four reserved server event fields (800, 801, 802, 803) in this list to retrieve the server event schema.

## vuiLabel

Label for the specific VUI from which to retrieve aliases. The system returns aliases from the default VUI if you specify NULL or if the specified VUI does not exist.

## [objPropList](#)

List of object properties to search for. Returns all schemas that match the object properties, for example, the application owner.

### Return values

#### [nameList](#)

A list of zero or more (accessible) schemas that match the criteria defined in the `SchemaType`, `changedSince`, `filedList`, and `objPropList` parameters. The system returns a list with zero items if no forms match the specified criteria.

#### [aliasList](#)

A list of zero or more VUI aliases that:

- Match the schemas in the `nameList` parameter.
- Are aliases from the form that the `name` parameter specifies.
- Conform to the locale that the `control` parameter specifies.

#### [status](#)

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

### See also

[ARCreateSchema](#), [ARDelSchema](#), [ARGetListSchema](#), [ARGetSchema](#), [ARSetSchema](#). See [FreeAR](#) for: `FreeARInternalIdList`, `FreeARNameList`, `FreeARStatusList`.

# ARGetListServer

**Description** Retrieves the list of available AR System servers defined in the `ar` directory file (UNIX only). BMC Remedy User, BMC Remedy Administrator, BMC Remedy Alert, and BMC Remedy Import connect to these servers automatically if no servers are specified at startup. If the `ar` file is under NIS control, the system uses the file specified by the NIS map instead of the local `ar` file. For information about the `ar` file, see the *Configuring* guide.

---

**Note:** In the Windows API, server information is retrieved from the registry instead of the `ar` file. API programs that run on the server (for example, through a filter or escalation) can use this function to retrieve the name of that local server only. Programs that run on a Windows client, however, cannot. In this case, the function always returns a list of zero servers.

---

**Privileges** All users.

**Synopsis**

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

int ARGetListServer(
    ARControlStruct          *control,
    ARServerNameList          *serverList,
    ARStatusList               *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**Return values**

**serverList**

A list of zero or more registered AR System servers. The system returns a list with zero items if no AR System servers are registered.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

See also [ar](#) directory file. See FreeAR for: FreeARServerNameList, FreeARStatusList.

## ARGetListSQL

**Description** Retrieves a list of rows from the underlying SQL database on the specified server. The server executes the SQL command you specify and returns the matching rows. A list with zero items and a warning message are returned if no SQL database resides on the server. The system returns information based on the access privileges of the user who launched the AR System server.

**Privileges** AR System administrator.

**Synopsis**

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

int ARGetListSQL(
    ARControlStruct *control,
    char *sqlCommand,
    unsigned int maxRetrieve,
    ARValueListList *valueListList,
    unsigned int numMatches,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**sqlCommand**

The SQL command to execute (following the syntax rules for the underlying database). The owner of the AR System server process must have permission to perform the specified SQL operation.

**maxRetrieve**

The maximum number of rows to retrieve. Use this parameter to limit the amount of data returned if the SQL query does not sufficiently narrow the list. Specify 0 (`AR_NO_MAX_LIST_RETRIEVE`) to assign no maximum.

**Return values****[valueListList](#)**

A list of zero or more (accessible) rows that match the criteria defined by the `sqlCommand` parameter. Each item in the list represents one matching row, each of which contains a list of the selected column values. The system returns a list with zero items if no rows match the specified criteria.

**[numMatches](#)**

The total number of (accessible) rows that match the SQL selection criteria. This value does not represent the number of rows returned unless the number of matching rows is less than or equal to the `maxRetrieve` value. Specify `NULL` for this parameter if you do not want to retrieve this value.

**Note:** Performing this count requires additional search time if the number of matching rows is more than the `maxRetrieve` value. In this case, the cost of completing the search diminishes the performance benefits of retrieving fewer rows.

**[status](#)**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** `ARExecuteProcess`, `ARGetListEntry`. See `FreeAR` for: `FreeARStatusList`, `FreeARValueListList`.

## ARGetListSupportFile

**Description** Retrieves a list of support file IDs for a specified type of object.

**Privileges** Any user who has access to the specified object.

**Synopsis**

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

int ARGetListSupportFile(
    ARControlStruct *control,
    FileType,
    name,
```

	ARI nternal Id ARTi mestamp ARI nternal IdLi st ARStatusLi st	i d2, changedSi nce, *fi l el dLi st, *status)
<b>Input arguments</b>	<b>control</b>	The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.
	<b>fileType</b>	The numerical value for the type of file, and the type of object the file is related to. Specify 1 (AR_SUPPORT_FI LE_EXTERNAL_REPORT) for an external report file associated with an active link.
	<b>name</b>	The name of the object the file is associated with, usually a form.
	<b>id2</b>	The ID of the field or VUI, if the object is a form. If the object is not a form, set this parameter to 0.
	<b>changedSince</b>	A time stamp that limits the IDs retrieved to those modified after the specified time. Specify 0 for this parameter to retrieve all IDs.
<b>Return values</b>	<b>fileIdList</b>	A list of support file IDs linked to an object.
	<b>status</b>	A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the <i>Error Messages</i> guide.
<b>See also</b>	ARCCreateAlertEvent, ARCreateSupportFile, ARDeleteActiveLink, ARDeleteSupportFile, ARGetActiveLink, ARGetSupportFile, ARSetActiveLink, ARSetSupportFile.	

# ARGetListUser

**Description** Retrieves a list of users on the specified AR System server. You can retrieve information about the current user, all registered users, or all users currently accessing the server.

**Privileges** Information about the current user can be retrieved by all users. Access to information about other users is limited to users with AR System administrator privileges only.

## Synopsis

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

int ARGetListUser(
    ARControlStruct *control,
    unsigned int userListType,
    ARTimestamp changedSince,
    ARUserList *userList,
    ARStatusList *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

### userListType

A value indicating the user type to retrieve.

- 0: Retrieve the current user (AR\_USER\_LIST\_MYSELF).
- 1: Retrieve all registered users (AR\_USER\_LIST\_REGISTERED).
- 2: Retrieve all users currently accessing the server (AR\_USER\_LIST\_CURRENT).
- 3: Retrieve all users with accounts marked invalid (AR\_USER\_LIST\_INVALID).
- 4: Retrieve all users with application licenses (AR\_LISTENCE\_TAG\_APP).

### changedSince

A time stamp that limits the list entries retrieved to those modified after the specified time. Specify 0 for this parameter to retrieve entries with any modification time stamp. The `changedSince` parameter only applies if the `userListType` parameter is set to: AR\_USER\_LIST\_REGISTERED.

**Return values****[userList](#)**

List of zero or more users that match the criteria defined by the `userListType` parameter. The list contains the user name and license type. If you retrieve the current user, the list also contains a time stamp identifying the last server access. The system returns a list with zero items if no users match the specified criteria.

**[status](#)**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** [ARGetListGroup](#). See FreeAR for: [FreeARUserInfoList](#), [FreeARStatusList](#).

## ARGetListVUI

**Description** Retrieves a list of form views (VUI) for a particular form on the specified server. You can retrieve all views or limit the list to views modified after a specified time.

**Privileges** This operation can be performed by users with access permission for the specified form.

**Synopsis**

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

```
int ARGetListVUI (
    ARControlStruct *control,
    ARNameType schema,
    ARTimestamp changedSince,
    *internalList,
    ARStatusList *status)
```

**Input arguments**

**[control](#)**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**[schema](#)**

The name of the form to retrieve views for.

## changedSince

A time stamp that limits the views retrieved to those modified after the specified time. Specify 0 for this parameter to retrieve views with any modification time stamp.

### Return values

#### **idList**

A list of zero or more view IDs that match the criteria defined by the changedSince parameter. The system returns a list with zero items if no views match the specified criteria.

#### **status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateVUI, ARDeleteVUI, ARGetVUI, ARSetVUI. See FreeAR for: FreeARInternalIdList, FreeARStatusList.

## ARGetLocalizedValue

**Description** Retrieves a localized text string from the BMC Remedy Message Catalog. The message that the server retrieves depends on the user locale.

**Privileges** All users.

### Synopsis

```
#i ncl ude "ar.h"
#i ncl ude "arerrno.h"
#i ncl ude "arextern.h"
#i ncl ude "arstruct.h"
```

```
int ARGetLocal i zedVal ue(
    ARControl Struct *control,
    ARLocal i zedRequestStruct *l ocal i zedRequest,
    ARVal ueStruct *l ocal i zedVal ue,
    ARTi mestamp *ti mestamp,
    ARStatusLi st *status)
```

### Input arguments

#### **control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

## localizedRequest

Identification for the localized value to be retrieved from the BMC Remedy Message Catalog. It contains the type of value, value name, and, for some types of values, unique identification numbers.

### Return values

#### localizedValue

The localized value to be retrieved. The value can be a character value or an attachment. Specify `NULL` for this argument if you do not want to retrieve this value.

#### timestamp

A time stamp identifying the last change to the value. Specify `NULL` for this parameter if you do not want to retrieve this value.

#### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

### See also

`ARGetMultipleLocalizedValues`. See `FreeAR` for: `FreeARValueStruct`, `FreeARStatusList`.

## ARGetMultipleActiveLinks

### Description

Retrieves multiple active link definitions. This function performs the same action as `ARGetActiveLinks` but is easier to use and more efficient than retrieving multiple entries one by one.

Information is returned in lists for each item, with one item in the list for each active link returned. For example, if the second item in the list for `existLi st` is `TRUE`, the name of the second active link is returned in the second item in the list for `actLi nkNameLi st`.

### Privileges

All users who have permission for the active link. Only AR System administrators have access to `groupLi st` information.

**Synopsis**

```
#i ncl ude "ar.h"
#i ncl ude "arerrno.h"
#i ncl ude "arextern.h"
#i ncl ude "arstruct.h"

int ARGetMul ti pl eActi veLi nks(
    ARControl Struct
    ARTi mestamp
    ARNameLi st
    ARBool eanLi st
    ARNameLi st
    ARUnsi gnedI ntLi st
    ARWorkfl owConnectLi st
    ARI nternal l dLi stLi st
    ARUnsi gnedI ntLi st
    ARI nternal l dLi st
    ARI nternal l dLi st
    ARUnsi gnedI ntLi st
    ARQual i fi erLi st
    ARActi veLi nkActi onLi stLi st
    ARActi veLi nkActi onLi stLi st
    ARTextStri ngLi st
    ARTi mestampLi st
    ARAccessNameLi st
    ARAccessNameLi st
    ARTextStri ngLi st
    ARPropLi stLi st
    ARStatusLi st
    *control ,
    changedSi nce
    *nameLi st,
    *exi stLi st,
    *acti onLi nkNameLi st,
    *orderLi st,
    *schemaLi st,
    *groupLi stLi st,
    *executeMaskLi st,
    *control Fi el dLi st,
    *focusFi el dLi st,
    *enabl eLi st,
    *queryLi st,
    *acti onLi stLi st,
    *el seLi stLi st,
    *hel pTextLi st,
    *ti mestampLi st,
    *ownersLi st,
    *lastChangedLi st,
    *changeDi aryLi st,
    *obj PropLi stLi st,
    *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**changedSince**

A time stamp that limits the active links retrieved to those modified after the specified time. Specify 0 for this parameter to retrieve active links with any modification time stamp.

## nameList

A list of names of the active links to retrieve that match the criteria in the changedSince argument. The system returns a list with zero items if no active links match the specified criteria.

---

**Note:** If you enter the same active link name two or more times in the nameList parameter, the server only retrieves information for the first occurrence, and ignores the information for the other repeat occurrences.

---

### Return values

#### existList

A list of flags and corresponding Boolean values indicating whether the active links exist and meet the qualifying criteria. Values: `True` = active link exists; `False` = active link does not exist.

#### actLinkNameList

A list of names of active links that this call returns. This argument returns names in the same order as the input list of active links to be retrieved. If namelist is not `NULL`, each return list maintains a corresponding position for each supplied name. As a consequence, you must see the existList returned to see if the active link in that position had any data returned for it.

#### orderList

A value between 0 and 1000 (inclusive) that determines the active link execution order. When multiple active links are associated with a form, the value associated with each active link determines the order in which they are processed (from lowest to highest). Specify `NULL` for this parameter if you do not want to retrieve this value.

#### schemaList

The list of form names the active link is linked to. The active link must be associated with a single form or a list of forms that currently exists on the server. Specify `NULL` for this parameter if you do not want to retrieve this value.

#### groupListList

A list of zero or more groups who can access these active links. Access to this information is limited to users with AR System administrator privileges only. Specify `NULL` for this parameter if you do not want to retrieve this value.

## executeMaskList

A list of bitmasks indicating the form operations that trigger the active links. See “ARCreateActiveLink” on page 128 for a description of the possible values. Specify `NULL` for this parameter if you do not want to retrieve this value.

## controlFieldList

A list of the IDs of the fields that represent the button, toolbar button, or menu items associated with executing the active links. The system returns zero if the `executeMask` does not include the `AR_EXECUTE_ON_BUTTON` condition. Specify `NULL` for this parameter if you do not want to retrieve this value.

## focusFieldList

A list of the IDs of the fields associated with executing the active links by pressing Return or selecting a character menu item. The system returns zero if the `executeMask` does not include the `AR_EXECUTE_ON_RETURN` or `AR_EXECUTE_ON_MENU_CHOICE` conditions. Specify `NULL` for this parameter if you do not want to retrieve the focus fields.

## enableList

A list of flags identifying whether the active link is disabled (0) or enabled (1). Specify `NULL` for this parameter if you do not want to retrieve these flags.

## queryList

A list of qualifications that determines whether the active links are executed. The system returns zero (`AR_COND_OP_NONE`) if the active links have no qualifications. Specify `NULL` for this parameter if you do not want to retrieve the queries.

## actionListList

A list of the sets of actions performed if the condition defined by the `query` parameter is satisfied. This list can contain from 1 to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve the action lists.

## elseListList

A list of the sets of actions performed if the conditions defined by the `query` parameter is not satisfied. This list can contain from 0 to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve the else list.

## helpTextList

A list of the help texts associated with the active links. Specify `NULL` for this parameter if you do not want to retrieve the help texts (which is useful if you are calling this function to verify whether instances of these objects exist).

## timestampList

A list of time stamps that identify the last changes to the active links. Specify `NULL` for this parameter if you do not want to retrieve this value.

## ownerList

A list of owners for the active links. Specify `NULL` for this parameter if you do not want to retrieve the owners.

## lastChangedList

A list of users who made the last changes to the active links. Specify `NULL` for this parameter if you do not want to retrieve this value.

## changeDiaryList

A list of the change diaries associated with the active link. Use `ARDecodeDiary` to parse the change diary into user, time stamp, and text components. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to retrieve the change diary (which is useful if you are calling this function to verify whether an instance of this object exists).

## objPropList

If the `objPropList` parameter is set to `NULL`, an object properties list with zero properties will be associated with the object, and when an `ARGetFirst` action is performed, zero properties are returned. See “Server object property tags” on page 79 for object property tag names and data types.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateActiveLink, ARDeleteActiveLink, ARGetActiveLink, ARGetListActiveLink, ARGetMultipleEntries, ARGetMultipleExtFieldCandidates, ARGetMultipleFields, ARGetMultipleLocalizedValues, ARGetMultipleSchemas, ARSetActiveLink. See FreeAR for: FreeARAccessNameList, FreeARActiveLinkActionListList, FreeARBooleanList, FreeARInternalIdList, FreeARInternalIdListList, FreeARNameList, FreeARPropListList, FreeARQualifierList, FreeARStatusList, FreeARTextStringList, FreeARTimestampList, FreeARUnsignedIntList, FreeARWorkflowConnectList.

## ARGetMultipleCharMenus

**Description** Retrieves information about a group of character menus on the specified server with the names specified by the `nameList` parameter. This function performs the same action as `ARGetCharMenu` but is easier to use and more efficient than retrieving multiple entries one by one.

Information is returned in lists for each item, with one item in the list for each menu returned. For example, if the second item in the list for `existList` is TRUE, the name of the second character menu is returned in the second item in the list for `charMenuNameList`.

**Privileges** All users.

### Synopsis

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

int ARGetMultipleCharMenus(
    ARControlStruct*control,
    ARTimestamp changedSince,
    *nameList,
    *existList,
    *charMenuNameList,
    *refreshCodeList,
    *menuDefnList,
    *helpTextList,
    *timestampList,
    *ownerList,
    *lastChangedList,
    *changeDictionaryList,
```

`int ARGetMultipleCharMenus(`

- `ARControlStruct`
- `ARTimestamp`
- `ARNameList`
- `ARBoolList`
- `ARNameList`
- `ARUnsignedIntList`
- `ARCharMenuStructList`
- `ARTextStringList`
- `ARTimestampList`
- `ARAccessNameList`
- `ARAccessNameList`
- `ARTextStringList`

- `*control,`
- `changedSince,`
- `*nameList,`
- `*existList,`
- `*charMenuNameList,`
- `*refreshCodeList,`
- `*menuDefnList,`
- `*helpTextList,`
- `*timestampList,`
- `*ownerList,`
- `*lastChangedList,`
- `*changeDictionaryList,`

```
ARPropListLi st  
ARStatusLi st  
*obj PropListLi st,  
*status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

### changedSince

A time stamp that limits the character menus retrieved to those modified after the specified time. Specify 0 for this parameter to retrieve character menus with any modification time stamp.

### nameList

The names of the character menus to retrieve. The nameList can be passed as a NULL or as an empty list, as in:

```
ARNameList emptyNameList = {0, 0};  
...  
ARGetMultiplerCharMenus(control, changedSince, &emptyNameList, ...)
```

In this case information is returned for every character menu that passes the changedSince criterion.

## Return values

### existList

A list of flags and corresponding Boolean values indicating whether the character menus exist. Values: True = character menu exists; False = character menu does not exist.

### charMenuNameList

A list of character menu names retrieved.

### refreshCodeList

A value indicating when the list of character menus is refreshed. See “ARCreateCharMenu” on page 134 for a description of the possible values. Specify NULL for this parameter if you do not want to retrieve this value.

### menuDefnList

A list of definitions of character menus. Specify NULL for this parameter if you do not want to retrieve this value.

## helpTextList

A list of help text items associated with the character menus. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of these objects exists).

## timeStampList

A list of time stamps identifying the last change to the character menus. Specify `NULL` for this parameter if you do not want to retrieve this value.

## ownerList

A list of owning users for the character menus. Specify `NULL` for this parameter if you do not want to retrieve this value.

## lastChangedList

A list of users who made the last change to the character menus. Specify `NULL` for this parameter if you do not want to retrieve this value.

## changeDiaryList

A list of change diaries associated with the character menus. Use `ARDecodeDiary` to parse the change diary into user, time stamp, and text components. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to retrieve the change diary list (which is useful if you are calling this function to verify whether an instance of this object exists).

## objPropListList

A list of object properties. If the `objPropListList` parameter is set to `NULL`, no object properties will be returned. See “Server object property tags” on page 79 for object property tag names and data types.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

### See also

`ARDeleteCharMenu`, `ARExpandCharMenu`, `ARGetCharMenu`, `ARGetListCharMenu`, `ARSetCharMenu`. See `FreeAR` for: `FreeARCharMenuStruct`, `FreeARStatusList`, `FreeARPropList`.

# ARGetMultipleContainers

**Description** Retrieves multiple container objects.

Information is returned in lists for each item, with one item in the list for each container returned. For example, if the second item in the list for `existList` is TRUE, the name of the second container is returned in the second item in the list for `containerNameList`.

**Privileges** All users.

**Synopsis**

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

int ARGetMultipleContainers (
    ARControlStruct *control,
    ARTimeStamp changedSince,
    ARNameList *nameList,
    ARContainerTypeList *containerTypes,
    Attributes *attributes,
    ARObjectList *ownerObjList,
    ARRefTypes *refTypes,
    ARExistList *existList,
    ARContainerNameList *containerNameList,
    ARGroupListList *groupListList,
    ARAdminGroupListList *adminGroupListList,
    ARObjectListList *ownerObjListList,
    ARLabelList *labelList,
    ARDescriptionList *descriptionList,
    ARTypeList *typeList,
    ARReferenceList *referenceList,
    ARHelpTextList *helpTextList,
    AROwnerList *ownerList,
    ARTimestampList *timestampList,
    ARLastChangedList *lastChangedList,
    ARChangeDiryList *changeDiryList,
    ARObjectPropListList *objPropListList,
    ARStatusList *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**changedSince**

A time stamp that limits the containers retrieved to those modified after the specified time. Specify 0 for this parameter to retrieve containers with any modification time stamp.

**nameList**

The names of the containers to retrieve.

**containerTypes**

A list of values indicating the container types to retrieve.

- 0: Retrieve all container types (ARCON\_ALL).
- 1: Retrieve all guide containers (ARCON\_GUIDE).
- 2: Retrieve all application containers (ARCON\_APP).
- 3: Retrieve all packing list containers (ARCON\_PACK).
- 4: Retrieve all filter guide containers (ARCON\_FILTER\_GUIDE).

**attributes**

Specify `AR_HIDDEN_INCREMENT` for this parameter to retrieve both visible and hidden containers. Specify `NULL` for this parameter to retrieve only visible containers.

**ownerObjList**

A list of schemas that own this container. This parameter can be `NULL` if the container exists globally.

**refTypes**

A list of the types of references (for example, forms and filters) to retrieve. You can specify individual types of references to retrieve, specify that all (`ARREF_ALL`) or none (`ARREF_NONE`) of the references be retrieved, or specify negative numbers to treat the types of references as exclude reference types. The exclude reference types take precedence over the include list; this means that if you specify a type as positive as well as negative, then all references of that type are excluded.

**Return values****[existList](#)**

A list of flags and corresponding Boolean values indicating whether the containers exist. Values: `True` = container exists; `False` = container does not exist.

**[containerNameList](#)**

The array of container names retrieved.

**[groupListList](#)**

A list of zero or more groups with permission for the container. Access to this information is limited to users with AR System administrator privileges only. Specify `NULL` for this parameter if you do not want to retrieve this value.

**[admingrpListList](#)**

A list of zero or more groups who can administer this container (and the referenced objects). If `ownerObj` does not return `NULL`, this list is the Subadministrator group list of the owning form. Users must belong to both a specified group *and* the Subadministrator group to obtain these privileges. Specify `NULL` for this parameter if you do not want to retrieve the administrator group list.

**[ownerObjListList](#)**

A list of schemas that own this container. Specify `NULL` for this parameter if you do not want to retrieve this value. If this parameter returns `NULL`, the container exists globally.

**[labelList](#)**

A list of labels for this container. Specify `NULL` for this parameter if you do not want to retrieve the list.

**[descriptionList](#)**

The list of descriptions for this container. Specify `NULL` for this parameter if you do not want to retrieve the list.

**[typeList](#)**

The list of the container's type.

- 0: Retrieve all container types (`ARCON_ALL`).
- 1: Retrieve all guide containers (`ARCON_GUIDE`).
- 2: Retrieve all application containers (`ARCON_APP`).

- 3: Retrieve all packing list containers (ARCON\_PACK).
- 4: Retrieve all filter guide containers (ARCON\_FILTER\_GUIDE).

### referenceList

The list of pointers to the objects (for example, forms or filters) referenced by this container. Specify `NULL` for this parameter if you do not want to retrieve the list.

### helpTextList

A list of the help texts associated with the containers. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether instances of these objects exist).

### ownerList

The list of owning users for the containers. Specify `NULL` for this parameter if you do not want to retrieve the list.

### timestampList

The list of time stamps identifying the last change to the containers. Specify `NULL` for this parameter if you do not want to retrieve the list.

### lastChangedList

The list of users who made the last change to the containers. Specify `NULL` for this parameter if you do not want to retrieve the list.

### changeDiaryList

The list of change diary entries associated with the containers. The server adds the user making the change and a time stamp when it saves the change diary. Use `ARDecodeDiary` to parse the change diary into user, time stamp, and text components. Specify `NULL` for this parameter if you do not want to retrieve the change diary list (which is useful if you are calling this function to verify whether an instance of this object exists).

### objPropListList

If the `objPropListList` parameter is set to `NULL`, no object properties will be returned. See “Server object property tags” on page 79 for object property tag names and data types.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARGetContainer, ARGetListContainer.

# ARGetMultipleCurrencyRatioSets

**Description** Retrieves a list of formatted currency ratio sets valid for the times specified in the `rati oTi mestamps` argument. You can use `ARGetCurrencyRatio` to extract a specific currency ratio from a ratio set that this call (`ARGetMul ti pl eCurrencyRatioSets`) returns.

**Privileges** All users.

### Synopsis

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

int ARGetMultipleCurrencyRatioSets(
    ARControlStruct *control,
    ARTimestampList *ratioTimestamps,
    ARTextStringList *currencyRatioSets,
    ARStatusList *status)
```

**Input arguments**

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

### ratioTimestamps

A list of time stamps that represent times to retrieve sets of ratios. Specify the `AR_CURRENT_CURRENCY_RATIOS` constant as a time stamp to retrieve a set of current ratios for each conversion combination. If no ratio exists for a currency code combination exactly at the specified point in time, the closest earlier ratio is returned.

**Return values****[currencyRatioSets](#)**

A pointer to a structure that contains a list of formatted character strings, each of which represents the set of currency ratios for the corresponding point in time, as the `ratioTimestamps` argument specifies. If there are no available currency ratios that qualify, such as when there are no earlier ratios, the corresponding character string will be an empty string.

You must free the memory allocated within the structure described previously. To determine the value for a specific ratio from the formatted character string, use the function `ARGetCurrencyRatio`.

**[status](#)**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** `ARGetCurrencyRatio`. See `FreeAR` for: `FreeARTextStringList`.

## ARGetMultipleEntryPoints

**Description**

Retrieves the entry points of multiple applications. It returns the entry points that are accessible by the user, taking into account user permissions, licenses, and application states.

**Privileges**

The system returns information based on the access privileges of the user you specify for the `control` parameter.

**Synopsis**

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

int ARGetMultipleEntryPoints(
    ARControlStruct *control,
    ARTimeStamp changedSince,
    ARNameList *appNameList,
    ARReferenceTypeList *refTypeList,
    *displayTag,
    viewType,
    entryPointNameList,
    entryPointTypeList,
    entryPointLabelList,
```

ARNameLi st	*ownerAppNameLi st,
ARTextStri ngLi st	*ownerAppDLabel Li st
ARPermi ssi onLi stLi st	*groupLi stLi st,
ARContai nerOwnerObj Li stLi st	*ownerObj Li stLi st,
ARTextStri ngLi st	*descripti onLi st,
ARReferenceLi stLi st	*referencesLi st,
ARTextStri ngLi st	*hel pTextLi st,
ARTi mestampLi st	*ti mestampLi st,
ARPropLi stLi st	*obj PropLi stLi st,
ARStatusLi st	*status)

## Input arguments

### **control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

### **changedSince**

A time stamp that limits the objects retrieved to those modified after the specified time. Specify 0 for this parameter to retrieve objects with any modification time stamp.

### **appNameList**

A list of applications objects that contains the entry points.

### **refTypeList**

A list of reference types to be retrieved.

### **displayTag**

The name of the form view (VUI) to use for the entry point label. If the specified view does not exist or you specify `NULL` for this parameter, the system uses the default view.

### **vuiType**

The type of VUI to retrieve. Specify `NULL` for this parameter if you do not want to retrieve the VUI name.

- 0: No VUI type (`AR_VUI_TYPE_NONE`).
- 1: Windows type (`AR_VUI_TYPE_WINDOWS`)—fields in BMC Remedy User.
- 2: Web relative type (`AR_VUI_TYPE_WEB`)—fields in view can be adjusted.
- 3: Web absolute type (`AR_VUI_TYPE_WEB_ABS_POS`)—fields in view are fixed.

**Return values****entryPointNameList**

A list of entry point names. If the entry point type is an active link guide, this is the name of the container. If the entry point type is a form, this is the name of the form.

**entryPointTypeList**

A list of entry point types.

- 1: The entry point type for an active link guide (AR\_ENTRYPOI\_NT\_TYPE\_AL\_GUI\_DE).
- 2: The entry point type for a form in default search mode (AR\_ENTRYPOI\_NT\_TYPE\_DEFAULT\_SEARCH).
- 3: The entry point type for a form in default new entry mode (AR\_ENTRYPOI\_NT\_TYPE\_DEFAULT\_NEW).

**entryPointDLabelList**

Display label for entry point, which is optionally localized.

**ownerAppNameList**

The application owner of the entry point.

**ownerAppDLabelList**

The display label for the application owner, which is optionally localized. This is the localized label for the application container object.

**groupListList**

A list of zero or more groups who can access this form or active link guide. Access to this information is limited to users with AR System administrator privileges only. Specify `NULL` for this parameter if you do not want to retrieve the group list.

**ownerObjListList**

For active link guide entry points, a list of the schemas that own this container. For form entry points, this is the name of the form that the default entry point came from. Specify `NULL` for this parameter if you do not want to retrieve this value. If this parameter returns `NULL`, the container exists globally.

## descriptionList

For active link guide entry points only, this is the list of descriptions for this entry point. Specify `NULL` for this parameter if you do not want to retrieve the list.

## referencesList

A list of pointers to the objects (for example, forms or filters) referenced by this container. Specify `NULL` for this parameter if you do not want to retrieve this value.

## helpTextList

A list of the help texts associated with the entry point. Specify `NULL` for this parameter if you do not want to retrieve the help texts (which is useful if you are calling this function to verify whether instances of these objects exist).

## timestampList

A list of time stamps identifying the last change to the entry point object. Specify `NULL` for this argument if you do not want to retrieve this value.

## objPropListList

The list of object properties. If the `objPropListList` parameter is set to `NULL`, no object properties will be returned. See “Server object property tags” on page 79 for object property tag names and data types.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARGetMultipleContainers.

# ARGetMultipleEntries

**Description** Retrieves the group of form entries specified by the `entryIdList` parameter. You can retrieve data for specific fields or all (accessible) fields. This function only returns the name, size, and compressed size of attachment fields. Use `ARGetEntryBLOB` to retrieve the contents of the attachment. This function performs the same action as `ARGetEntry` but is easier to use and more efficient than retrieving multiple entries one by one.

---

**Note:** A maximum of 100 entries can be returned by this function. If you need to return more than 100 entries, call this function multiple times.

---

**Privileges** The system returns data based on the access privileges of the user you specify for the `control` parameter. User permissions are verified for each specified entry and field. The system returns values for accessible fields and warning messages for fields the user cannot access.

## Synopsis

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

int ARGetMultipleEntries(
    ARControlStruct *control,
    ARNameType schema,
    *entryIdList,
    *idList,
    *existList,
    *fieldList,
    *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

### schema

The name of the form containing the entries to retrieve.

### entryIdList

The list of zero or more entries to retrieve.

## **IdList**

A list of zero or more internal IDs specifying the fields to retrieve. Specify `NULL` for this parameter (or zero fields) to retrieve all (accessible) fields. To improve performance, specify only the fields you need if you do not require the data for all fields. If an attachment field is specified in the list, only its name, size, and compressed size will be returned. Use `ARGetEntryBLOB` to retrieve the contents of the attachment.

### **Return values**

#### **existList**

A list of flags and corresponding Boolean values indicating whether the entries exist. Values: `True` = entry exists; `False` = entry does not exist.

#### **fieldList**

A list of zero or more fields and associated values that identify the data for the specified entries. The fields are returned in the order specified by `fieldList`. If a specified entry does not exist, the system returns zero (or `NULL`) for that entry. If the user does not have permission for a specified field or the field does not exist, the system returns `NULL` for the associated value within each entry value list.

#### **status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

### **See also**

`ARCreatEntry`, `ARDecodeDiary`, `ARDecodeStatusHistory`, `ARDeleteEntry`, `ARGetEntry`, `ARGetListEntry`, `ARGetMultipleActiveLinks`, `ARGetMultipleExtFieldCandidates`, `ARGetMultipleFields`, `ARGetMultipleLocalizedValues`, `ARGetMultipleSchemas`, `ARSetEntry`. See `FreeAR` for: `FreeARBooleanList`, `FreeAREntryIdList`, `FreeARInternalIdList`, `FreeARStatusList`.

# ARGetMultipleEscalations

**Description** Retrieves information about a group of escalations on the specified server with the names specified by the `nameList` parameter. This function performs the same action as `ARGetEscalation` but is easier to use and more efficient than retrieving multiple entries one by one.

Information is returned in lists for each item, with one item in the list for each escalation returned. For example, if the second item in the list for `existList` is TRUE, the time stamp identifying the last change to the second escalation is returned in the second item in the list for `timeStampList`.

**Privileges** All users.

## Synopsis

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

```
int ARGetMultipleEscalations(
    ARControlStruct *control,
    ARTimestamp changedSince,
    *nameList,
    *existList,
    *escalNameList,
    *escalTimeList,
    *workflowConnectList,
    *enableList,
    *queryList,
    *actionList,
    *elementList,
    *helpTextList,
    *timestampList,
    *ownerList,
    *lastChangedList,
    *changeDirectoryList,
    *objPropList,
    *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

## changedSince

A time stamp that limits the escalations retrieved to those modified after the specified time. Specify 0 for this parameter to retrieve escalations with any modification time stamp.

## nameList

The names of the escalations to retrieve. The nameList can be passed as a `NULL` or as an empty list, as in:

```
ARNameList emptyNameList = {0, 0};  
...  
ARGetMultipleEscalations(control, changedSince, &emptyNameList,  
...)
```

In this case information is returned for every escalation that passes the `changedSince` condition.

## Return values

### existList

A list of flags and corresponding Boolean values indicating whether the escalations exist. Values: `True` = escalation exists; `False` = escalation does not exist.

### escalNameList

A list of zero or more escalations that match the criteria defined in the `changedSince` and `nameList` parameters. The system returns a list with zero items if no escalations match the specified criteria.

### escalITMList

A list of `AREscalationTmStruct`. `ARGetEscalation` returns a single `AREscalationTmStruct`, which describes when the escalation executes, such as once an hour or on specified days.

### workflowConnectList

A list of the items that `ARGetEscalation` returns one at a time, which are the schemas to which the escalation is attached.

### enableList

A list of flags identifying whether the escalation is disabled (0) or enabled (1). Specify `NULL` for this parameter if you do not want to retrieve these flags.

## queryList

A list of qualifications that determines whether the escalations are executed. The system returns zero (AR\_COND\_OP\_NONE) if the escalations have no qualifications. Specify `NULL` for this parameter if you do not want to retrieve the queries.

## actionListList

A list of the sets of actions performed if the condition defined by the `query` parameter is satisfied. This list can contain from 1 to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve the action lists.

## elseListList

A list of the sets of actions performed if the conditions defined by the `query` parameter is not satisfied. This list can contain from 0 to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve the else list.

## helpTextList

A list of help text items associated with the escalations. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of these objects exists).

## timeStampList

A list of time stamps identifying the last change to the escalations. Specify `NULL` for this parameter if you do not want to retrieve this value.

## ownerList

A list of owning users for the escalations. Specify `NULL` for this parameter if you do not want to retrieve this value.

## lastChangedList

A list of users who made the last change to the escalations. Specify `NULL` for this parameter if you do not want to retrieve this value.

## changeDiaryList

A list of change diaries associated with the escalations. Use ARDecodeDi ary to parse the change diary into user, time stamp, and text components. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to retrieve the change diary list (which is useful if you are calling this function to verify whether an instance of this object exists).

## objPropListList

A list of object properties. If the `obj PropLi stLi st` parameter is set to `NULL`, no object properties will be returned. See “Server object property tags” on page 79 for object property tag names and data types.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateEscalation, ARDecodeDi ary, ARDeleteEscalation, ARGetEscalation, ARGetListEscalation, ARSetEscalation. See FreeAR for: FreeARFilterActionList, FreeARPropList, FreeARQualifierStruct, FreeARStatusList.

# ARGetMultipleExtFieldCandidates

**Description** Retrieves a list of all available external data source fields (field candidates). Users choose the data from one of these field candidates to populate the vendor form.

**Privileges** The system returns information based on the access privileges of the user you specify for the `control` parameter. All lists, therefore, are limited to fields the user can access.

**Synopsis**

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

int ARGetMultipleExtFieldCandidates(
    ARControlStruct *control,
    ARCompoundSchema *schema,
    ARFieldMappingList *fieldMapping,
```

	ARFi el dLi mi tLi st ARUnsi gnedl ntLi st ARStatusLi st	*I i mi t, *dataType, *status)
<b>Input arguments</b>	<b>control</b>	The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.
	<b>schema</b>	The definition of the candidate form or table that contains the candidate field to retrieve.
<b>Return values</b>	<b>fieldMapping</b>	A list of candidate field definitions.
	<b>limit</b>	A list of field limits for the candidate fields.
	<b>datatype</b>	A list of field datatypes for the candidate fields.
	<b>status</b>	A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the <i>Error Messages</i> guide.
<b>See also</b>	ARGetListExtSchemaCandidates, ARGetMultipleActiveLinks, ARGetMultipleEntries, ARGetMultipleFields, ARGetMultipleLocalizedValues, ARGetMultipleSchemas. See FreeAR for: FreeARCompoundSchema, FreeARFieldMappingList, FreeARFieldLimitList, FreeARUnsignedIntList, FreeARStatusList.	

## ARGetMultipleFields

<b>Description</b>	Retrieves one or more field definitions for a specified form. This function performs the same action as ARGetFi el d but is easier to use and more efficient than retrieving multiple entries one by one.
--------------------	---

Information is returned in lists for each item, with one item in the list for each field returned. For example, if the second item in the list for `extList` is TRUE, the name of the second field is returned in the second item in the list for `fieldNames`.

**Privileges** This operation can be performed by users with any level of access permission for the fields in the specified form. Access to permissions information is limited to users with AR System administrator privileges only.

### Synopsis

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

int ARGetMultiFields(
    ARControlStruct *control,
    ARNameType schema,
    *fieldList,
    *extList,
    *fieldList2,
    *fieldName,
    *fieldMap,
    *dataType,
    *option,
    *createMode,
    *fieldOption
    *defaultValue,
    *permissions,
    *limit,
    *instanceList,
    *helpText,
    *timestamp,
    *owner,
    *lastChanged,
    *changeDairy,
    *status)
```

### Input arguments

#### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

## schema

The name of the form containing the field to retrieve.

## fieldId

The internal ID list of fields to retrieve. Specify `NULL` or `fieldIdList.numItems=0` for this parameter to retrieve data for *all* the fields in the form.

## Return values

### existList

A list of flags and corresponding Boolean values indicating whether the fields exist. Values: `True` = field exists; `False` = field does not exist.

### fieldId2

The list of internal IDs of the fields retrieved. Specify `NULL` for this parameter if you do not want to retrieve the field IDs.

### fieldName

The list of names of the fields. Specify `NULL` for this parameter if you do not want to retrieve the field names.

### fieldMap

The underlying form in which to create the field (applicable for join forms only). If you are creating a field in a base form, specify a field type of 1 (`AR_FIELD_REGULAR`). Otherwise, specify a field type of 2 (`AR_FIELD_JOIN`), and identify the member form (primary or secondary) and field ID for the new field. If the member form is also a join form, create fields in all nested join forms until you can map the field to an underlying base form. Specify `NULL` for this parameter if you do not want to retrieve the field mapping.

### dataType

The list data types of the fields. See “`ARCreateField`” on page 143 for a description of the possible values. Specify `NULL` for this parameter if you do not want to retrieve the data types.

### option

A list of flags indicating whether users must enter a value in the fields. See “`ARCreateField`” on page 143 for a description of the possible values. Specify `NULL` for this parameter if you do not want to retrieve this flag.

## createMode

A list of flags indicating the permission status for the fields when users submit entries. See “ARCreateField” on page 143 for a description of the possible values. Specify `NULL` for this parameter if you do not want to retrieve this value.

## fieldOption

A list of bitmasks that indicates whether the field should be audited or copied when other fields are audited.

- Bit 0: Audit this field. (`AR_FIELD_IS_AUDITED`)
- Bit 1: Copy this field when other fields in the form are audited. (`AR_FIELD_IS_COPIED`)
- Bit 2: Indicates this field is for Log Key 1. (`AR_FIELD_IS_LOG_KEY1`)
- Bit 3: Indicates this field is for Log Key 2. (`AR_FIELD_IS_LOG_KEY2`)
- Bit 2 and 3: Indicates this field is for Log Key 3. (`AR_FIELD_IS_LOG_KEY3`)

## defaultVal

The list of values to apply if a user submits an entry with no field value (applicable for data fields only). The system returns 0 (`AR_DEFAULT_VALUE_NONE`) if the field has no default. Specify `NULL` for this parameter if you do not want to retrieve the default values.

## permissions

A list of lists containing zero or more groups who can access the fields retrieved. Access to this information is limited to users with AR System administrator privileges only. Specify `NULL` for this parameter if you do not want to retrieve the permissions.

## limit

The list of value limits for the fields and other properties specific to the individual fields’ respective types. See the `ARFieldDefinition` definition in the `ar.h` file to find the contained structure that applies to the type of field you are creating. Specify `NULL` for this parameter if you do not want to retrieve the limits and properties.

## dInstanceList

A list of lists containing zero or more display properties associated with the fields. See “[ARCreateField](#)” on page 143 for a description of the possible values. The system returns 0 (AR\_DPROP\_NONE) if the field has no display properties. Specify `NULL` for this parameter if you do not want to retrieve this list.

## helpText

The list of help text associated with the fields. Specify `NULL` for this parameter if you do not want to retrieve the help text.

## timestamp

A list of time stamps identifying the last change to the fields. Specify `NULL` for this parameter if you do not want to retrieve this value.

## owner

The list of owning users for the field. Specify `NULL` for this parameter if you do not want to retrieve the owner.

## lastChanged

The list of users who made the last change to the fields. Specify `NULL` for this parameter if you do not want to retrieve this value.

## changeDiary

The list of change diary entries associated with the fields. The server adds the user making the change and a time stamp when it saves the change diary. Use `ARDecodeDiary` to parse the change diary into user, time stamp, and text components. Specify `NULL` for this parameter if you do not want to retrieve the change diary.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “[Error checking](#)” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateField, ARDecodeDiary, ARDeleteField, ARGetField, ARGetListField, ARGetMultipleActiveLinks, ARGetMultipleEntries, ARGetMultipleExtFieldCandidates, ARGetMultipleLocalizedValues, ARGetMultipleSchemas, ARGetSchema, ARSetField. See FreeAR for: FreeARBooleanList, FreeARAccessNameList, FreeARDisplayInstanceListList, FreeARFieldLimitList, FreeARFieldMappingList, FreeARInternalIdList, FreeARNameList, FreeARPermissionListList, FreeARStatusList, FreeARTextStringList, FreeARTimestampList, FreeARUnsignedIntList, FreeARValueList.

## ARGetMultipleFilters

**Description** Retrieves information about a group of filters on the specified server with the names specified by the `nameList` parameter. This function performs the same action as `ARGetFilter` but is easier to use and more efficient than retrieving multiple entries one by one.

Information is returned in lists for each item, with one item in the list for each filter returned. For example, if the second item in the list for `existList` is TRUE, the name of the second filter is returned in the second item in the list for `filterNameList`.

**Privileges** All users.

### Synopsis

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

int ARGetMultipleFilters(
    ARControlStruct*control,
    ARTimestamp changedSince,
    *nameList,
    *existList,
    *filterNameList,
    *orderList,
    *workflowConnectList,
    *opSetList,
    *enableList,
    *queryList,
    *actionList,
    *elseList,
    *helpTextList,
```

```

ARTimestampList
ARAccessNameList
ARAccessNameList
ARTextStringList
ARPropListList
ARStatusList
*timestampList,
*ownerList,
*lastChangedList,
*changeDirectoryList,
*objPropListList,
*status)

```

## Input arguments

### **control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

### **changedSince**

A time stamp that limits the filters retrieved to those modified after the specified time. Specify 0 for this parameter to retrieve filters with any modification time stamp.

### **nameList**

The names of the filters to retrieve. The `nameList` can be passed as a `NULL` or as an empty list, as in:

```

ARNameList emptyNameList = {0, 0};

ARGetMultipleFilters(control, changedSince, &emptyNameList, ... )

```

In this case information is returned for every filter that passes the `changedSince` condition.

## Return values

### **existList**

A list of flags and corresponding Boolean values indicating whether the filters exist. Values: `True` = escalation exists; `False` = escalation does not exist.

### **filterNameList**

A list of zero or more filters that match the criteria defined in the `changedSince` and `nameList` parameters. The system returns a list with zero items if no filters match the specified criteria.

## orderList

A list of values between 0 and 1000 (inclusive) that determines the filter execution order. When multiple filters are associated with a form, the value associated with each filter determines the order in which they are processed (from lowest to highest). Specify `NULL` for this parameter if you do not want to retrieve this value.

## workflowConnectList

A list of the items that `ARGetFilter` returns one at a time, which are the schemas to which the filter is attached.

## opSetList

Each member of `opSetList` is an integer bit mask indicating on which conditions the filter executes: form get, create, delete, update, merge. For more information, see “`ARCreateFilter`” on page 160.

## enableList

A list of flags identifying whether the filter is disabled (0) or enabled (1). Specify `NULL` for this parameter if you do not want to retrieve these flags.

## queryList

A list of qualifications that determines whether the filters are executed. The system returns zero (`AR_COND_OP_NONE`) if the filters have no qualifications. Specify `NULL` for this parameter if you do not want to retrieve the queries.

## actionListList

A list of the sets of actions performed if the condition defined by the `query` parameter is satisfied. This list can contain from 1 to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve the action lists.

## elseListList

A list of the sets of actions performed if the conditions defined by the `query` parameter is not satisfied. This list can contain from 0 to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve the else list.

## helpTextList

A list of help text items associated with the filters. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of these objects exists).

## timeStampList

A list of time stamps identifying the last change to the filters. Specify `NULL` for this parameter if you do not want to retrieve this value.

## ownerList

A list of owning users for the filters. Specify `NULL` for this parameter if you do not want to retrieve this value.

## lastChangedList

A list of users who made the last change to the filters. Specify `NULL` for this parameter if you do not want to retrieve this value.

## changeDiaryList

A list of change diaries associated with the filters. Use `ARDecodeDiary` to parse the change diary into user, time stamp, and text components. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to retrieve the change diary list (which is useful if you are calling this function to verify whether an instance of this object exists).

## objPropListList

A list of object properties. If the `objPropListList` parameter is set to `NULL`, no object properties will be returned. See “Server object property tags” on page 79 for object property tag names and data types.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** `ARCCreateFilter`, `ARDecodeDiary`, `ARDeleteFilter`, `ARGetFilter`, `ARGetListFilter`, `ARSetFilter`. See `FreeAR` for: `FreeARFilterActionList`, `FreeARPropList`, `FreeARQualifierStruct`, `FreeARStatusList`.

# ARGetMultipleLocalizedValues

**Description** Retrieves multiple localized text strings from the BMC Remedy Message Catalog. The messages that the server retrieves depend on the user locale in the control structure. This function performs the same action as ARGetLocalizedValues but is easier to use and more efficient than retrieving multiple values one by one.

**Privileges** All users.

## Synopsis

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

int ARGetMultipleLocalizedValues(
    ARControlStruct *control,
    ARLocalizedRequestList *localizedRequestList,
    ARValueList *localizedValueList,
    ARTimestampList *timestampList,
    ARStatusList *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

### localizedRequestList

A list of identifications for the localized value to be retrieved. The list contains the type of value, value name, and, for some types of values, unique identification numbers.

## Return values

### localizedValueList

A list of the localized values to be retrieved. The values can be a character value or an attachment. Specify `NULL` for this argument if you do not want to retrieve a value.

### timestampList

A list of time stamps identifying the last change to the values. Specify `NULL` for this argument if you do not want to retrieve this value.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARGetLocalizedValue, ARGetMultipleActiveLinks, ARGetMultipleEntries, ARGetMultipleExtFieldCandidates, ARGetMultipleFields, ARGetMultipleSchemas. See FreeAR for: FreeARLocalizedRequestList, FreeARValueList, FreeARTimestampList, FreeARStatusList.

# ARGetMultipleSchemas

**Description** Retrieves information about multiple forms with the indicated names on the specified server. This information does not include the form field definitions (see “ARGetField” on page 233 or “ARGetMultipleFields” on page 299). This function performs the same action as `ARGetSchema` but is easier to use and more efficient than retrieving multiple forms one by one.

Information is returned in lists for each item, with one item in the list for each form returned. For example, if the second item in the list for `extLi st` is TRUE, the name of the second form is returned in the second item in the list for `schemaNameLi st`.

**Privileges** This operation can be performed by users with access permission for the form. Access to `groupLi st` and `adminGrpLi st` information is limited to users with AR System administrator privileges only.

## Synopsis

```
#i ncl ude "ar.h"
#i ncl ude "arerrno.h"
#i ncl ude "arextern.h"
#i ncl ude "arstruct.h"

int ARGetMultipleSchemas(
    ARControlStruct*control,
    ARTimestamp changedSince,
    *SchemaTypeList*schemaTypeList,
    *NameList*nameList,
    *FieldList*fieldList,
    *ExtList*extList,
    *SchemaNameList*schemaNameList,
    *SchemaList*schemaList,
    *SchemaNameList*schemaNameListList);
```

```

ARPermi ssi onLi stLi st
ARI nternal l dLi stLi st
AREntryLi stFi el dLi stLi st
ARSortLi stLi st
ARI ndexLi stLi st
ARArchivel nfoLi st
ARAudi tI nfoLi st
ARNameLi st
ARTextStringLi st
ARTimestampLi st
RAccessNameLi st
RAccessNameLi st
ARTextStringLi st
ARPropLi stLi st
ARStatusLi st
*groupLi stLi st,
*admi ngrpLi stLi st,
*getLi stFi el dsLi st,
*sortLi stLi st,
*indexLi stLi st,
archivel nfoLi st.
audi tI nfoLi st.
*defaul tVui Li st,
*hel pTextLi st,
*timestampLi st,
*ownerLi st,
*lastChangedLi st,
*changeD i aryLi st,
*obj PropLi stLi st,
*status)

```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

### changedSince

A time stamp that limits the forms retrieved to those modified after the specified time. Specify 0 for this parameter to retrieve forms with any modification time stamp.

### schemaTypeList

A list of values indicating the form types to retrieve (some types are mutually exclusive).

- Bi t 0: Retrieve a list of all form types (AR\_LIST\_SCHEMA\_ALL).
- Bi t 1: Retrieve a list of all base forms (AR\_LIST\_SCHEMA\_REGULAR).
- Bi t 2: Retrieve a list of all join forms (AR\_LIST\_SCHEMA\_JOIN).
- Bi t 3: Retrieve a list of all view forms (AR\_LIST\_SCHEMA\_VIEW).
- Bi t 6: Retrieve a list of all display-only forms (AR\_LIST\_SCHEMA\_DISPLAY).
- Bi t 7: Retrieve a list of all forms with data fields (AR\_LIST\_SCHEMA\_ALL\_WITH\_DATA).
- Bi t 8: Retrieve a list of all vendor schemas (AR\_SCHEMA\_VENDOR).

---

To retrieve both visible and hidden forms, add 1024 (AR\_HI DDEN\_I NCREMENT) to the form type. For example, specify AR\_LI ST\_SCHEMA\_ALL | AR\_HI DDEN\_I NCREMENT for this parameter. This call does not support the AR\_LI ST\_SCHEMA\_UPLINK and AR\_LI ST\_SCHEMA\_DOWNLINK form types.

### nameList

The names of the schemas to retrieve.

### fieldIdList

List of form fields that exist in the retrieved schema. The system only returns the forms that contain all the fields in this list. Specify NULL for this parameter (or zero fields) to retrieve all (accessible) fields.

---

**Note:** Archive forms are not returned if you specify a list of form fields. You must specify NULL to return archive forms.

---

### archiveInfoList

The list of archive information associated with the form. Specify NULL for this parameter to not retrieve archive information.

### auditInfoList

The list of audit information associated with the form. Specify NULL for this parameter to not retrieve audit information.

## Return values

### existList

A list of flags and corresponding Boolean values indicating whether the forms exist. Values: True = forms exists; False = form does not exist.

### schemaNameList

An array of retrieved schema names.

### schemaList

A list of form types (base form or join form). See *ARCreateSchema* on page 158 for a description of the possible values. Specify NULL for this parameter if you do not want to retrieve this value.

### schemaInheritanceListList

This is reserved for future use. Specify NULL.

## groupListList

A list of zero or more groups who can access this form. Access to this information is limited to users with AR System administrator privileges only. Specify `NULL` for this parameter if you do not want to retrieve the group list.

## admingrpListList

A list of zero or more groups who can administer this form (and the associated filters, escalations, and active links). Users must belong to both a specified group *and* the Subadministrator group to obtain these privileges. Specify `NULL` for this parameter if you do not want to retrieve the administrator group list.

## getListFieldsList

A list of zero or more fields that identifies the default query list data for retrieving form entries. Specify `NULL` for this parameter if you do not want to retrieve the default query list fields.

## sortListList

A list of zero or more fields that identifies the default sort order for retrieving form entries. Specify `NULL` for this parameter if you do not want to retrieve this value.

## indexListList

The set of zero or more indexes for the form. Specify `NULL` for this parameter if you do not want to retrieve the index list.

## defaultVuiList

The default VUI label. Specify `NULL` for this parameter if you do not want to retrieve the index list.

## helpTextList

A list of the help texts associated with the forms. Specify `NULL` for this parameter if you do not want to retrieve the help texts (which is useful if you are calling this function to verify whether instances of these objects exist).

## timestampList

A list of time stamps that identify the last changes to the forms. Specify `NULL` for this parameter if you do not want to retrieve this value.

## ownerList

A list of form owners. Specify `NULL` for this parameter if you do not want to retrieve the owners.

## lastChangedList

A list of users who made the last changes to the forms. Specify `NULL` for this parameter if you do not want to retrieve this value.

## changeDiaryList

A list of the change diaries associated with the forms. Use `ARDecodeDiary` to parse the change diary into user, time stamp, and text components. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to retrieve the change diary (which is useful if you are calling this function to verify whether an instance of this object exists).

## objPropListList

If the `objPropListList` parameter is set to `NULL`, an object properties list with zero properties will be associated with the object, and when an `ARGetSchema` action is performed, zero properties are returned. See “Server object property tags” on page 79 for object property tag names and data types.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

### See also

`ARCreatSchema`, `ARDeleteSchema`, `ARGetSchema`, `ARGetListSchema`, `ARGetMultipleActiveLinks`, `ARGetMultipleEntries`, `ARGetMultipleExtFieldCandidates`, `ARGetMultipleFields`, `ARGetMultipleLocalizedValues`, `ARSetSchema`. See `FreeAR` for: `FreeARAccessNameList`, `FreeARBooleanList`, `FreeARCompoundSchemaList`, `FreeAREntryListFieldListList`, `FreeARIndexListList`, `FreeARInternalIdList`, `FreeARInternalIdListList`, `FreeARNameList`, `FreeARPermissionListList`, `FreeARPropListList`, `FreeARSortListList`, `FreeARStatusList`, `FreeARTextStringList`, `FreeARTimestampList`, `FreeARUnsignedIntList`.

# ARGetMultipleVUIs

**Description** Retrieves information about a group of form views (VUIs) on the specified server with the names specified by the `nameList` parameter. This function performs the same action as `ARGetVUI` but is easier to use and more efficient than retrieving multiple entries one by one.

Information is returned in lists for each item, with one item in the list for each VUI returned. For example, if the second item in the list for `existList` is TRUE, the name of the second VUI is returned in the second item in the list for `nameList`.

**Privileges** All users.

## Synopsis

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

```
int ARGetMultipleVUIs(
    ARControlStruct *control,
    ARNameType schema,
    *wantList,
    changedSince,
    *existList,
    *gotList,
    *nameList,
    *localList,
    *vuiTypeList,
    *dPropListList,
    *helpTextList,
    *timestampList,
    *ownerList,
    *lastChangedList,
    *changeDictionaryList,
    *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

## schema

The names of the forms containing the VUIs to retrieve.

## wantList

The IDs of the VUIs to retrieve. The wantList can be passed as a NULL or as an empty list, as in:

```
ARWantList emptyWantList = {0, 0};  
...  
ARGetMultipleVUIs(control, changedSince, &emptyWantList, ...)
```

In this case information is returned for every VUI that passes the changedSince condition.

## changedSince

A time stamp that limits the VUIs retrieved to those modified after the specified time. Specify 0 for this parameter to retrieve VUIs with any modification time stamp.

## Return values

### existList

A list of flags and corresponding Boolean values indicating whether the VUIs exist and meet the qualifying criteria. Values: True = VUI exists; False = VUI does not exist.

### nameList

A list of VUIs retrieved.

### localeList

A list of locales retrieved.

### vuiTypeList

A list of VUI types retrieved.

### dPropListList

A list of display properties of VUIs. Specify NULL for this parameter if you do not want to retrieve this value.

### helpTextList

A list of help text items associated with the VUIs. Specify NULL for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of these objects exists).

### timeStampList

A list of time stamps identifying the last change to the VUIs. Specify `NULL` for this parameter if you do not want to retrieve this value.

### ownerList

A list of owning users for the VUIs. Specify `NULL` for this parameter if you do not want to retrieve this value.

### lastChangedList

A list of users who made the last change to the VUIs. Specify `NULL` for this parameter if you do not want to retrieve this value.

### changeDiaryList

A list of change diaries associated with the VUIs. Use `ARDecodeDiary` to parse the change diary into user, time stamp, and text components. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to retrieve the change diary list (which is useful if you are calling this function to verify whether an instance of this object exists).

### objPropListList

A list of object properties. If the `objPropListList` parameter is set to `NULL`, no object properties will be returned. See “Server object property tags” on page 79 for object property tag names and data types.

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** `ARCreatVUI`, `ARDecodeDiary`, `ARDeleteVUI`, `ARGetListVUI`, `ARGetVUI`, `ARSetVUI`. See `FreeAR` for: `FreeARPropList`, `FreeARStatusList`.

## ARGetSchema

**Description** Retrieves information about the form with the indicated name on the specified server. This information does not include the form’s field definitions (see “`ARGetField`” on page 233).

**Privileges** This operation can be performed by users with access permission for the form. Access to groupList and administrator information is limited to users with AR System administrator privileges only.

### Synopsis

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

int ARGetSchema(
    ARControlStruct *control,
    ARNameType name,
    ARCompoundSchema *schema,
    ARSchemaList *schemas,
    *groupList,
    *adminGroupList,
    *getListFields,
    *sortList,
    *indexList,
    *archivedInfo,
    *audioInfo,
    defaultVui,
    **helpText,
    *timestamp,
    owner,
    lastChanged,
    **changeDir,
    *objPropList,
    *status)
```

### Input arguments

#### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

#### name

The name of the form to retrieve.

### Return values

#### schema

The form type (base form or join form). See “ARCreateSchema” on page 164 for a description of the possible values. Specify NULL for this parameter if you do not want to retrieve the form type.

## schematicInheritance

This is reserved for future use. Specify `NULL`.

## groupList

A list of zero or more groups who can access the form. Access to this information is limited to users with AR System administrator privileges only. Specify `NULL` for this parameter if you do not want to retrieve the group list.

## admingrpList

A list of zero or more groups who can administer this form (and the associated filters, escalations, and active links). Users must belong to both a specified group *and* the Subadministrator group to obtain these privileges. Specify `NULL` for this parameter if you do not want to retrieve the administrator group list.

## getListFields

A list of zero or more fields that identifies the default query list data for retrieving form entries. Specify `NULL` for this parameter if you do not want to retrieve this value.

## sortList

A list of zero or more fields that identifies the default sort order for retrieving form entries. Specify `NULL` for this parameter if you do not want to retrieve the default sort fields.

## indexList

The set of zero or more indexes for the form. Specify `NULL` for this parameter if you do not want to retrieve the index list.

## archiveInfo

If a form is to be archived, this is the archive information for the form. Specify `NULL` for this parameter if you do not want to retrieve the archive information. These are the values for archive type:

- 0: Deletes the archive settings for the selected form. The selected form will not be archived (`AR_ARCHIVE_NONE`).
- 1: Entries on the selected form are copied to the archive form (`AR_ARCHIVE_FORM`).
- 2: Entries on the selected form are deleted from the source form (`AR_ARCHIVE_DELETE`).

- 4: Entries on the selected form are copied to an XML format file (AR\_ARCHI VE\_FI LE\_XML).
- 8: Entries on the selected form are copied to an ARX format file (AR\_ARCHI VE\_FI LE\_ARX).
- 32: Attachment fields are not copied to the archive form. (AR\_ARCHI VE\_NO\_ATTACHMENTS).
- 64: Diary fields are not copied to the archive form. (AR\_ARCHI VE\_NO\_DIARY).

In addition to the archive type, archi vel nfo also stores the form name (if archive type is AR\_ARCHI VE\_FORM), time to trigger the archive, and the archive qualification criteria. For an archive form, the archive type will be AR\_ARCHI VE\_NONE and the name of the source form for which this is the archive form is returned.

### auditInfo

If a form is to be audited, this is the audit information for the form. Specify NULL for this parameter if you do not want to retrieve the audit information. These are the values for audit type:

- 0: The selected form is not to be audited. (AR\_AUDI T\_NONE).
- 1: Audit fields and copy fields on the selected form are copied to the audit shadow form (AR\_AUDI T\_COPY).
- 2: Audit fields and copy fields on the selected form are copied to the audit log form (AR\_AUDI T\_LOG).

In addition to the audit type, audi tI nfo also stores the form name (if audit type is AR\_AUDI T\_COPY or AR\_AUDI T\_LOG) and the audit qualification criteria.

### defaultVui

The label for the default view.

### helpText

The help text associated with the form. Specify NULL for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

### timestamp

A time stamp identifying the last change to the form (includes changing or adding fields or character menus). Specify NULL for this parameter if you do not want to retrieve this value.

**owner**

The owning user for the form. Specify `NULL` for this parameter if you do not want to retrieve the owner.

**lastChanged**

The user who made the last change to the form. Specify `NULL` for this parameter if you do not want to retrieve this value.

**changeDiary**

The change diary associated with the form. Use `ARDecodeDiary` to parse the change diary into user, time stamp, and text components. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to retrieve the change diary (which is useful if you are calling this function to verify whether an instance of this object exists).

**objPropList**

If the `objPropList` parameter is set to `NULL`, no object properties will be returned. See “Server object property tags” on page 79 for object property tag names and data types.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** `ARCreatSchema`, `ARDecodeDiary`, `ARDeleteSchema`, `ARGetField`, `ARGetListField`, `ARGetListAlertUser`, `ARGetMultipleSchemas`, `ARSetSchema`. See `FreeAR` for: `FreeARCompoundSchema`, `FreeAREntryListFieldList`, `FreeARIndexList`, `FreeARIInternalIdList`, `FreeARPermissionList`, `FreeARPropList`, `FreeARSortList`, `FreeARStatusList`.

## ARGetServerCharSet

<b>Description</b>	Retrieves a string that represents the name of the character set the API library uses to communicate with the server. If this differs from the client charset, the API will convert the data to the right character set.
<b>Privileges</b>	All users.

**Synopsis**

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

int ARGetServerCharSet(
    ARControlStruct *control,
    char *charSet,
    ARStatusList *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**Return values****charSet**

A string that names the character set the API library uses to communicate with the server. `charSet` is a buffer whose size is at least `AR_MAX_LANG_SIZE+1`.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARGetClientCharSet

# ARGetServerInfo

Description	Retrieves the requested configuration information for the specified server.
Privileges	All users.
Synopsis	<pre>#include "ar.h" #include "arerrno.h" #include "arextern.h"  int ARGetServerInfo(     ARControlStruct          *control,     ARServerInfoRequestList *requestList,     ARServerInfoList        *serverInfo,     ARStatusList             *status)</pre>
Input arguments	<p><b>control</b> The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The <code>user</code> and <code>server</code> fields are required.</p> <p><b>requestList</b> A list of one or more server options to return (see the Server options section that follows).</p>
Return values	<p><b>serverInfo</b> The information retrieved from the server (see the Server options section that follows). The system returns <code>NULL</code> (<code>AR_DATA_TYPE_NULL</code>) for server options not retrieved due to error.</p> <p><b>status</b> A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the <i>Error Messages</i> guide.</p>
Server options	The server options represent server information about a specific server. The following server options can be retrieved using the <code>ARGetServerInfo</code> . If the option is not marked ( <i>get only</i> ), then the server option can be set with the <code>ARSetServerInfo</code> call.

**AR\_SERVER\_I\_NFO\_ACTLI\_NK\_DI\_R:**

The name of a directory where all external processes to be run by active links must reside. If the value is `NULL`, active links can run processes located anywhere.

**AR\_SERVER\_I\_NFO\_ACTLI\_NK\_SHELL:**

The name of a shell to use when running external processes from an active link. If the value is `NULL`, processes are run from `/bin/sh` (UNIX) or without a shell (Windows).

**AR\_SERVER\_I\_NFO\_ADMIN\_ONLY:**

A flag indicating whether the server is in Administrator Only mode (T) or not (F). When not in Administrator Only mode, subadministrators can also perform administrator duties. The default value is F (Not in Administrator Only mode).

**AR\_SERVER\_I\_NFO\_ADMIN\_OP\_PROGRESS (get only):**

A flag indicating whether to report the progress of administrative operations, such as importing. A setting of 0 (the default) means that no progress will be provided, and 1 means that progress will be provided. You can view (but not set) this call. Its value contains progress indicator string if an administrative operation (such as importing) provides one *and if AR\_SERVER\_I\_NFO\_ADMIN\_OP\_TRACKING was previously set to 1*. Otherwise the value for `AR_SERVER_I_NFO_ADMIN_OP_PROGRESS` will be empty.

This is a read-only setting. Its value usually remains as an empty string. If `AR_SERVER_I_NFO_ADMIN_OP_TRACKING` was set to 1 previously; then, during an administrative operation, the server might periodically provide a progress indicator string value here. AR System API client programs can poll for this setting to track progress of long running operations.

**AR\_SERVER\_I\_NFO\_ADMIN\_OP\_TRACKING:**

A flag indicating whether to provide any progress indicative information while an administrative operation (such as import) is taking place. A setting of 0 (the default) means that progress information need not be provided, and 1 means that progress information will be provided.

AR\_SERVER\_I NFO\_ALLOW\_BACKQUOTE\_IN\_PROCESS:

A flag indicating whether backquotes are allowed in a run process command string. A setting of 1 means backquotes are allowed. A setting of 0 means they are disallowed.

AR\_SERVER\_I NFO\_ALLOW\_GUESTS:

A flag indicating whether the server accepts guest users. Guest users are users not registered with the AR System. If allowed, guest users have no permissions but are allowed to perform some basic operations. Guest users can submit entries to forms for which permission has been given to the Public group and enter data in fields that have been defined as allowing any user to submit. If not allowed, unregistered users have no access to the system. Values for this option are 1 (TRUE) and 0 (FALSE). The default value is 1 (allow guest users).

AR\_SERVER\_I NFO\_AP\_DEFN\_CHECK:

The length of time in seconds between checks by an application service to verify that definitions it is using from the AR System are correct. The value can be from 0 to 3600 seconds (1 hour). A value of 0 means to check definitions with each command. A larger value means a slight delay in recognizing changes to some definition, while a smaller value means the significant overhead of checking the definitions often. The default value is 300 seconds (5 minutes).

AR\_SERVER\_I NFO\_AP\_LOGFILE:

The name of the file to use if approval tracing is turned on (see AR\_SERVER\_I NFO\_DEBUG\_MODE).

AR\_SERVER\_I NFO\_AP\_RPC\_SOCKET:

The RPC program number that the approval server uses when contacting the AR System. This allows you to define a specific AR System server for private use by the approval server.

AR\_SERVER\_I NFO\_API\_LOGFILE:

The name of the file to use if API tracing is turned on (see AR\_SERVER\_I NFO\_DEBUG\_MODE).

AR\_SERVER\_I\_NFO\_APP\_SERVICE\_PASSWORD:

The Application Service password. During a `get` action, this option returns a `NULL` if there is no password and returns an empty string if there is a password. For security reasons, the actual password string is not returned.

AR\_SERVER\_I\_NFO\_APPL\_PENDING (get only):

The name of the form that contains the pending list of application commands to be processed. This form is specific to the use of Application-Commands.

AR\_SERVER\_I\_NFO\_ARKFORD\_LOGFILE:

The name of the file to use if arkfard tracing is turned on (see AR\_SERVER\_I\_NFO\_DEBUG\_MODE).

AR\_SERVER\_I\_NFO\_CASE\_SENSITIVE (get only):

A value indicating whether the underlying database is case-sensitive. Valid values for this option are 0 (case-sensitive) and 1 (case-insensitive). The default value is 1 (case-insensitive).

AR\_SERVER\_I\_NFO\_CONFIGFILE (get only):

The path name of the AR configuration file.

AR\_SERVER\_I\_NFO\_CLUSTERED\_INDEX:

A value that indicates whether a clustered index is created on the Entry-Id field when a new form is created. Valid values for this option are 1 (create a clustered index) and 0 (create a unique index). The default value is 1 (clustered), but you can override this in the AR System configuration file.

AR\_SERVER\_I\_NFO\_CURR\_PART\_DATE\_STR (get only):

The currency date string.

AR\_SERVER\_I\_NFO\_CURR\_PART\_TYPE\_STR (get only):

The currency type string.

AR\_SERVER\_I\_NFO\_CURR\_PART\_VALUE\_STR (get only):

The currency value string.

AR\_SERVER\_I NFO\_CURRENCY\_I NTERVAL:

The time interval, in minutes, before client programs will refresh currency ratios. A value of zero means never refresh.

AR\_SERVER\_I NFO\_DB2\_DB\_ALI AS (get only):

DB2 database alias name.

AR\_SERVER\_I NFO\_DB2\_SERVER (get only):

Name of the server where the DB2 database resides.

AR\_SERVER\_I NFO\_DB\_CHAR\_SET (get only):

The character set of the AR System database.

AR\_SERVER\_I NFO\_DB\_CONNECTI ON\_RETRI ES:

An integer that indicates the number of times the AR System server tries to connect to the database.

AR\_SERVER\_I NFO\_DB\_NAME (get only):

The name of the underlying SQL database (*not applicable for Oracle databases*). The default value is ARSystem.

AR\_SERVER\_I NFO\_DB\_TYPE (get only):

The underlying database type (character string).

AR\_SERVER\_I NFO\_DB\_USER (get only):

The user name AR System uses to access the underlying database.

AR\_SERVER\_I NFO\_DB\_VERSI ON (get only):

The underlying database version (character string).

AR\_SERVER\_I NFO\_DBCONF (get only):

The complete contents of the db.conf (ardb.cfg) file.

AR\_SERVER\_I NFO\_DBHOME\_DI R (get only):

The home directory for the underlying SQL database.

## AR\_SERVER\_I\_NFO\_DEBUG\_MODE:

A bitmask indicating the server debug modes.

- Bit 0: Enables SQL tracing. The default file for SQL tracing is arsql.log (see AR\_SERVER\_I\_NFO\_SQL\_LOG\_FILE).
- Bit 1: Enables filter tracing. The default file for filter tracing is arfilter.log (see AR\_SERVER\_I\_NFO\_FILTER\_LOG\_FILE).
- Bit 2: Enables user tracing. The default file for user tracing is aruser.log (see AR\_SERVER\_I\_NFO\_USER\_LOG\_FILE).
- Bit 3: Enables escalation tracing. The default file for escalation tracing is arescl.log (see AR\_SERVER\_I\_NFO\_ESCALATION\_LOG\_FILE).
- Bit 4: Enables API tracing. The default file for API tracing is arapi.log (see AR\_SERVER\_I\_NFO\_API\_LOG\_FILE).
- Bit 5: Enables thread tracing. The default file for thread tracing is arthread.log (see AR\_SERVER\_I\_NFO\_THREAD\_LOG\_FILE).
- Bit 6: Enables alert tracing for the arserverd process. The default file for alert tracing is aralert.log (see AR\_SERVER\_I\_NFO\_ALERT\_LOG\_FILE).
- Bit 7: Enables arforkd tracing for the arserverd process. The default file for arforkd tracing is arforkd.log (see AR\_SERVER\_I\_NFO\_ARFORK\_LOG\_FILE).
- Bit 8: Enables server group tracing. The default file for server group tracing is arsrvgrp.log (see AR\_SERVER\_I\_NFO\_SERVERGROUP\_LOG\_FILE).
- Bit 15: Enables distributed server tracing (applicable for Distributed Server Option (DSO) only). The default file for distributed server tracing is ardist.log (see AR\_SERVER\_I\_NFO\_DS\_LOG\_FILE).
- Bit 16: Enables approval server debugging.
- Bit 17: Enables plug-in tracing (applicable for external data sources only). The default file for plug-in tracing is arplugin.log (see AR\_SERVER\_I\_NFO\_PLUGIN\_LOG\_FILE).

## AR\_SERVER\_I\_NFO\_DEFAULT\_ORDER\_BY:

An integer (1 or 0) indicating the default order of requests in a form when issuing ARGetListEntry calls, so that the sorting is done on the database and before the results are returned. Valid values for this option are 0 (no sort order) or 1 (use EntryId order). The default value is 1.

AR\_SERVER\_I\_NFO\_DELAYED\_CACHE:

A flag indicating whether to disable recaching of the shared cache after a time-out (T) or not (F). The default value is F (delayed recache not disabled).

AR\_SERVER\_I\_NFO\_DFLT\_ALLOW\_CURRENCIES

Default list of allowable currencies when new currency fields are created.

AR\_SERVER\_I\_NFO\_DFLT\_FUNC\_CURRENCIES

Default list of functional currencies when new currency fields are created.

AR\_SERVER\_I\_NFO\_DISABLE\_ARCHIVE:

The value indicating whether archiving is disabled. Valid values for this option are 1 (Disabled) and 0 (Enabled).

AR\_SERVER\_I\_NFO\_DISABLE\_FTS\_INDEXER:

The value indicating whether the full text search (FTS) indexer is disabled. Valid values for this option are 1 (Disabled) and 0 (Enabled).

AR\_SERVER\_I\_NFO\_DISABLE\_NON\_UNICODE\_CLIENTS:

A flag indicating whether a Unicode server will decline calls from non-Unicode clients. Regardless of the setting, however, calls from BMC Remedy Administrator and BMC Remedy Alert are always accepted.

Valid values are as follows:

0: The Unicode server responds to calls from all clients.

nonzero: The Unicode server returns error status AR\_RETURN\_ERROR and corresponding error message AR\_ADMIN\_ONLY\_MODE to calls from non-Unicode clients.

AR\_SERVER\_I\_NFO\_DS\_LOGFILE:

The name of the file to use if distributed server tracing is turned on (see AR\_SERVER\_I\_NFO\_DEBUG\_MODE). Valid values for this option are 1 (TRUE) and 0 (FALSE).

AR\_SERVER\_I\_NFO\_DS\_MAPPING (get only):

The name of the form that contains distributed mapping definitions (applicable for DSO only).

AR\_SERVER\_I\_NFO\_DS\_PENDING (get only):

The name of the form that contains the pending operation list (applicable for DSO only).

AR\_SERVER\_I\_NFO\_DS\_RPC\_SOCKET:

The specific server socket to use for the distributed server. Valid values for this option are 390600 and 390680 through 390694. Any other value causes the distributed server to use the default server.

AR\_SERVER\_I\_NFO\_DS\_SVR\_LICENSE:

The distributed server license type (character string).

AR\_SERVER\_I\_NFO\_DSO\_DEST\_PORT:

The TCP port that the distributed server option uses to communicate with the AR System server.

AR\_SERVER\_I\_NFO\_DSO\_ERROR\_RETRY:

An integer specifying the error retry behavior for DSO.

- 0: Standard connection and transmission errors qualify for retry. This is the default setting (see AR\_DSO\_ERROR\_RETRY\_STANDARD).
- 1: No errors qualify for retry (see AR\_DSO\_ERROR\_RETRY\_NO\_RETRY).
- 2: All errors qualify for retry (see AR\_DSO\_ERROR\_RETRY\_ALL\_RETRY).
- 3: Standard connection and transmission errors plus database errors qualify for retry (see AR\_DSO\_ERROR\_RETRY\_STANDARD\_DB).

AR\_SERVER\_I\_NFO\_DSO\_MAX\_QUERY\_SIZE:

The maximum number of entries that the Distributed Server Option (DSO) retrieves in a single query to the server. The minimum value for this option is 1, and there is no upper boundary.

AR\_SERVER\_I\_NFO\_DSO\_MERGE\_STYLE:

An integer (1 or 0) indicating the merge style to use for the DSO. A 0 indicates the new style; 1 indicates the old style. The default is 0. For a 1 value (old style), the merge style overwrites entries; for a 0 value (new style), a true merge is performed according to the rules of the DSO setup. For backward compatibility, this server option can be set to 1.

**AR\_SERVER\_I NFO\_DSO\_RPCPROG\_NUM:**

The RPC program number settings that DSO uses to access the local server. The values for the settings for private servers are:

- 390600
- 390621 to 390634
- 390636 to 390669
- 390680 to 390694

**AR\_SERVER\_I NFO\_DSO\_TARGET\_PASSWD:**

The password for the target DSO server. During a `get` action, this option returns a list of servers that have been assigned a password. For security reasons, the actual password strings are not returned.

**AR\_SERVER\_I NFO\_DSO\_TI\_MEOU NORMAL:**

The number of seconds the DSO server waits before disconnecting inactive clients. The range of values is 60 to 21600 (60 seconds to 6 hours). The default value is 120.

**AR\_SERVER\_I NFO\_DSO\_USER\_PASSWD:**

The password for the DSO server. During a `get` action, this option returns a `NULL` if there is no password and returns an empty string if there is a password. For security reasons, the actual password string is not returned.

**AR\_SERVER\_I NFO\_EA\_GROUP\_MAPPING:**

Specifies mapping of LDAP groups to AR System groups for the purpose of external authentication. The value of this option consists of one or more pairs of group names separated by semicolons, as shown in the following example:

" LDAP-1 " " ARS-1 " ; " LDAP-2 " " ARS-2 " ; " LDAP-3 " " ARS-3 "

Use mapping as an alternative to creating an AR System group corresponding to each LDAP group. You can map each LDAP group to an AR System group (as in the example) or multiple LDAP groups to a single AR System group. (If you try to map a single LDAP group to multiple AR System groups, only the first mapping expression will be valid.)

This option can be combined with

`AR_SERVER_I NFO_EA_IGNORE_EXCESS_GROUPS`, if desired.

For more information, see *Integrating with Plug-ins and Third-Party Products*.

AR\_SERVER\_I\_NFO\_EA\_I\_GNORE\_EXCESS\_GROUPS:

A flag used by AR System during external authentication. A value of 0 indicates that the user is authenticated when there is a match between every LDAP group to which the user belongs and a corresponding group in AR System. A value of 1 indicates that the user is authenticated when any single LDAP group to which the user belongs matches a group in AR System, and the excess LDAP groups are ignored. The default value is 0.

This option can be combined with

AR\_SERVER\_I\_NFO\_EA\_GROUP\_MAPPING, if desired.

For more information, see *Integrating with Plug-ins and Third-Party Products*.

AR\_SERVER\_I\_NFO\_EA\_RPC\_SOCKET:

The RPC socket number on which an external authentication server awaits requests for authentication. A 0 value means external authentication will not be used. This attribute persists in the ar.cfg/ar.conf file. The default value is 0.

AR\_SERVER\_I\_NFO\_EA\_RPC\_TIMEOUT:

The RPC timeout (seconds) used when making calls to the authentication (AREA) server (for example, 30 seconds). This attribute persists in the ar.cfg/ar.conf file. The default value is 30 seconds.

AR\_SERVER\_I\_NFO\_EA\_SYNC\_TIMEOUT:

The internal (seconds) the AR System server uses to periodically invoke the external authentication server's AREANeedToSyncCallback() function, which instructs the AR System server to renew its internally stored user information in the event there are changes made to the source used to authenticate users. A 0 value means that the AR System server will not invoke the call to the external authentication (AREA) server. This attribute persists in the ar.cfg/ar.conf file. The default value is 300 seconds.

AR\_SERVER\_I\_NFO\_EMAIL\_AIX\_USE\_OLD\_EMAIL:

A flag indicating whether to use the old email or the new Email engine. A zero (0) setting indicates that the new email is used. A setting of one (1) indicates that the old email is used. (AIX only)

AR\_SERVER\_I\_NFO\_EMAIL\_FROM:

The sender name to use for filter-generated email notifications where no subject is specified. Only trusted email users can use this name (see documentation about the `/etc/sendmail.l.cf` file for more information about trusted users). (UNIX only)

AR\_SERVER\_I\_NFO\_EMAIL\_IMPORT\_FORM:

A flag indicating whether to enable or disable the import of the email forms. If set to zero (0), the forms are not imported. If set to one (1), the forms are imported.

AR\_SERVER\_I\_NFO\_EMAIL\_LINE\_LENGTH:

The maximum line length of e-mail messages.

AR\_SERVER\_I\_NFO\_EMAIL\_SYSTEM:

A character string that indicates the email system on UNIX.

AR\_SERVER\_I\_NFO\_EMAIL\_TIMEOUT:

(UNIX only) The length of time in seconds before the system closes the pipe to `sendmail.l` to unblock the AR System server. Valid values for this option are from 1 to 300. The default value is 10.

AR\_SERVER\_I\_NFO\_EMBEDDED\_SQL (get only):

A value indicating whether the underlying SQL database is embedded (bundled with AR System). Valid values for this option are 0 (not embedded) and 1 (embedded).

AR\_SERVER\_I\_NFO\_ERROR\_EXCEPTION\_LIST:

A list of error codes that will cause diagnostic output to be written to the error log when the server encounters the errors.

AR\_SERVER\_I\_NFO\_ESCALATION\_LOGFILE:

The name of the file to use if escalation tracing is turned on (see AR\_SERVER\_I\_NFO\_DEBUG\_MODE).

AR\_SERVER\_I\_NFO\_EXCEPTION\_OPTION

A value indicating the diagnostic output option for exceptions. A setting of 0 indicates that a stack trace should be included. A setting of 1 indicates that a stack trace should not be included.

**AR\_SERVER\_I\_NFO\_EXPORT\_VERSION (get only):**

An integer indicating the server's export version number.

- 3: Export version of AR System 3.x
- 4: Export version of AR System 4.0.3
- 5: Export version of AR System 4.5
- 6: Export version of AR System 5.0
- 7: Export version of AR System 5.1
- 8: Export version of AR System 6.0 and AR System 6.3

**AR\_SERVER\_I\_NFO\_EXT\_AUTH\_CAPABILITIES:**

A value indicating the return data capabilities for the AREA plug-in in use. This setting does not control the AREA plug-in. It only describes the behavior of the plug-in, allowing for server optimization.

- 1: No email address (see `AR_EXT_AUTH_DATA_NO_EMAIL_ADDR`).
- 2: No notify mechanism (see `AR_EXT_AUTH_DATA_NO_NOTIFY_MECH`).
- 4: No group identifiers (see `AR_EXT_AUTH_DATA_NO_GROUP_IDS`).
- 8: No license information (see `AR_EXT_AUTH_DATA_NO_LICENSE_INFO`).

**AR\_SERVER\_I\_NFO\_FILTER\_MAX\_STACK:**

The maximum number of levels of recursion allowed for a given operation. The data modification performed by an `AR_FILTER_ACTION_FILTER` filter action could trigger a second set, or level, of filters, one of which could trigger filters a third level down and so on. This option limits the number of times such recursion can happen, preventing the server crash that would occur if the recursion continued indefinitely. The default value is 25.

**AR\_SERVER\_I\_NFO\_FILTER\_MAX\_TOTAL:**

The maximum number of filters that the server will execute for a given operation. The default value is 10000.

**AR\_SERVER\_I\_NFO\_FILTER\_API\_RPC\_TIMEOUT:**

The time limit, in seconds, that is allowed for the filter API RPC to respond to the server's request before returning an error. The default value is 60 seconds. The minimum value is 0, and the maximum value is 300.

AR\_SERVER\_I NFO\_FI LTER\_LOG\_FI LE:

The name of the file to use if filter tracing is turned on (see AR\_SERVER\_I NFO\_DEBUG\_MODE).

AR\_SERVER\_I NFO\_FI XED\_LI CENSE (get only):

The number of fixed user licenses (int).

AR\_SERVER\_I NFO\_FLASH\_DAEMON:

A flag indicating whether Flashboards is installed (T) or not (F). The default value is 1 if the server has one or more Flashboards licenses, and 0 if the server has no licenses.

AR\_SERVER\_I NFO\_FLOAT\_LI CENSE (get only):

The number of floating write licenses defined.

AR\_SERVER\_I NFO\_FLOAT\_TI MEOUT:

The number of hours the server waits before disconnecting inactive users. If a user is holding a floating write license token, the system also frees the token at this time. The default value is 2 hours.

AR\_SERVER\_I NFO\_FLUSH\_LOG\_LI NES:

A value (1 (TRUE) and 0 (FALSE)) that indicates whether the server flushes every line written to a log to disk. If the line is not flushed to disk, the line is buffered by the file system.

AR\_SERVER\_I NFO\_FT\_CASE\_SENSITIVITY:

A flag specifying case sensitivity for the server's full text option. A value of 0 indicates that full text searching is case-sensitive. A value of 1 indicates that full text searching is not case-sensitive. The default value is 1.

AR\_SERVER\_I NFO\_FT\_COLLECTION\_DIR:

The location in the file system where full text indexes are stored.

AR\_SERVER\_I NFO\_FT\_CONFIGURATION\_DIR:

The location in the file system where search engine configuration files are stored.

**AR\_SERVER\_I NFO\_FT\_DI SABLE\_SEARCH:**

A value indicating the server's full text indexing state. Valid settings are 0 (full text indexing enabled) and 1 (full text indexing disabled). The default value is 0.

**AR\_SERVER\_I NFO\_FT\_OPTI MI ZE\_THRESHOLD:**

The number of additions, modifications, or deletions that must occur before the server will optimize an index. The default value is 1000. The minimum value is 10.

**AR\_SERVER\_I NFO\_FT\_RECOVERY\_I NTERVAL:**

The number of minutes the server waits between periodic attempts to index entries that unexpectedly failed to index in a prior attempt. The default value is 60.

**AR\_SERVER\_I NFO\_FT\_REI NDEX:**

A value specifying reindex operation on the server. A setting of 1 initiates a reindex operation on the server. On retrieval, this option will be set to 1 if a reindex operation is in progress. Otherwise, the setting is 0.

**AR\_SERVER\_I NFO\_FT\_SEARCH\_MATCH\_OP:**

The server's match option for full text search match option. Valid settings are as follows:

- 0: Force leading and trailing wildcard.
- 1: Force trailing wildcard.
- 2: Ignore leading wildcard.
- 3: Remove wildcards.
- 4: Unchanged.

The default value is 4.

**AR\_SERVER\_I NFO\_FT\_STOP\_WORDS:**

A list of words to be ignored during indexing. The words are separated by semicolons.

**AR\_SERVER\_I NFO\_FT\_TEMP\_DI R:**

The location where temporary files associated with the search engine are stored.

AR\_SERVER\_I NFO\_FT\_THRESHOLD\_HI GH:

During the processing of full-text-search data, the server combines results from sub-queries to generate a final set of results. If the number of rows created during processing exceeds this value, the server will return an error indicating that the search is too complex. The default value is 1,000,000.

AR\_SERVER\_I NFO\_FT\_THRESHOLD\_LOW:

During the processing of full-text-search data, the server creates a temporary table if the number of matches equals or exceeds this value. If the number of matches is under this value, the server will use the SQL IN operator for a query on an existing table. The default value is 200.

AR\_SERVER\_I NFO\_FTEXT\_FI XED (get only):

The number of fixed FTS licenses defined.

AR\_SERVER\_I NFO\_FTEXT\_FLOAT (get only):

The number of floating FTS licenses defined.

AR\_SERVER\_I NFO\_FTEXT\_TI MEOUT:

The number of hours the server waits before disconnecting inactive users with FTS licenses. If a user is holding a floating FTS license token, the system also frees the token at this time.

AR\_SERVER\_I NFO\_FTINDEXER\_LOG\_FI LE:

The file name used for the full text indexer log.

AR\_SERVER\_I NFO\_FULL\_HOSTNAME (get only):

The full DNS host name of the server (“long” machine name).

AR\_SERVER\_I NFO\_G\_CACHE\_CHANGE (get only):

The time of the last change to the group cache.

AR\_SERVER\_I NFO\_GUESTS\_RESTRICT\_READ:

Specifies whether guest users are granted a restricted read license.

Values for this option are 1 (TRUE) or 0 (FALSE). The default value is 0.

AR\_SERVER\_I NFO\_HARDWARE (get only):

The server hardware type (character string).

AR\_SERVER\_I NFO\_HOMEPAGE\_FORM:

The server homepage form (character string).

AR\_SERVER\_I NFO\_HOSTNAME (get only):

The host name of the server (“short” machine name).

AR\_SERVER\_I NFO\_I NFORMI X\_DBN (get only):

Informix database server name. This option has a value only if you are using an Informix database.

AR\_SERVER\_I NFO\_I NFORMI X\_RELAY\_MOD (get only):

A character string that indicates the Informix relay module.

AR\_SERVER\_I NFO\_I NFORMI X\_TBC (get only):

The Informix configuration file. This option has a value only if you are using an Informix database. The default value is `onconfig`.

AR\_SERVER\_I NFO\_I P\_NAME:

A string that contains multiple names for the server, separated by semicolons.

AR\_SERVER\_I NFO\_JAVA\_VM\_OPTI ONS:

The Java Virtual Machine (VM) options.

AR\_SERVER\_I NFO\_LB\_MAP\_I P\_ADDR:

IP address map between the load balancer and server.

AR\_SERVER\_I NFO\_LOCALIZED\_SERVER:

A variable indicating whether the server is localized. Values for this option are 1 (TRUE) and 0 (FALSE). The default value is 1 (server is localized).

AR\_SERVER\_I NFO\_LOGFILE\_APPEND:

A value (1 (TRUE) or 0 (FALSE)) that indicates whether to create a separate \*.bak file or to append to the existing log file. A 0 value creates a \*.bak file; a 1 value indicates that new log information be appended to the existing file. The default value is 0 (creates a \*.bak file).

AR\_SERVER\_I NFO\_MAX\_BAD\_PWD\_ATTEMPTS:

The maximum number of password attempts allowed. The default value is 3. A value of 0 turns the feature off, and any positive integer sets the limit.

AR\_SERVER\_I\_NFO\_MAX\_ENTRIES:

The maximum number of entries returned by a single query. Because users can also specify the maximum number of entries returned (in query preferences), the actual maximum is the lower of these two values. The default value is no (server-defined) maximum.

AR\_SERVER\_I\_NFO\_MAX\_LOG\_FILE\_SIZE:

The maximum size for system log files. The default value is 0 (no limit).

AR\_SERVER\_I\_NFO\_MAX\_PASSWORD\_ATTEMPTS:

The maximum number of times you can attempt to log in with an invalid password. If this number is exceeded, the account is locked until the password is reset by the user or an administrator. Valid values are 0 (unlimited) or any positive integer. The default value is 0.

AR\_SERVER\_I\_NFO\_MAX\_SCHEMAS (get only):

The maximum number of forms allowed. The default value is F (no limit).

AR\_SERVER\_I\_NFO\_MESSAGE\_CAT\_SCHEMA (get only):

The name of the BMC Remedy Message Catalog form.

AR\_SERVER\_I\_NFO\_MID\_TIER\_PASSWD:

The password that the mid tier uses to access the AR System server. During a get action, this option returns a NULL if there is no password and returns an empty string if there is a password. For security reasons, the actual password string is not returned.

AR\_SERVER\_I\_NFO\_MINIMUM\_API\_VERSION:

A value indicating the minimum API version that the server supports. The API versions are:

- 5: API version of AR System 3.x
- 6: API version of AR System 4.0.3
- 7: API version of AR System 4.5
- 8: API version of AR System 5.0
- 9: API version of AR System 5.1
- 10: API version of AR System 6.0
- 11: API version of AR System 6.3

**AR\_SERVER\_I\_NFO\_MULTI\_SERVER (get only):**

An integer (0/1) value indicating the use of the multi-server option. A 0 value indicates a single server; a 1 value indicates the use of the multi-server option.

**AR\_SERVER\_I\_NFO\_NEXT\_I\_D\_BLOCK\_SIZE:**

Specifies the block size at which entryIDs for all schemas will be allocated. This setting is used to enhance the performance of the CREATE ENTRY operation.

Values for this option are as follows:

- 1: EntryIDs values increase consistently by one, and there are no gaps in the entryID sequence.
- 2 to 1000: EntryIDs will be allocated in blocks of this size.

The default value is 1. If an invalid setting is encountered, the option is forced to a setting of 1.

You can override AR\_SERVER\_I\_NFO\_NEXT\_I\_D\_BLOCK\_SIZE for a specific schema by using the NextID-Block-Size schema property in BMC Remedy Administrator.

AR\_SERVER\_I\_NFO\_NEXT\_I\_D\_BLOCK\_SIZE does not work on the “distributed pending” schema. Correct operation of Distributed Server Option (DSO) requires entryIDs to be specified in the order in which they were written.

Use of AR\_SERVER\_I\_NFO\_NEXT\_I\_D\_BLOCK\_SIZE might result in gaps in the entryID sequence. A gap will occur if an entryID transaction completes successfully, but the remainder of the operation is rolled back. The entryID would be lost because it cannot be returned to the system.

**AR\_SERVER\_I\_NFO\_NEXT\_I\_D\_COMMIT:**

During a CREATE ENTRY operation for a schema, specifies whether to commit the current transaction immediately after an entryID is retrieved and updated, and start a new one for the remainder of the operation. Values are 1 (TRUE) and 0 (FALSE). If this setting is not specified in the ar.conf file, the default value is FALSE.

If this option is set to TRUE, then during creation of an entry, the existing transaction is committed and a new transaction is started immediately after the entryID has been assigned. This behavior allows other pending transactions to proceed sooner.

**AR\_SERVER\_I\_NFO\_NEXT\_ID\_COMMIT** provides a small performance improvement at the expense of potential gaps in the entryID sequence. A gap will occur if an entryID transaction completes successfully, but the remainder of the operation is rolled back. The entryID would be lost because it cannot be returned to the system.

Instead of AR\_SERVER\_I\_NFO\_NEXT\_ID\_COMMIT, you might prefer to use AR\_SERVER\_I\_NFO\_NEXT\_ID\_BLOCK\_SIZE. The latter option is more flexible and offers a more significant enhancement in performance.

**AR\_SERVER\_I\_NFO\_NOTIFY\_WEB\_PATH:**

The default WWW path for URL shortcuts embedded in e-mail notifications.

**AR\_SERVER\_I\_NFO\_NFY\_TCP\_PORT:**

The TCP port that the server uses.

**AR\_SERVER\_I\_NFO\_ORACLE\_CLOB\_STORE\_IN\_ROW:**

Specifies whether CLOBs in the Oracle database will be stored IN ROW. Values for this option are 0 (DISABLE STORAGE IN ROW) and 1 (ENABLE STORAGE IN ROW). The default value is 0.

It is strongly recommended that diary fields and character fields with unlimited length should be created with IN ROW storage disabled.

**AR\_SERVER\_I\_NFO\_ORACLE\_SID (get only):**

The system ID of the Oracle database you are accessing. This option has a value only if you are using an Oracle database.

**AR\_SERVER\_I\_NFO\_ORACLE\_TWO\_TASK (get only):**

The two-task environment setting for remote access to an Oracle database. This option has a value only if you are using an Oracle database.

**AR\_SERVER\_I\_NFO\_OS (get only):**

A character string that indicates the server operating system that includes the version number.

## AR\_SERVER\_I\_NFO\_PER\_THREAD\_LOGS:

A value (1 (TRUE) and 0 (FALSE)) that indicates whether the server creates a log for each worker thread. A value of zero means the worker threads use the shared logs. A value of 1 causes each thread to create its own copy of the shared logs.

## AR\_SERVER\_I\_NFO\_PLUGI\_N\_ALIAS:

A list of plug-in aliases.

## AR\_SERVER\_I\_NFO\_PLUGI\_N\_DEFAULT\_TIMEOUT:

The number of seconds the server waits for a response from the plug-in server. The range of values is 60 to 600 (60 seconds to 10 minutes). The default value is 60.

## AR\_SERVER\_I\_NFO\_PLUGI\_N\_LOGFILE:

The name of the plug-in service log file (see AR\_SERVER\_I\_NFO\_DEBUG\_MODE).

## AR\_SERVER\_I\_NFO\_PLUGI\_N\_LOG\_LEVEL:

An integer specifying the logging level for plug-ins. Values range from 100 to 10,000. The default value is 10,000 (OFF).

For descriptions of the predefined logging levels, see “Plug-in logging facility” on page 496.

## AR\_SERVER\_I\_NFO\_PLUGI\_N\_PASSWORD:

The plug-in service password. During a `get` action, this option returns a `NULL` if there is no password and returns an empty string if there is a password. For security reasons, the actual password string is not returned.

## AR\_SERVER\_I\_NFO\_PLUGI\_N\_TARGET\_PASSWORD:

The password that the AR System server uses to communicate with a plug-in service that runs at the host name and port number specified. During a `get` action, this option returns a list of servers that have been assigned a password. For security reasons, the actual password strings are not returned.

**AR\_SERVER\_I\_NFO\_PREF\_SERVER\_OPTI ON:**

An integer specifying the preference server behavior for this server, to be honored by the client. The default value is 1.

- 1: User-defined value.
- 2: Always use this server.
- 3: Do not use this server.

**AR\_SERVER\_I\_NFO\_PS\_RPC\_SOCKET:**

The RPC program number and port pairs the Private Servers will use. A value of 0 for the port means no port was specified.

**AR\_SERVER\_I\_NFO\_REGI STER\_PORTMAPPER:**

A flag indicating whether to register with BMC Remedy Portmapper (T) or not (F). The default value is 1.

**AR\_SERVER\_I\_NFO\_Rem\_SERV\_I D (get only):**

The AR System server ID associated with the license (character string).

**AR\_SERVER\_I\_NFO\_RPC\_NON\_BLOCKI NG\_I O:**

A flag indicating whether the server is in RPC non blocking I/O mode or not. Valid values are 1 (TRUE) and 0 (FALSE). The default value is 0 (FALSE). For more information, see the *Configuring* guide.

**AR\_SERVER\_I\_NFO\_SAVE\_LOGI N:**

A value indicating whether users must log in to clients.

- 0: Controlled by user (default setting).
- 1: Force no login (controlled by the administrator).
- 2: Force login (controlled by the administrator).

**AR\_SERVER\_I\_NFO\_SCC\_COMMENT\_CHECKI N:**

An integer (0 or 1) value indicating whether a source code control integration requires you to enter a comment at check-in time. The default 1 value means no comment is required.

**AR\_SERVER\_I\_NFO\_SCC\_COMMENT\_CHECKOUT:**

An integer (0 or 1) value indicating whether a source code control integration requires you to entire a comment at checkout time. A default 1 value means no comment is required.

**AR\_SERVER\_I\_NFO\_SCC\_ENABLED:**

A value indicating whether a source code control system is being used with the AR System. An F value (default) means a source code control system is not present.

**AR\_SERVER\_I\_NFO\_SCC\_I\_INTEGRATION\_MODE:**

An integer (T or F) value indicating the level of source code control integration. An F (the default) value indicates an advisory level of integration, which means you can modify and save an object without having it checked out from the source code control application. A T value means enforced integration, which has strict rules for controlling source files (for example, enforced integration means that other administrators will not be able to modify and save changes on an object that you have checked out).

**AR\_SERVER\_I\_NFO\_SCC\_PROVIDER\_NAME:**

A character string for the source code control system provider name. If none is present, this value is NULL. This string is limited to 255 characters.

**AR\_SERVER\_I\_NFO\_SCC\_TARGET\_DIR:**

A character string for the source code control system target directory. If none is present, this value is NULL. This string is limited to 255 characters.

**AR\_SERVER\_I\_NFO\_SERVER\_DIR (get only):**

The data directory for the AR System. This directory contains support files for the underlying SQL database (but no actual database files). For flat file databases, this directory contains both database definition and data files.

**AR\_SERVER\_I\_NFO\_SERVER\_IDENT (get only):**

The unique identifier for the server (character string).

**AR\_SERVER\_I\_NFO\_SERVER\_LANG (get only):**

The local language setting of the server.

**AR\_SERVER\_I\_NFO\_SERVER\_LICENSE (get only):**

The server license type (character string).

## AR\_SERVER\_I NFO\_SERVER\_NAME:

An alias for the current server name. The AR System server provides this name when it receives a request for the server name. The value for this field is the name without the domain extension. The AR System server automatically appends the current domain name to the name. For example, if the server is in the domain arrow.com, and the value for this setting is al pha, the name is al pha.arrow.com. Note that a setting of al pha.arrow.com results in an unrecognizable name:  
al pha.arrow.com.arrow.com.

See also Server-Name in the ar.conf file.

## AR\_SERVER\_I NFO\_SERVERGROUP\_I NTERVAL

The interval (in seconds) for server group checking.

## AR\_SERVER\_I NFO\_SERVERGROUP\_LOG\_FI LE:

The file name used for the server group log.

## AR\_SERVER\_I NFO\_SERVERGROUP\_MEMBER:

The value indicating whether the server is a server group member.

0: Not a server group member.

1: A server group member.

## AR\_SERVER\_I NFO\_SERVERGROUP\_NAME:

The server group name.

## AR\_SERVER\_I NFO\_SET\_PROC\_TI ME:

The number of seconds the server waits before ending a set fields process that has not completed. Valid values for this option are 1 through 20. The default value is five seconds.

## AR\_SERVER\_I NFO\_SG\_EMAI L\_STATE (get only):

The server group email state.

0: No server group.

1: Suspended.

2: Resumed.

AR\_SERVER\_I NFO\_SG\_FLASHBOARDS\_STATE (get only):

The server group flashboards state.

0: No server group.

1: Suspended.

2: Resumed.

AR\_SERVER\_I NFO\_SQL\_LOG\_FILE:

The name of the file to use if SQL tracing is turned on (see AR\_SERVER\_I NFO\_DEBUG\_MODE).

AR\_SERVER\_I NFO\_SSTABLE\_CHUNK\_SIZE:

The maximum number of entries fetched during server side table processing. Zero (0) is unlimited.

AR\_SERVER\_I NFO\_STRUCT\_CHANGE (get only):

The last structure change (for this run of the server).

AR\_SERVER\_I NFO\_SUBMITTER\_MODE:

A value indicating whether the Submitter field can be changed and whether the submitter of an entry must have a license to modify it.

In *locked* mode:

the Submitter field cannot be changed after submit.

the submitter with or a read or a write license, but not with a restricted read license, can modify the entry (if the Submitter group has Change permission).

In *changeable* mode:

the Submitter field can be changed after submit, but the submitter must have a write license to modify the entry (if the Submitter group has Change permission).

Valid values for this option are 1 (locked) and 2 (changeable). The default value is 2 (changeable).

AR\_SERVER\_I NFO\_SUPPRESS\_WARN:

A series of zero or more message numbers (separated by spaces) that identify the informational or warning messages that the system suppresses.

## AR\_SERVER\_INFO\_SVR\_EVENT\_LIST:

A character string that consists of a list of numeric event types separated by semicolons. If this tag is not set or does not include event types, the server will not create an event schema or record schema events. If the server does not recognize an event type, it will ignore that element. When this option has a value of `NULL`, the server will remove the event list.

1: AR_SVR_EVENT_CHG_SCHEMA	Logs changes to schemas.
2: AR_SVR_EVENT_CHG_FIELD	Logs changes to fields.
3: AR_SVR_EVENT_CHG_CHARMENU	Logs changes to character menus.
4: AR_SVR_EVENT_CHG_FILTER	Logs changes to filters.
5: AR_SVR_EVENT_CHG_IMPORT	Logs changes to imports.
6: AR_SVR_EVENT_CHG_ACTLINK	Logs changes to active links.
7: AR_SVR_EVENT_CHG_ESCAL	Logs changes to escalations.
8: AR_SVR_EVENT_CHG_VUI	Logs changes to VUIs.
9: AR_SVR_EVENT_CHG_CONTAINER	Logs changes to containers.
10: AR_SVR_EVENT_CHG_USERS	Logs the users added, modified, or deleted.
11: AR_SVR_EVENT_CHG_GROUPS	Logs the groups added, modified, or deleted.
12: AR_SVR_EVENT_CHG_SVR_SETTINGS	Logs changes to server settings.
13: AR_SVR_EVENT_CHG_ALERT_USERS	Logs changes to user alert messages.
14: AR_SVR_EVENT_ARCHIVE_DONE	Logs the status of archive for a form.
15: AR_SVR_EVENT_SERVGROUP_ACTION	Logs server group events.

## AR\_SERVER\_INFO\_SVR\_STATS\_REC\_INTERVAL:

The time interval, in seconds, that the server records statistics into the server statistics form.

## AR\_SERVER\_I NFO\_SVR\_STATS\_REC\_MODE:

An integer that specifies the type of recording modes that the server uses to record server statistics. This determines if statistics are recorded in the server statistics form.

0: AR_SVR_STATS_RECMODE_OFF	The server does not record statistics.
1: AR_SVR_STATS_RECMODE_CUMUL_ONLY	The server records only cumulative queue statistics.
2: AR_SVR_STATS_RECMODE_CUMUL_QUEUE	The server records both cumulative and individual queue statistics.

## AR\_SERVER\_I NFO\_SYBASE\_CHARSET (get only):

The character set being used to access a Sybase database. This option has a value only if you are using a Sybase database.

## AR\_SERVER\_I NFO\_SYBASE\_SERV (get only):

The Sybase logical server name. This option has a value only if you are using a Sybase database. The default value is SYBASE.

## AR\_SERVER\_I NFO\_SYS\_LOGGING\_OPTIONS:

A value indicating the logging options for operating system logging.

- 1: Suppress logging to the system log file (see AR\_SYSTEM\_LOGGING\_OPTION\_NONE).

## AR\_SERVER\_I NFO\_TCD\_TCP\_PORT:

The TCP port that the AR System server uses. The dispatcher is randomly assigned to an available port if you do not specify this option.

## AR\_SERVER\_I NFO\_THREAD\_LOGFILE:

The name of the file to use if thread tracing is turned on (see AR\_SERVER\_I NFO\_DEBUG\_MODE).

## AR\_SERVER\_I NFO\_TWO\_DIGIT\_YEAR\_CUTOFF:

An integer indicating the two-digit year cutoff.

## AR\_SERVER\_I NFO\_U\_CACHE\_CHANGE (get only):

The time of the last change to the user cache.

AR\_SERVER\_I\_NFO\_UNQUAL\_QUERIES:

A flag indicating whether the server allows unqualified queries. Unqualified queries are ARGetListEntry calls in which the qualifier parameter is either NULL or has an operation value of 0 (AR\_COND\_OP\_NONE). These queries can cause performance problems, especially for large forms, because they return all entries for a given form. If not allowed, you can return all form entries by specifying a “dummy” qualification such as  $1 = 1$ . Valid values for this option are 1 (TRUE) and 0 (FALSE). The default value is T (allow unqualified queries).

AR\_SERVER\_I\_NFO\_USE\_ETC\_PASSWD:

A flag indicating whether the /etc/passwd file is used to validate users not registered with the AR System (UNIX only). If so, users in /etc/passwd are considered valid users of the AR System and are assigned to a group identified by the UNIX group ID. Valid values for this option are 1 (TRUE) and 0 (FALSE). The default value is 1 (use password file).

AR\_SERVER\_I\_NFO\_USER\_CACHE\_UTILS:

A flag indicating whether the arcache and arrel load utilities are enabled. Valid values for this option are 0 (disabled) and 1 (enabled).

AR\_SERVER\_I\_NFO\_USER\_LOGFILE:

The name of the file to use if user tracing is turned on (see AR\_SERVER\_I\_NFO\_DEBUG\_MODE).

AR\_SERVER\_I\_NFO\_VERSION (get only):

The AR System server version (character string).

AR\_SERVER\_I\_NFO\_XREF\_PASSWORDS:

A flag indicating whether the /etc/passwd file is cross-referenced if an AR System user has no password (UNIX only). This option enables you to manage group membership and other support information by using the AR System but still manage passwords using the /etc/passwd file. Valid values for this option are 1 (TRUE) and 0 (FALSE). The default value is 0 (blank passwords are not cross referenced).

**See also** ARGetServerStatistics, ARSetServerInfo. See FreeAR for:  
FreeARServerInfoList, FreeARServerInfoRequestList, FreeARStatusList.

# ARGetSessionConfiguration

**Description** Retrieves a public API session variable.

**Privileges** AR System administrator.

**Synopsis**

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

int ARGetSessionConfiguration(
    ARControlStruct *control,
    unsigned int variablenumber,
    ARValueStruct *variablevalue,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The **user** and **sessionid** fields are required.

**variableId**

Identification for the variable type. Identification of the variable type:

- 1: Maximum size for a single response (AR\_SESS\_CHUNK\_RESPONSE\_SIZE).
- 2: Timeout for normal operations (AR\_SESS\_TIMELIMIT\_NORMAL).
- 3: Timeout for long operations (AR\_SESS\_TIMELIMIT\_LONG).
- 4: Timeout for extra long operations (AR\_SESS\_TIMELIMIT\_XLONG).
- 5: Socket number to lock to (AR\_SESS\_LOCK\_TO\_SOCKET\_NUMBER).
- 6: Flag with Boolean value indicating if the session is pooled (AR\_SESS\_POOLED).
- 7: API program client type (AR\_SESS\_CLIENT\_TYPE).
- 8: Type of VUI view (AR\_SESS\_VUI\_TYPE).
- 9: Flag with Boolean value indicating if the user would like to override the session from the previous IP (AR\_SESS\_OVERRIDE\_IP).
- 10: The name of the API command log (AR\_SESS\_API\_CMD\_LOG).
- 11: The name of the API result log (AR\_SESS\_API\_RES\_LOG).

**variableValue**

Configuration variable value:

**AR\_SESS\_CHUNK\_RESPONSE\_SIZE (integer)**

An integer value for the maximum data packet size returned from the server to the client in one transmission.

**AR\_SESS\_TIMEOUT\_NORMAL (integer)**

An integer value for the client timeout value used in fast server operations.

**AR\_SESS\_TIMEOUT\_LONG (integer)**

An integer value for the client timeout value used in list server operations.

**AR\_SESS\_TIMEOUT\_XLONG (integer)**

An integer value for the client timeout value used in definition updating and in import/export operations.

**AR\_SESS\_LOCK\_TO\_SOCKET\_NUMBER (integer)**

An integer value for the socket number that the client will lock to for all server interaction.

**AR\_SESS\_POOLED (integer)**

An integer value for the flag indicating if the session is pooled. Valid values for this option are 0 (No) and 1 (Yes).

**AR\_CLIENT\_TYPE\_\* (integer)**

An integer value for the client type. For more information, see AR\_CLIENT\_TYPE\_\* in the ar.h file.

**Range of values for user-defined, client-type variables:**

AR\_CLIENT\_TYPE\_END\_OF\_RESERVED\_RANGE: >5000.

**AR\_SESS\_VUI\_TYPE (integer)**

An integer value for the type of VUI. For more information, see AR\_VUI\_TYPE\_\* in the ar.h file.

**AR\_SESS\_OVERRIDE\_PREV\_IP (integer)**

An integer value for the flag indicating if the session can be overridden. Valid values for this option are 0 (No) and 1 (Yes).

**AR\_SESS\_API\_CMD\_LOG (char)**

A character value for the name of the API client side logging command file.

**AR\_SESS\_API \_RES\_LOG (char)**

A character value for the name of the API client side logging result file.

**Return values** [status](#)

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARSetSessionConfiguration.

## ARGetServerStatistics

**Description** Retrieves the requested statistics for the specified server. The counts returned generally represent the number of occurrences since starting the server. If a statistic reaches the maximum value for a long integer, the system resets the counter and begins incrementing again.

**Privileges** All users.

**Synopsis**

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

int ARGetServerStatistics(
    ARControlStruct          *control,
    ARServerInfoRequestList  *requestList,
    ARServerInfoList          *serverInfo,
    ARStatusList               *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**requestList**

A list of one or more statistics to return (see the Statistics Options section that follows).

**Return values****[serverInfo](#)**

The statistics retrieved from the server (see the Statistic Options that follows). The system returns `NULL` (`AR_DATA_TYPE_NULL`) for statistics not retrieved due to error.

**[status](#)**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**Statistics options****`AR_SERVER_STAT_API_REQUESTS` (integer):**

The total number of API requests received.

**`AR_SERVER_STAT_API_TIME` (integer):**

The total processor time (in 1/100 of a second) spent performing API function calls.

**`AR_SERVER_STAT_BAD_PASSWORD` (integer):**

The total number of times an incorrect password was specified during login.

**`AR_SERVER_STAT_CACHE_TIME` (integer):**

The total processor time (in 1/100 of a second) spent loading the internal cache to improve performance.

**`AR_SERVER_STAT_CPU` (integer):**

The total CPU time (in 1/100 of a second) used by the server.

**`AR_SERVER_STAT_CREATE_E_COUNT` (integer):**

The total number of calls made to the `ARCreateEntry` function.

**`AR_SERVER_STAT_CREATE_E_TIME` (integer):**

The total processor time (in 1/100 of a second) spent performing the `ARCreateEntry` function.

**`AR_SERVER_STAT_CURRENT_USERS` (integer):**

The total number of users currently accessing the system.

**`AR_SERVER_STAT_DELETE_E_COUNT` (integer):**

The total number of calls made to the `ARDel eteEntry` function.

AR\_SERVER\_STAT\_DELETE\_E\_TIME (**integer**):

The total processor time (in 1/100 of a second) spent performing the ARDeleteEntry function.

AR\_SERVER\_STAT\_ENTRY\_TIME (**integer**):

The total processor time (in 1/100 of a second) spent performing API function calls that manipulate entries.

AR\_SERVER\_STAT\_ESCL\_DISABLED (**integer**):

The total number of escalations that were evaluated but skipped because they were marked disabled.

AR\_SERVER\_STAT\_ESCL\_SKIPPED (**integer**):

The total number of escalations that were skipped (qualification criteria not met).

AR\_SERVER\_STAT\_ESCL\_FILED (**integer**):

The total number of push fields escalation actions performed.

AR\_SERVER\_STAT\_ESCL\_SET (**integer**):

The total number of set fields escalation actions performed.

AR\_SERVER\_STAT\_ESCL\_FILED\_FLTAPI (**integer**):

The number of escalations that modify fields through the API.

AR\_SERVER\_STAT\_ESCL\_FILED\_PROCESS (**integer**):

The number of escalations that modify fields.

AR\_SERVER\_STAT\_ESCL\_FILED\_SQL (**integer**):

The number of escalations that modify SQL fields.

AR\_SERVER\_STAT\_ESCL\_LOG (**integer**):

The total number of log escalation actions performed.

AR\_SERVER\_STAT\_ESCL\_NOTIFY (**integer**):

The total number of notify escalation actions performed.

AR\_SERVER\_STAT\_ESCL\_PASSED (**integer**):

The total number of escalations that were executed (qualification criteria met).

AR\_SERVER\_STAT\_ESCL\_PROCESS (**integer**):

The total number of run process escalation actions performed.

AR\_SERVER\_STAT\_ESCL\_SQL (**integer**):

The total number of direct SQL escalation actions performed.

AR\_SERVER\_STAT\_ESCL\_TIME (**integer**):

The total processor time (in 1/100 of a second) spent checking and processing escalations.

AR\_SERVER\_STAT\_E\_STATS\_COUNT (**integer**):

The total number of calls made to the ARGetEntryStatistics function.

AR\_SERVER\_STAT\_E\_STATS\_TIME (**integer**):

The total processor time (in 1/100 of a second) spent performing the ARGetEntryStatistics function.

AR\_SERVER\_STAT\_FILTER\_CALL\_GUIDE (**integer**):

The number of filter guide actions that call another process.

AR\_SERVER\_STAT\_FILTER\_DISABLED (**integer**):

The total number of filters that were evaluated but skipped because they were marked disabled.

AR\_SERVER\_STAT\_FILTER\_EXIT\_GUIDE (**integer**):

The number of filter guide actions that are terminated.

AR\_SERVER\_STAT\_FILTER\_FAILED (**integer**):

The total number of filters that were skipped (qualification criteria not met).

AR\_SERVER\_STAT\_FILTER\_FILEDP (**integer**):

The total number of push fields filter actions performed.

AR\_SERVER\_STAT\_FILTER\_FIELDSDS (**integer**):

The total number of set fields filter actions performed.

AR\_SERVER\_STAT\_FILTER\_FIELDSDS\_SQL (**integer**):

The number of filters that modify SQL fields.

AR\_SERVER\_STAT\_FILTER\_FIELDSDS\_FLTAPI\_70 (**integer**):

The number of filter actions that modify fields through the API.

AR\_SERVER\_STAT\_FILTER\_ELDs\_PROCESS (**integer**):

The number of filter actions that modify fields.

AR\_SERVER\_STAT\_FILTER\_GOTO\_ACTION (**integer**):

The number of filter actions that branch to another process.

AR\_SERVER\_STAT\_FILTER\_LOG (**integer**):

The total number of log filter actions performed.

AR\_SERVER\_STAT\_FILTER\_MESSAGE (**integer**):

The total number of message filter actions performed.

AR\_SERVER\_STAT\_FILTER\_NOTIFY (**integer**):

The total number of notify filter actions performed.

AR\_SERVER\_STAT\_FILTER\_PASSED (**integer**):

The total number of filters that were executed (qualification criteria met).

AR\_SERVER\_STAT\_FILTER\_PROCESS (**integer**):

The total number of run process filter actions performed.

AR\_SERVER\_STAT\_FILTER\_SQL (**integer**):

The total number of direct SQL filter actions performed.

AR\_SERVER\_STAT\_FILTER\_TIME (**integer**):

The total processor time (in 1/100 of a second) spent checking and processing filters.

AR\_SERVER\_STAT\_FTS\_SRCH\_COUNT (**integer**):

The total number of FTS operations performed.

AR\_SERVER\_STAT\_FTS\_SRCH\_TIME (**integer**):

The total processor time (in 1/100 of a second) spent performing FTS operations.

AR\_SERVER\_STAT\_FULL\_FIXED (**integer**):

The total number of connected users with fixed FTS licenses.

AR\_SERVER\_STAT\_FULL\_FLOATING (**integer**):

The total number of connected users with floating FTS licenses.

AR\_SERVER\_STAT\_FULL\_NONE (**integer**):

The total number of connected users with no FTS license.

AR\_SERVER\_STAT\_GET\_E\_COUNT (**integer**):

The total number of calls made to the ARGetEntry function.

AR\_SERVER\_STAT\_GET\_E\_TIME (**integer**):

The total processor time (in 1/100 of a second) spent performing the ARGetEntry function.

AR\_SERVER\_STAT\_GETLIST\_E\_COUNT (**integer**):

The total number of calls made to the ARGetListEntry function.

AR\_SERVER\_STAT\_GETLIST\_E\_TIME (**integer**):

The total processor time (in 1/100 of a second) spent performing the ARGetListEntry function.

AR\_SERVER\_STAT\_IDLE\_TIME (**integer**):

The total idle time (in 1/100 of a second) when the server is not processing any requests.

AR\_SERVER\_STAT\_MERGE\_E\_COUNT (**integer**):

The total number of calls made to the ARMergeEntry function.

AR\_SERVER\_STAT\_MERGE\_E\_TIME (**integer**):

The total processor time (in 1/100 of a second) spent performing the ARMergeEntry function.

AR\_SERVER\_STAT\_NET\_RESP\_TIME (**integer**):

The total time (in 1/100 of a second) spent on the network responding to the client.

AR\_SERVER\_STAT\_NO\_FULL\_TOKEN (**integer**):

The total number of times a user tried to connect and no floating FTS token was available.

AR\_SERVER\_STAT\_NO\_WRITE\_TOKEN (**integer**):

The total number of times a user tried to connect and no floating write token was available.

AR\_SERVER\_STAT\_NUM\_THREADS (**integer**):

The number of threads in a particular queue.

**AR\_SERVER\_STAT\_NUMBER\_BLOCKED (integer):**

The total number of blocked processes. This statistic, in conjunction with the AR\_SERVER\_STAT\_TIMES\_BLOCKED statistic, enables you to determine the approximate number of processes being blocked per instance. For example, if AR\_SERVER\_STAT\_TIMES\_BLOCKED is three and AR\_SERVER\_STAT\_NUMBER\_BLOCKED is six, the six blocked processes could be distributed in the following possible ways:

- Two blocked processes in each of the three instances ( $2+2+2 = 6$ ).
- Three blocked processes in one instance, two blocked processes in one instance, and one blocked process in one instance ( $3+2+1 = 6$ ).
- Four blocked processes in one instance and one blocked process in both of the other two instances ( $4+1+1 = 6$ ).

In this example, the blocked processes could not be distributed as five in one instance, one in one instance, and none in one instance because, by definition, the number of instances represents those times when at least one process is blocked.

**AR\_SERVER\_STAT\_OTHER\_TIME (integer):**

The total processor time (in 1/100 of a second) spent performing API function calls that do not manipulate entries or restructure the database.

**AR\_SERVER\_STAT\_RESTRUCT\_TIME (integer):**

The total processor time (in 1/100 of a second) spent performing API function calls that restructure the database.

**AR\_SERVER\_STAT\_WRITE\_RESTRICTED\_READ (integer):**

The total number of connected users with restricted read licenses.

**AR\_SERVER\_STAT\_SET\_E\_COUNT (integer):**

The total number of calls made to the ARSetEntry function.

**AR\_SERVER\_STAT\_SET\_E\_TIME (integer):**

The total processor time (in 1/100 of a second) spent performing the ARSetEntry function.

`AR_SERVER_STAT_SI NCE_START` (integer):

The number of seconds (in 1/100 of a second) since the server was started. That is, the total real time that the server process has been running.

`AR_SERVER_STAT_SQL_DB_COUNT` (integer):

The total number of SQL commands sent to the database.

`AR_SERVER_STAT_SQL_DB_TI ME` (integer):

The total processor time (in 1/100 of a second) spent performing SQL commands.

`AR_SERVER_STAT_START_TI ME` (time stamp):

The UNIX time at which this server was started.

`AR_SERVER_STAT_TI MES_BLOCKED` (integer):

The total number of instances in which at least one process was blocked by the current process. This statistic, in conjunction with the `AR_SERVER_STAT_NUMBER_BLOCKED` statistic, enables you to determine the approximate number of processes being blocked per instance.

`AR_SERVER_STAT_WRI TE_FI XED` (integer):

The total number of connected users with fixed write licenses.

`AR_SERVER_STAT_WRI TE_FLOATI NG` (integer):

The total number of connected users with floating write licenses.

`AR_SERVER_STAT_WRI TE_READ` (integer):

The total number of connected users with no write license.

**See also** `ARGetServerInfo`. See FreeAR for: `FreeARServerInfoList`, `FreeARServerInfoRequestList`, `FreeARStatusList`.

## ARGetSupportFile

**Description** Retrieves a file supporting external reports.

**Privileges** Any user who has access to the specified object.

**Synopsis**

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

int ARGetSupportFile(
    ARControlStruct *control,
    unsigned int fileType,
    ARNameType name,
    ARInternalId id2,
    ARInternalId fileId,
    FILE *filePtr,
    ARTimestamp timeStamp,
    ARStatusList *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**fileType**

The numerical value for the type of file, and the type of object the file is associated with. Specify 1 (`AR_SUPPORT_FILE_EXTERNAL_REPORT`) for an external report file associated with an active link.

**name**

The name of the object the file is associated with, usually a form.

**id2**

The ID of the field or VUI, if the object is a form. If the object is not a form, set this parameter to 0.

**fileId**

The unique identifier of a file within its object.

**filePtr**

A pointer to a file into which the support file contents will be retrieved. Specify `NULL` for this parameter if you do not want to retrieve the file contents. If you are using Windows, you must open the file in binary mode.

**Return values****timeStamp**

A time stamp identifying the last change to the field. Specify `NULL` for this parameter if you do not want to retrieve this value.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

`ARCreatAlertEvent`, `ARCreatSupportFile`, `ARDeleteActiveLink`, `ARDeleteSupportFile`, `ARGetActiveLink`, `ARGetListSupportFile`, `ARSetActiveLink`, `ARSetSupportFile`.

## ARGetTextForErrorMessage

**Description** Retrieves the message text for the specified error from the local catalog (in the local language). The length of the text is limited by `AR_MAX_MESSAGE_SIZE` (255 bytes).

**Privileges** All users.

**Synopsis**

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

```
int ARGetTextForErrorMessage(
    char* msgId)
```

**Input arguments**

**msgId**

The error number whose message text you want to retrieve. This number is provided in the `messageNum` member of `ARStatusStruct`.

**Return values****return**

The function return value is a pointer to the retrieved message text. You must free this memory after calling this function.

## ARGetVUI

**Description**

Retrieves information about the form view (VUI) with the indicated ID on the specified server.

<b>Privileges</b>	This operation can be performed by users with access permission for the form with which the VUI is associated.
<b>Synopsis</b>	<pre>#i ncl ude " ar. h" #i ncl ude " arerrno. h" #i ncl ude " arextern. h" #i ncl ude " arstruct. h"  int ARGetVUI (     ARControl Struct *control,     ARNameType schema,     ARInternalId vui Id,     Vui Name,     Locale locale,     *Vui Type,     *PropList dPropList,     **Text pText,     *Timestamp timestamp,     AccessNameType owner,     LastChanged lastChanged,     **ChangeDictionary changeDictionary,     *Status status)</pre>
<b>Input arguments</b>	<p><b>control</b>  The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The <code>user</code> and <code>server</code> fields are required.</p> <p><b>schema</b>  The name of the form containing the VUI to retrieve.</p> <p><b>vuid</b>  The internal ID of the VUI to retrieve.</p>
<b>Return values</b>	<p><b>vuiName</b>  The name of the VUI. Specify <code>NULL</code> for this parameter if you do not want to retrieve the VUI name.</p> <p><b>locale</b>  The locale of the VUI. Specify <code>NULL</code> for this parameter if you do not want to retrieve the locale.</p>

## vuiType

The type of VUI. Specify `NULL` for this parameter if you do not want to retrieve the VUI type.

- 0: No VUI type (`AR_VUI_TYPE_NONE`).
- 1: Windows type (`AR_VUI_TYPE_WINDOWS`) - fields in BMC Remedy User.
- 2: Web relative type (`AR_VUI_TYPE_WEB`) - fields in view can be adjusted.
- 3: Web absolute type (`AR_VUI_TYPE_WEB_ABS_POS`) - fields in view are fixed.

## dPropList

A list of zero or more display properties associated with the VUI. See “`ARCreateVUI`” on page 169 for a description of the possible values. The system returns 0 (`AR_DPROP_NONE`) if the VUI has no display properties. Specify `NULL` for this parameter if you do not want to retrieve this list.

## helpText

The help text associated with the VUI. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

## timestamp

A time stamp identifying the last change to the VUI. Specify `NULL` for this parameter if you do not want to retrieve this value.

## owner

The owning user for the VUI. Specify `NULL` for this parameter if you do not want to retrieve the owner.

## lastChanged

The user who made the last change to the VUI. Specify `NULL` for this parameter if you do not want to retrieve this value.

## changeDiary

The change diary associated with the VUI. Use `ARDecodeDiary` to parse the change diary into user, time stamp, and text components. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to retrieve the change diary (which is useful if you are calling this function to verify whether an instance of this object exists).

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateVUI, ARDecodeDiary, ARDeleteVUI, ARGetListVUI, ARGetMultipleVUIs, ARSetVUI. See FreeAR for: FreeARPropList, FreeARStatusList.

# ARImport

**Description** Imports the indicated structure definitions to the specified server. Use this function to copy structure definitions from one AR System server to another (see “ARExport” on page 206).

**Privileges** AR System administrator.

**Synopsis**

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

int ARImport(
    ARControlStruct *control,
    ARStructItemList *structItems,
    char *importBuf,
    importOption,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**structItems**

A list of zero or more structure items to import (identified by type and name). Specify `NULL` for this parameter to import all structures in the import buffer.

- 1: Form (includes all associated views) (`AR_STRUCT_ITEM_SCHEMA`).
- 5: Filter (`AR_STRUCT_ITEM_FILTER`).

- 6: Active link (AR\_STRUCT\_I TEM\_ACTIVE\_LINK).
- 8: Character menu (AR\_STRUCT\_I TEM\_CHAR\_MENU).
- 9: Escalation (AR\_STRUCT\_I TEM\_ESCALATION).
- 10: Distributed mapping (AR\_STRUCT\_I TEM\_DIST\_MAP).
- 12: Container (AR\_STRUCT\_I TEM\_CONTAINER).
- 14: VUI (AR\_STRUCT\_I TEM\_VUI).
- 16: Application (AR\_STRUCT\_I TEM\_APP)
- 30: Form data (AR\_STRUCT\_I TEM\_SCHEMA\_DATA)

---

**Note:** You must specify AR\_STRUCT\_I TEM\_SCHEMA to import a form. The three partial form types defined in ar.h (AR\_STRUCT\_I TEM\_SCHEMA\_DEFN, AR\_STRUCT\_I TEM\_SCHEMA\_VIEW, and AR\_STRUCT\_I TEM\_SCHEMA\_MAIL) do not contain complete form definitions and are used for caching purposes only.

---

## importBuf

A buffer that contains the definition text for the items specified for the structItems parameter. This buffer can contain other structure definitions, but the system imports the specified items only.

## importOption

A value indicating whether to replace objects being imported if they already exist, and how to replace them. There are two sets of import options. The first set is basic options, which are create and overwrite, and indicate whether to replace objects. The second set is advanced options, which enable further configuration of ARImport functionality by indicating how to replace objects. The advanced options are stored using a bit mask. This is because the advanced options should be set along with one of the basic options, and multiple advanced options can be turned on at the same time.

Basic options (the value for the overwrite option can be added to values for advanced options):

- 0: Creates new objects and generates an error if an object already exists. (AR\_IMPORT\_OPT\_CREATE).
- 1: Overwrites existing objects. (AR\_IMPORT\_OPT\_OVERWRITE).

Advanced options (add the values to select more than one option):

- 16: Checks for conflicting data types and reports an error.  
(AR\_IMPORT\_OPT\_HANDLE\_CONFLICT\_ERROR).
- 32: Checks for conflicting data types and overwrites objects on the server with that on file. (AR\_IMPORT\_OPT\_HANDLE\_CONFLICT\_OVERWRITE)
- 48: Checks for conflicting data types and does not change objects on the server if there is a conflict.  
(AR\_IMPORT\_OPT\_HANDLE\_CONFLICT\_NO\_ACTION).
- 64: Decides whether fields can be deleted.  
(AR\_IMPORT\_OPT\_NOT\_DELETE\_FIELD).

**Return values**

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

ARExport. See FreeAR for: FreeARStatusList, FreeARStructItemList.

# ARImportLicense

**Description** Imports a buffer, which is the contents of a license file, including checksums and encryption and an option, which tells whether to overwrite the existing license file or append to it.

When called, the server validates that the buffer is a valid license file and either appends the licenses in the file to the existing license file or replaces the existing license file with the new file.

**Privileges** AR System administrator.

**Synopsis**

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

int ARImportLicense(
    ARControlStruct *control,
    char *importBuf,
    unsigned int importOption,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**importBuf**

The image of the license file.

**importOption**

Indicates whether the license should be appended or overwritten.

- 0: Append the license file (AR\_LICENSE\_IMPORT\_APPEND).
- 1: Overwrite the license file (AR\_LICENSE\_IMPORT\_OVERWRITE).

**Return values**

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

See also    [ARExportLicense](#).

## ARInitialization

**Description**    Performs server- and network-specific initialization operations for each AR System session. All API programs that interact with the AR System must call this function before calling any other AR System API functions. Your program can call this function again to create additional sessions.

**Privileges**    All users.

**Synopsis**

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

int ARIinitialization(
    ARControlStruct          *control,
    ARStatusList               *status)
```

**Input arguments**

**control**

The control record for the operation. The `control` parameter must be the first input argument. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required. The `session` field, which you must supply in the control record for most other functions, will be returned by this function.

To use Unicode (UTF-8) as the client character set, set the `ARControl Struct.localInfo.charset` field as follows:

```
memset(&control, '\0', sizeof(control));
strcpy(control.localInfo.charset, "UTF-8");
...
if (ARIinitialization(&control, &status) >= AR_RETURN_ERROR)
```

Pass an empty string as `localInfo.charset` to get the normal API behavior. That is, the client character set will be the character set implied by the client's locale.

The other fields of `ARControl Struct` may safely be null-valued, and `ARIinitialization` will automatically fill them in.

**Return values** [status](#)

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARTermination. See FreeAR for: FreeARStatusList.

## ARJulianDateToDate

**Description** Converts a Julian date to a year, month, and day value. The Julian date is the number of days since noon, Universal Time, on January 1, 4713 BCE (on the Julian calendar). The changeover from the Julian calendar to the Gregorian calendar occurred in October, 1582. The Julian calendar is used for dates on or before October 4, 1582. The Gregorian calendar is used for dates on or after October 15, 1582.

**Privileges** All users.

### Synopsis

```
#i ncl ude "ar.h"
#i ncl ude "arerrno.h"
#i ncl ude "arextern.h"
#i ncl ude "arstruct.h"

int ARJulianDateToDate(
    ARControlStruct *control,
    int jd,
    ARDateStruct *date,
    ARStatusList *status)
```

**Input arguments**

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

### jd

The julian date value to convert.

**Return values**

### date

The resulting date structure holding the year, month, and day value.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARDateToJulianDate.

# ARLoadARQualifierStruct

**Description** Loads the specified qualification string (if valid for the form) into an ARQualifierStruct structure. This function simplifies the process of specifying qualifications in the required format.

**Synopsis**

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

int ARLoadARQualifierStruct(
    ARControlStruct *control,
    ARNameType schema,
    ARNameType displayTag,
    char *qualString,
    ARQualifierStruct *qualifier,
    ARStatusList *status)
```

**Input arguments**

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

### schema

The name of the form to which the qualification applies.

### displayTag

The name of the form view (VUI) to use for resolving field names. If the specified view does not exist or does not contain a field specified in the qualification string (or you specify `NULL` for this parameter), the system uses the field name in the default view.

**qualString**

A character string containing the qualification to load (following the syntax rules for entering qualifications in the BMC Remedy User query bar).

**Return values****qualifier**

An ARQual i fi erStruct structure that contains the specified qualification. Use FreeARQual i fi erStruct to recursively free the allocated memory associated with this structure as soon as you no longer need it.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

ARGetEntryStatistics, ARGetListEntry. See FreeAR for:  
FreeARQualifierStruct, FreeARStatusList.

## ARMergeEntry

**Description**

Merges an existing database entry into the indicated form on the specified server. You can merge entries into base forms only. To add entries to join forms, merge them into one of the underlying base forms.

**Privileges**

The system merges data based on the access privileges of the user you specify for the control parameter and the createMode setting for each field (see *ARCreateField* on page 139). User permissions are verified for each specified field. The system generates an error if the user does not have write permission for a field or a field does not exist.

**Synopsis**

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

int ARMergeEntry(
    ARControlStruct *control,
    ARNameType schema,
    ARFieldList *fieldList,
    ARMergeType mergeType,
    AREntryIdType entryId,
    ARStatusList *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**schema**

The name of the form to merge the entry into.

**fieldList**

A list of one or more field/value pairs (specified in any order) that identifies the data for the new entry. You must specify values for all required fields that do not have defined defaults. Values must be of the data type defined for the field or have a data type of 0 (`AR_DATA_TYPE_NULL`). `NULL` values can be specified for optional fields only, and assigning `NULL` overrides any defined field default. An error is generated if a field does not exist or the user specified by the `control` parameter does not have write permission for a field.

---

**Note:** You must specify a formatted diary string (such as that returned by `ARGetEntry`) for any diary fields you want to merge. In addition, unlike creating a new entry, you can specify values for the Entry ID, Create Date, Last Modified By, Modified Date, and Status History fields when merging an existing entry.

---

**mergeType**

A value indicating the action to take if `fieldList` includes the Entry ID field and the ID specified already exists in the target form. This parameter is ignored if you do not specify the Entry ID field or the ID specified does not conflict with existing entry IDs.

- 1: Generate an error (`AR.Merge_Entry_Dup_Error`).
- 2: Create a new entry with a new ID (`AR.Merge_Entry_Dup_New_ID`).
- 3: Delete the existing entry and create a new one in its place (`AR.Merge_Entry_Dup_Overwrite`).
- 4: Update the fields specified in `fieldList` in the existing entry (`AR.Merge_Entry_Dup_Merge`).

To omit some field validation steps, add the appropriate increments to the merge type.

1024: Allow NULL in required fields (not applicable for the Submitter, Status, or Short-Description core fields) (AR\_MERGE\_NO\_REQUIRED\_INCREMENT).

2048: Skip field pattern checking (including \$MENU\$) (AR\_MERGE\_NO\_PATTERNS\_INCREMENT).

For more information about field patterns, see the *Form and Application Objects* guide.

#### Return values

##### [entryId](#)

The ID of the merged entry. If you do not specify the Entry ID field in the file definition, the system generates and returns a new ID. If you do specify the Entry ID field and the ID specified is unique in the target form, the system returns that ID. If the ID specified is not unique and you specify AR\_MERGE\_ENTRY\_DUP\_NEW\_ID for the mergeType parameter, the system generates and returns a new ID.

##### [status](#)

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

See also [ARCreatEntry](#). See FreeAR for: [FreeARFieldValueList](#), [FreeARStatusList](#).

## ARRegisterForAlerts

**Description** Registers the specified user with the AR System server to receive alerts.

**Privileges** All users.

#### Synopsis

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

int ARRegisterForAlerts(
    ARControlStruct *control,
    int clientPort,
    unsigned int registrationFlags,
    ARStatusList *status)
```

<b>Input arguments</b>	<b>control</b>
	The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.
	<b>clientPort</b>
	The client port number.
	<b>registrationFlags</b>
	This value is reserved for future use and should be set to zero.
<b>Return values</b>	<b>status</b>
	A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the <i>Error Messages</i> guide.
<b>See also</b>	ARDeregisterForAlerts, ARCreateAlertEvent, ARGetAlertCount. See FreeAR for: FreeARStatusList.

## ARSetActiveLink

**Description** Updates the active link with the indicated name on the specified server. The changes are added to the server immediately and returned to users who request information about active links. Because active links operate on clients, individual clients do not receive the updated definition until they reconnect to the form (thus reloading the form from the server).

**Privileges** AR System administrator.

**Synopsis**

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

int ARSetActiveLink(
    ARControlStruct *control,
    ARNameType name,
    ARNameType newName,
    *order,
    *workflowConnect,
    *groupList,
    *executeMask,
```

int ARSetActiveLink(

    ARControlStruct

\*control,

    ARNameType

name,

    ARNameType

newName,

    \*order,

    ARWorkflowConnectStruct

\*workflowConnect,

    ARIinternalList

\*groupList,

    \*executeMask,

ARIInternalId	*controlField,
ARIInternalId	*focusField,
unsignedList	*enable,
ARQualifierFieldStruct	*query,
ARActiveLinkActionList	*actionList,
ARActiveLinkActionList	*elseList,
char	*helpText,
ARAccessNameType	owner,
char	*changeDictionary,
ARPropList	*objectPropList,
ARStatusList	*status)

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

### name

The name of the active link to update.

### newName

The new name for the active link. The names of all active links on a given server must be unique. Specify `NULL` for this parameter if you do not want to change the name of the active link.

### order

A value between 0 and 1000 (inclusive) that determines the active link execution order. When multiple active links are associated with a form, the value associated with each active link determines the order in which they are processed (from lowest to highest). Specify `NULL` for this parameter if you do not want to change the order.

### workflowConnect

The list of form names the active link is linked to. The active link must be associated with a single form or a list of forms that currently exists on the server. Specify `NULL` for this parameter if you do not want to change the list of forms. If the object is locked, you can append a form name to the list but you cannot remove a form name from the list.

## groupList

A list of zero or more groups who can access this active link. Users can execute an active link if they belong to a group that has access to it. Specifying an empty group list defines an active link accessible by users with administrator capability only. Specifying group ID 0 (Public) provides access to all users. The group list you specify replaces all existing group permissions. Specify `NULL` for this parameter if you do not want to change the group list.

## executeMask

A bitmask indicating the form operations that trigger the active link.

- Bit 0: Execute the active link when a user selects a button, toolbar button, or menu item specified by the `controlField` parameter (`AR_EXECUTE_ON_BUTTON`).
- Bit 1: Execute the active link when a user presses Return in the field specified by the `focusField` parameter (`AR_EXECUTE_ON_RETURN`).
- Bit 2: Execute the active link when a user submits an entry (*before* data is sent to the AR System server) (`AR_EXECUTE_ON_SUBMIT`).
- Bit 3: Execute the active link when a user modifies an individual entry (*before* data is sent to the AR System server) (`AR_EXECUTE_ON_MODIFY`).
- Bit 4: Execute the active link when a user displays an entry (*after* data is retrieved from the AR System server) (`AR_EXECUTE_ON_DISPLAY`).
- Bit 7: Execute the active link when a user selects an item from a character menu associated with the field specified by the `focusField` parameter or selects a row in the table field specified by the `focusField` parameter (`AR_EXECUTE_ON_MENU_CHOICE`).
- Bit 8: Execute the active link when a user moves out of a field (either manually or through workflow) (`AR_EXECUTE_ON_LOSE_FOCUS`).
- Bit 9: Execute the active link when a user sets default values (either manually or with preference settings) (`AR_EXECUTE_ON_SET_DEFAULT`).
- Bit 10: Execute the active link when a user retrieves one or more entries (*before* the query is sent to the AR System server) (`AR_EXECUTE_ON_QUERY`).
- Bit 11: Execute the active link when a user modifies an individual entry (*after* data is committed to the database) (`AR_EXECUTE_ON_AFTER MODIFY`).
- Bit 12: Execute the active link when a user submits an entry (*after* data is committed to the database) (`AR_EXECUTE_ON_AFTER_SUBMIT`).
- Bit 13: Execute the active link when a user moves into a field (either manually or through workflow) (`AR_EXECUTE_ON_GAIN_FOCUS`).
- Bit 14: Execute the active link when a user opens any form window (`AR_EXECUTE_ON_WINDOW_OPEN`).

- Bi t 15: Execute the active link when a user closes any form window (AR\_EXECUTE\_ON\_WINDOW\_CLOSE).
- Bi t 16: Execute the active link when a user moves off an entry (that is, when a record is removed from the server) (AR\_EXECUTE\_ON\_UNDISPLAY).
- Bi t 17: Execute the active link when a user performs a Copy to Submit operation (AR\_EXECUTE\_ON\_COPY\_SUBMIT).

Specify `NULL` for this parameter if you do not want to change the execute mask.

### controlField

The ID of the field that represents the button, toolbar button, or menu item associated with executing the active link. The `AR_EXECUTE_ON_BUTTON` condition (see “executeMask” on page 375) is ignored unless you specify this parameter. This parameter is ignored if you do not specify `AR_EXECUTE_ON_BUTTON`. Specify `NULL` for this parameter if you do not want to change the control field.

### focusField

The ID of the field associated with executing the active link by pressing Return or selecting a character menu item. The `AR_EXECUTE_ON_RETURN` or `AR_EXECUTE_ON_MENU_CHOICE` conditions (see “executeMask” on page 375) are ignored unless you specify this parameter (you must create another active link to specify a different field for each condition). This parameter is ignored if you do not specify either condition. Specify `NULL` for this parameter if you do not want to change the focus field.

### enable

A flag to enable or disable this active link. A value of 0 disables the active link, causing it to be invisible to the user and unavailable for use. A value of 1 enables the active link, causing it to be visible and available for use. Specify `NULL` for this parameter if you do not want to change this flag.

### query

A qualification that determines whether the active link is executed. Assign an operation value of 0 (`AR_COND_OP_NONE`) to execute the active link unconditionally. Specify `NULL` for this parameter if you do not want to change the query.

## actionList

The set of actions performed if the condition defined by the `query` parameter is satisfied. You can specify from 1 to 25 actions in this list (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to change the action list.

## elseList

The set of actions performed if the condition defined by the `query` parameter is not satisfied. You can specify from 0 to 25 actions in this list (limited by `AR_MAX_ACTIONS`). Specify a list with zero items to define no else actions. Specify `NULL` for this parameter if you do not want to change the else list.

## helpText

The help text associated with the active link. This text can be of any length. Specify `NULL` for this parameter if you do not want to change the help text.

## owner

The owning user for the active link. Specify `NULL` for this parameter if you do not want to change the owner.

## changeDiary

The additional change diary text to associate with the active link. This text can be of any length and is appended at the end of any existing text. Existing change diary text cannot be deleted or changed. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to add to the change diary.

## objPropList

If the `objPropList` parameter is set to `NULL`, no object properties are set. See “Server object property tags” on page 79 for object property tag names and data types.

## Return values

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## See also

`ARCreatAlertEvent`, `ARDeleteActiveLink`, `ARGetActiveLink`, `ARGetListActiveLink`, `ARGetMultipleActiveLinks`. See `FreeAR` for: `FreeARActiveLinkActionList`, `FreeARInternalIdList`, `FreeARPropList`, `FreeARQualifierStruct`, `FreeARStatusList`.

# ARSetApplicationState

**Description** Sets the application state (maintenance, test, or production) in the AR System Application State form.

**Privileges** All users.

**Synopsis**

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

```
int ARSetApplicationState(
    ARControlStruct *control,
    ARNameType applicationName,
    ARNameType stateName,
    *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**applicationName**

The name of the application.

**stateName**

The value for the `state` field, `currentStateName`, on the AR System Application State form. There is one entry per application.

**Return values**

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARGetApplicationState, ARGetListApplicationState.

# ARSetCharMenu

Description	Updates the character menu with the indicated name on the specified server. The changes are added to the server immediately and returned to users who request information about character menus. Because character menus operate on clients, individual clients do not receive the updated definition until they reconnect to the form (thus reloading the form from the server).
Privileges	AR System administrator.

---

**Note:** The `refreshCode` parameter has no effect on when the updated menu definition is retrieved. Rather, it controls how often the menu contents are retrieved by using the cached menu definition.

---

## Synopsis

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

int ARSetCharMenu(
    ARControlStruct          *control,
    ARNameType                name,
    ARNameType                newName,
    *refreshCode,
    *menuDefn,
    *helpText,
    owner,
    *changeDictionary,
    *objPropList,
    *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

### name

The name of the character menu to update.

## newName

The new name for the character menu. The names of all character menus on a given server must be unique. Specify `NULL` for this parameter if you do not want to change the name of the character menu.

## refreshCode

A value indicating when the menu is refreshed. This parameter enables you to balance menu consistency with performance.

- 1: Refresh only when the form is opened (`AR_MENU_REFRESH_CONNECT`).
- 2: Refresh every time the menu is opened (`AR_MENU_REFRESH_OPEN`).
- 3: Refresh the first time the menu is opened and every 15 minutes thereafter (`AR_MENU_REFRESH_INTERVAL`).

Specify `NULL` for this parameter if you do not want to change the refresh code.

## menuDefn

The definition of the character menu. Specify `NULL` for this parameter if you do not want to change the menu definition.

## helpText

The help text associated with the character menu. This text can be of any length. Specify `NULL` for this parameter if you do not want to change the help text.

## owner

The owning user for the character menu. Specify `NULL` for this parameter if you do not want to change the owner.

## changeDiary

The additional change diary text to associate with the character menu. This text can be of any length and is appended at the end of any existing text. Existing change diary text cannot be deleted or changed. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to add to the change diary.

## objPropList

If the `objPropList` parameter is set to `NULL`, no object properties are set. See “Server object property tags” on page 79 for object property tag names and data types.

**Return values** [status](#)

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateCharMenu, ARDeleteCharMenu, ARExpandCharMenu, ARGetCharMenu, ARGetListCharMenu. See FreeAR for: FreeARCharMenuStruct, FreeARPropList, FreeARStatusList.

## ARSetContainer

**Description** Updates the definition for the container with the indicated name on the specified server.

**Privileges** AR System administrator.

**Synopsis**

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

int ARSetContainer(
    ARControlStruct *control,
    ARNameType name,
    ARNameType newName,
    *groupList,
    *adminGrpList,
    *ownerObjList,
    *label,
    *description,
    *type,
    *references,
    removeFlag,
    *helpText,
    owner,
    *changeDir,
    *objPropList,
    *status)
```

**Input  
arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**name**

The name of the container to update.

**newName**

The new name for the container. The names of all containers on a given server must be unique. The system automatically updates all workflow references to the container if you rename it. Specify `NULL` for this parameter if you do not want to change the name of the container.

**groupList**

A list of zero or more groups who can access this container. Specifying an empty group list defines a container accessible by users with administrator capability only. Specifying group ID 0 (Public) provides access to all users. The permission value you assign for each group determines whether users in that group see the container in the container list.

- 1: Users see the container in the container list (`AR_PERMISSIONS_VISIBLE`).
- 2: Users do not see the container in the container list (`AR_PERMISSIONS_HIDDEN`).

The group list you specify replaces all existing group permissions. Specify `NULL` for this parameter if you do not want to change the group list.

**admingrpList**

A list of zero or more groups who can administer this container (and the referenced objects). If `ownerObj` is not `NULL`, this parameter is ignored and the Subadministrator group list of the owning form is used instead. Users must belong to both a specified group *and* the Subadministrator group to obtain these privileges. Specifying an empty administrator group list defines a container that can be administered by users with administrator capability only. Specifying group ID 0 (Public) provides subadministrator capability to all members of the Subadministrator group. The group list you specify replaces all existing group permissions. Specify `NULL` for this parameter if you do not want to change the administrator group list.

## ownerObjList

A list of schemas that own this container. Specify ARCONOWNER\_NONE for this parameter if you want the container to exist globally. Specify NULL for this parameter if you do not want to change the container's owning object. If the object is locked, you can add owners but you cannot delete them.

## label

The label for this container. It can be as many as 255 characters long. Specify NULL for this parameter if you do not want to change the label.

## description

The description for this container. It can be as many as 2000 characters long. Specify NULL for this parameter if you do not want to change the description.

## type

The type for this container, either ARCON\_GUI\_DE or ARCON\_APP. Specify NULL for this parameter if you do not want to change the type.

## references

Pointers to the objects (for example, forms or filters) referenced by this container. Specify NULL for this parameter if you do not want to change the references.

## removeFlag

A flag specifying how invalid object references are removed. If FALSE, references to nonexistent AR System objects will be removed with no error generated. If TRUE, an error will be generated. Specify NULL for this parameter if you do not want to change the flag.

## helpText

The help text associated with the container. This text can be of any length. Specify NULL for this parameter if you do not want to change the help text.

## owner

The owning user for the container. Specify NULL for this parameter if you do not want to change the owner.

## changeDiary

The additional change diary text to associate with the container. This text can be of any length and is appended at the end of any existing text. Existing change diary text cannot be deleted or changed. The server adds the user making the change and a time stamp when it saves the change diary. Specify NULL for this parameter if you do not want to add to the change diary.

## objPropList

If the objPropList parameter is set to NULL, no object properties are set. See “Server object property tags” on page 79 for object property tag names and data types.

### Return values

#### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

### See also

ARCreateContainer, ARDeleteContainer, ARGetContainer, ARGetListContainer, ARSetSchema. See FreeAR for: FreeARContainerInfoList, FreeARInternalIdList, FreeARPermissionList, FreeARPropList, FreeARReferenceList, FreeARStatusList.

## ARSetEntry

**Description** Updates the form entry with the indicated ID on the specified server.

**Privileges** The system updates data based on the access privileges of the user you specify for the control parameter. User permissions are verified for each specified field. The system generates an error if the user does not have write permission for a field or a field does not exist.

### Synopsis

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

int ARSetEntry(
    ARControlStruct *control,
    ARNameType schema,
    AREntryIdList *entryId,
    *fieldList,
    getTimestamp,
```

```
    unsi gned i nt          opti on,  
    ARStatusLi st           *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

### schema

The name of the form containing the entry to update.

### entryId

The ID of the entry to update.

---

**Note:** The system identifies entries in join forms by concatenating the entry IDs from the member forms. As a result, an entry ID can consist of one or more values of type `AREntryIdType` and, therefore, is represented by using the `AREntryIdList` structure.

---

### fieldList

A list of one or more field/value pairs (specified in any order) that identifies the new data for the entry. Values must be of the data type defined for the field or have a data type of 0 (`AR_DATA_TYPE_NULL`). NULL values can be specified for optional fields only. An error is generated if a field does not exist or the user specified by the `control` parameter does not have write permission for a field.

### getTime

A time stamp identifying when the entry was last retrieved. The system compares this value with the value in the `Modi fi ed Date` core field to determine whether the entry has been changed since the last retrieval. The system updates the entry if the value you specify is greater than `Modi fi ed Date`. If not, the system returns an error. You can either retrieve the entry again and determine whether to apply your changes or specify 0 for this parameter to bypass the time stamp comparison.

## option

A value indicating whether users can update fields specified in the join qualification (applicable for join forms only).

- 0: Users can update fields used in the join criteria (thereby causing the entry to no longer appear in the join form) (AR\_JOIN\_SETOPTION\_NONE).
- 1: Users cannot update fields used in the join criteria (AR\_JOIN\_SETOPTION\_REF).

## Return values

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateEntry, ARDeleteEntry, ARGetEntry, ARGetListEntry, ARMergeEntry. See FreeAR for: FreeAREntryIdList, FreeARFieldValueList, FreeARStatusList.

## ARSetEscalation

**Description** Updates the escalation with the indicated name on the specified server. The changes are added to the server immediately and returned to users who request information about escalations.

**Privileges** AR System administrator.

### Synopsis

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

int ARSetEscalation(
    ARControlStruct *control,
    ARNameType name,
    ARNameType newName,
    *escalationTm,
    *workflowConnect,
    *enable,
    *query,
    *actionList,
    *elseList,
```

```

char *helpText,
ARAccessNameType owner,
char *changeDictionary,
ARPropList *objectProperties,
ARStatusList *status)

```

## Input arguments

### **control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

### **name**

The name of the escalation to update.

### **newName**

The new name for the escalation. The names of all escalations on a given server must be unique. Specify `NULL` for this parameter if you do not want to change the name of the escalation.

### **escalationTm**

The time specification for evaluating the escalation condition. This parameter can take one of two forms: a time interval that defines how frequently the server checks the escalation condition (in seconds) or a bitmask that defines a particular day (by month or week) and time (hour and minute) for the server to check the condition. Specify `NULL` for this parameter if you do not want to change the escalation time.

### **workflowConnect**

The list of form names the escalation is linked to. The escalation must be associated with a single form or a list of forms that currently exists on the server. Specify `NULL` for this parameter if you do not want to specify the form list. If the object is locked, you can append a form name to the list but you cannot remove a form name from the list.

### **enable**

A flag to enable or disable this escalation. A value of 0 disables the escalation, causing its condition checks and associated actions not to be performed. A value of 1 enables the escalation, causing its conditions to be checked at the specified time interval. Specify `NULL` for this parameter if you do not want to change this flag.

## query

A query operation performed when the escalation is executed that determines the set of entries to which the escalation actions (defined by the `actionList` parameter) are applied. Assign an operation value of 0 (`AR_COND_OP_NONE`) to match all form entries. Specify `NULL` for this parameter if you do not want to change the query.

## actionList

The set of actions performed for each entry that matches the criteria defined by the `query` parameter. You can specify from 1 to 25 actions in this list (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to change the action list.

## elseList

The set of actions performed if no entries match the criteria defined by the `query` parameter. These actions are *not* performed for all non-matching entries. You can specify from 0 to 25 actions in this list (limited by `AR_MAX_ACTIONS`). Specify a list with zero items to define no else actions. Specify `NULL` for this parameter if you do not want to change the else list.

## helpText

The help text associated with the escalation. This text can be of any length. Specify `NULL` for this parameter if you do not want to change the help text.

## owner

The owning user for the escalation. Specify `NULL` for this parameter if you do not want to change the owner.

## changeDiary

The additional change diary text to associate with the escalation. This text can be of any length and is appended at the end of any existing text. Existing change diary text cannot be deleted or changed. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to add to the change diary.

## objPropList

If the `objPropList` parameter is set to `NULL`, no object properties are set. See “Server object property tags” on page 79 for object property tag names and data types.

**Return values** [status](#)

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateEscalation, ARDeleteEscalation, ARGetEscalation, ARGetListEscalation. See FreeAR for: FreeARFilterActionList, FreeARPropList, FreeARQualifierStruct, FreeARStatusList.

## ARSetField

**Description** Updates the definition for the form field with the indicated ID on the specified server.

**Privileges** AR System administrator.

**Synopsis**

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

int ARSetField(
    ARControlStruct *control,
    ARNameType schema,
    ARInternalId fieldId,
    ARNameType fName,
    *fieldMap,
    *option,
    *createMode,
    *fieldDOption,
    *defaultVal,
    *permissions,
    *limit,
    *instanceList,
    *helpText,
    owner,
    *changeDir,
    setFieldOptions
    *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**schema**

The name of the form containing the field to update.

**fieldId**

The internal ID of the field to update.

**fieldName**

The new name for the field. The names of all fields and VUIs associated with a given form must be unique. Specify `NULL` for this parameter if you do not want to change the name of the field.

**fieldMap**

The underlying form in which to create the field (applicable for join forms only). If you are creating a field in a base form, specify a field type of 1 (AR\_FI\_ELD\_REGULAR). Otherwise, specify a field type of 2 (AR\_FI\_ELD\_JOIN), and identify the member form (primary or secondary) and field ID for the new field. If the member form is also a join form, create fields in all nested join forms until you can map the field to an underlying base form. Specify `NULL` for this parameter if you do not want to set the field mapping.

**option**

A flag indicating whether users must enter a value in the field.

- 1: Required (data fields only) (AR\_FI\_ELD\_OPTI\_ON\_REQUIRED).
- 2: Optional (data fields only) (AR\_FI\_ELD\_OPTI\_ON\_OPTIONAL).
- 3: System (core data fields only) (AR\_FI\_ELD\_OPTI\_ON\_SYSTEM).
- 4: Display-only (nondata fields only) (AR\_FI\_ELD\_OPTI\_ON\_DISPLAY).

Specify `NULL` for this parameter if you do not want to change this flag.

## createMode

A flag indicating the permission status for the field when users submit entries. This parameter is ignored for display-only fields.

- 1: Any user (including guest users) can enter data in the field when submitting (AR\_FIELD\_OPEN\_AT\_CREATE).
- 2: Only users who have been granted permission can enter data in the field when submitting (AR\_FIELD\_PROTECTED\_AT\_CREATE).

Specify `NULL` for this parameter if you do not want to change this flag.

## fieldOption

A bitmask that indicates whether the field should be audited or copied when other fields are audited.

- Bit 0: Audit this field. (AR\_FIELD\_BITOPTION\_AUDIT)  
Bit 1: Copy this field when other fields in the form are audited. (AR\_FIELD\_BITOPTION\_COPY)  
Bit 2: Indicates this field is for Log Key 1. (AR\_FIELD\_BITOPTION\_LOG\_KEY1)  
Bit 3: Indicates this field is for Log Key 2. (AR\_FIELD\_BITOPTION\_LOG\_KEY2)  
Bit 2 and 3: Indicates this field is for Log Key 3. (AR\_FIELD\_BITOPTION\_LOG\_KEY3)

## defaultVal

The value to apply if a user submits an entry with no field value (applicable for data fields only). The default value can be as many as 255 bytes in length (limited by `AR_MAX_DEFAULT_SIZE`) and must be of the same data type as the field. Specify `0` (`AR_DEFAULT_VALUE_NONE`) for trim and control fields or to assign no default. Specify `NULL` for this parameter if you do not want to change the default value.

## permissions

A list of zero or more groups who can access this field. Specifying an empty permission list defines a field accessible by users with administrator capability only. Specifying group ID 0 (Public) provides access to all users. The permission value you assign for each group determines whether users in that group can modify the field.

- 1: Users can view but not modify the field (AR\_PERMISSIONS\_VIEW).
- 2: Users can view and modify the field (AR\_PERMISSIONS\_CHANGE).

The permission list you specify replaces all existing group privileges. Specify `NULL` for this parameter if you do not want to change the permissions.

#### **limit**

The value limits for the field and other properties specific to the field's type. See the `ARFieldListStruct` definition in the `ar.h` file to find the contained structure that applies to the type of field you are modifying. The limits and properties you assign must be of the same data type as the field. Specify `0` (`AR_FIELD_LIST_TYPE_NONE`) for trim and control fields or if you do not want to assign any limits or field type-specific properties. Specify `NULL` for this parameter if you do not want to change the field limits and properties.

#### **dInstanceList**

A list of zero or more display properties to associate with the field. See “[ARCreateField](#)” on page 143 for a description of the possible values. You can define both properties common to all form views (VUIs) and display properties specific to particular views. The system includes the field in each view you specify, regardless of whether you define any display properties for those views. If you do not specify a property for a particular view, the system uses the default value (if defined). Specify a list with zero items to remove *all* display properties for the field. Specify `NULL` for this parameter if you do not want to change this list.

#### **helpText**

The help text associated with the field. This text can be of any length. Specify `NULL` for this parameter if you do not want to change the help text.

#### **owner**

The owning user for the field. Specify `NULL` for this parameter if you do not want to change the owner.

#### **changeDiary**

The additional change diary text to associate with the field. This text can be of any length and is appended at the end of any existing text. Existing change diary text cannot be deleted or changed. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to add to the change diary.

## setFieldOptions

A bitmask for requesting special treatment of display instance items. You can set either (but not both) of the following bits:

- Bit 0: The server replaces its display instance items for this field that are present in the ARDisplayInstanceList\* argument to ARSetField(). The server does not modify or replace display instance items for this field that are not present in the list.  
(AR\_SETFIELD\_OPT\_PRESERVE\_UNLISTED\_DISPLAY\_INSTANCES)
- Bit 1: The server deletes the display instance items for this field present in the ARDisplayInstanceList\* arguments to ARSetField(). The server does not modify, replace, or delete any other display instance item.  
(AR\_SETFIELD\_OPT\_DELETE\_LISTED\_DISPLAY\_INSTANCES)

If both bits are set to 0, the server replaces all display instance items for this field with the values passed in dInstanceList.

### Return values

#### **status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

### See also

ARCreatField, ARDeleteField, ARGetField, ARGetListField, ARGetMultipleFields. See FreeAR for: FreeARDisplayInstanceList, FreeARFieldLimitList, FreeARPermissionList, FreeARStatusList, FreeARValueStruct.

## ARSetFilter

<b>Description</b>	Updates the filter with the indicated name on the specified server. The changes are added to the server immediately and returned to users who request information about filters.
<b>Privileges</b>	AR System administrator.

**Synopsis**

```
#i ncl ude "ar.h"
#i ncl ude "arerrno.h"
#i ncl ude "arextern.h"
#i ncl ude "arstruct.h"

int ARSetFilter(
    ARControlStruct *control,
    ARNameType name,
    ARNameType newName,
    *order,
    *workflowConnect,
    *opSet,
    *enable,
    *query,
    *actionList,
    *elementList,
    *helpText,
    owner,
    *changeDictionary,
    *objPropList,
    *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**name**

The name of the filter to update.

**newName**

The new name for the filter. The names of all filters on a given server must be unique. Specify `NULL` for this parameter if you do not want to change the name of the filter.

**order**

A value between 0 and 1000 (inclusive) that determines the filter execution order. When multiple filters are associated with a form, the value associated with each filter determines the order in which they are processed (from lowest to highest). Specify `NULL` for this parameter if you do not want to change the order.

## workflowConnect

The list of form names the filter is linked to. The filter must be associated with a single form or a list of forms that currently exists on the server. Specify `NULL` for this parameter if you do not want to specify the form list. If the object is locked, you can append a form name to the list but you cannot remove a form name from the list.

## opSet

A bitmask indicating the form operations that trigger the filter.

- Bit 0: Execute the filter on get operations (`AR_OPERATION_GET`).
- Bit 1: Execute the filter on set operations (`AR_OPERATION_SET`).
- Bit 2: Execute the filter on create operations (`AR_OPERATION_CREATE`).
- Bit 3: Execute the filter on delete operations (`AR_OPERATION_DELETE`).
- Bit 4: Execute the filter on merge operations (`AR_OPERATION_MERGE`).

Specify `NULL` for this parameter if you do not want to change the operation set.

## enable

A flag to enable or disable this filter. A value of 0 disables the filter, causing its condition checks and associated actions not to be performed. A value of 1 enables the filter, causing its conditions to be checked for each form operation specified by the `opSet` parameter. Specify `NULL` for this parameter if you do not want to change this flag.

## query

A qualification that determines whether the filter is executed. Assign an operation value of 0 (`AR_COND_OP_NONE`) to execute the filter unconditionally. Specify `NULL` for this parameter if you do not want to change the query.

## actionList

The set of actions performed if the condition defined by the `query` parameter is satisfied. You can specify from 1 to 25 actions in this list (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to change the action list.

### elseList

The set of actions performed if the condition defined by the `query` parameter is not satisfied. You can specify from 0 to 25 actions in this list (limited by `AR_MAX_ACTIONS`). Specify a list with zero items to define no else actions. Specify `NULL` for this parameter if you do not want to change the else list.

### helpText

The help text associated with the filter. This text can be of any length. Specify `NULL` for this parameter if you do not want to change the help text.

### owner

The owning user for the filter. Specify `NULL` for this parameter if you do not want to change the owner.

### changeDiary

The additional change diary text to associate with the filter. This text can be of any length and is appended at the end of any existing text. Existing change diary text cannot be deleted or changed. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to add to the change diary.

### objPropList

If the `objPropList` parameter is set to `NULL`, no properties will be set. See “Server object property tags” on page 79 for object property tag names and data types.

## Return values

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## See also

`ARCreatFilter`, `ARDeleteFilter`, `ARGetFilter`, `ARGetListFilter`. See `FreeAR` for: `FreeARFilterActionList`, `FreeARPropList`, `FreeARQualifierStruct`, `FreeARStatusList`.

## ARSetImpersonatedUser

**Description** Enables plug-ins, the mid tier, and other programs (such as the Email Engine) to run as an administrator but to perform operations as a specific user (with the user's permissions and licensing in effect). For more information, see “Impersonating a user” on page 446.

**Privileges** AR System administrator.

**Synopsis**

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

int ARSetImpersonatedUser(
    ARControlStruct *control,
    ARAccessNameType impersonatedUser,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**impersonatedUser**

The name of the user that the API is impersonating. It is `NULL` if you want to stop impersonating.

**Return values**

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## ARSetLogging

**Description** Activates and deactivates client-side logging of server activity.

**Privileges** Users with permission to perform client-side logging. (See the *Optimizing and Troubleshooting* guide.)

**Synopsis**

```
#i ncl ude "ar.h"
#i ncl ude "arerrno.h"
#i ncl ude "arextern.h"

int ARSetLogging(
    ARControlStruct *control,
    long logTypeMask,
    long whereToWriteMask,
    FILE *filePtr,
    ARStatusList *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**logTypeMask**

A bitmask indicating the type of logging to set. This is for SQL, filter, API, and plug-in server logging. The possible values are:

- Bit 1: Database or SQL logging (`AR_DEBUG_SERVER_SQL`).
- Bit 2: Filter logging (`AR_DEBUG_SERVER_FILTER`).
- Bit 5: API logging (`AR_DEBUG_SERVER_API`).
- Bit 18: Plug-in logging (`AR_DEBUG_SERVER_PLUGIN`).

**whereToWriteMask**

A value indicating where to return log information. The possible values are

- 1: Logs to a file (`AR_WRI_TE_TO_FILE`).
- 2: Logs as part of an `ARStatusList` (`AR_WRI_TE_TO_STATUS_LIST`).

**filePtr**

A file handle to the log file. It is assumed the file has already been opened. A `NULL` value indicates that no writing be done to the file.

**Return values****status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

If the whereToWriteMask input parameter is set to AR\_WRI TE\_TO\_STATUS\_LI ST, the message number is of type messageNum (number of the status structure), and everything that is logged is of type AR\_NOTE\_LOG\_I NFO.

## ARSetSchema

**Description** Updates the definition for the form with the indicated name on the specified server. If the schema is locked, only the indexList and the defaultVui can be set.

**Privileges** AR System administrator.

**Synopsis**

```
#i ncl ude "ar.h"
#i ncl ude "arerrno.h"
#i ncl ude "arextern.h"
#i ncl ude "arstruct.h"

int ARSetSchema(
    ARControl Struct *control,
    ARNameType name,
    ARNameType newName,
    *schema,
    *schemasmList,
    *groupList,
    *adminGroupList,
    *getListFields,
    *sortList,
    *indexList,
    *archivedInfo,
    *auditInfo,
    defaultVui,
    *helpText,
    owner,
    *changeDictionary,
    *objPropList,
    setOption
    *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**name**

The name of the form to update.

**newName**

The new name for the form. The names of all forms on a given server must be unique. The system automatically updates all workflow references to the form if you rename it. Specify `NULL` for this parameter if you do not want to change the name of the form.

**schema**

The form type (base form or join form). See “`ARCreateSchema`” on page 164 for a description of the possible values. You cannot change a join form to a base form or vice versa. You can, however, modify the join criteria for a join form. Specify `NULL` for this parameter if you do not want to change the form information.

**schemaInheritanceList**

This is reserved for future use. Specify `NULL`.

**groupList**

A list of zero or more groups who can access this form. Specifying an empty group list defines a form accessible by users with administrator capability only. Specifying group ID 0 (Public) provides access to all users. The permission value you assign for each group determines whether users in that group see the form in the form list.

- 1: Users see the form in the form list (`AR_PERMISSIONS_VISIBLE`).
- 2: Users do not see the form in the form list (`AR_PERMISSIONS_HIDDEN`).

The group list you specify replaces all existing group permissions. Specify `NULL` for this parameter if you do not want to change the group list.

## admingrpList

A list of zero or more groups who can administer this form (and the associated filters, escalations, and active links). Users must belong to both a specified group *and* the Subadministrator group to obtain these privileges. Specifying an empty administrator group list defines a form that can be administered by users with administrator capability only. Specifying group ID 0 (Public) provides subadministrator capability to all members of the Subadministrator group. The group list you specify replaces all existing group permissions. Specify `NULL` for this parameter if you do not want to change the administrator group list.

## getListFields

A list of zero or more fields that identifies the default query list data for retrieving form entries. The combined length of all specified fields, including separator characters, can be as many as 128 bytes (limited by `AR_MAX_SDESC_SIZE`). Specify zero fields to assign the Short-Description core field as the default query list data. Specifying a `getListFields` argument when calling the `ARGetListEntry` function overrides the default query list data. Specify `NULL` for this parameter if you do not want to change the default query list fields.

## sortList

A list of zero or more fields that identifies the default sort order for retrieving form entries. Specifying a `sortList` argument when calling the `ARGetListEntry` function overrides the default sort order. Specify `NULL` for this parameter if you do not want to change the default sort fields.

## indexList

The set of zero or more indexes to create for the form. You can specify from 1 to 16 fields for each index (limited by `AR_MAX_INDEX_FIELDS`). Diary fields and character fields larger than 255 bytes cannot be indexed. Specify `NULL` for this parameter if you do not want to change the index list.

## archiveInfo

If a form is to be archived, this is the archive information for the form. Specify `NULL` for this parameter if you do not want to create or set the archive information. These are the values for archive type:

- 0: Deletes the archive settings for the selected form. The selected form will not be archived (`AR_ARCHIVE_NONE`).
- 1: Entries on the selected form are copied to the archive form (`AR_ARCHIVE_FORM`).
- 2: Entries on the selected form are deleted from the source form (`AR_ARCHIVE_DELETE`).
- 4: Entries on the selected form are copied to an XML format file (`AR_ARCHIVE_FILE_XML`).
- 8: Entries on the selected form are copied to an ARX format file (`AR_ARCHIVE_FILE_ARX`).
- 32: Attachment fields are not copied to the archive form. (`AR_ARCHIVE_NO_ATTACHMENTS`).
- 64: Diary fields are not copied to the archive form. (`AR_ARCHIVE_NO_DIARY`).

In addition to the archive type, `archiveInfo` also stores the form name (if archive type is `AR_ARCHIVE_FORM`), time to trigger the archive, and the archive qualification criteria. For an archive form, only the name of the source form is maintained, and none of this archive information can be set.

## auditInfo

If a form is to be audited, this is the audit information for the form. Specify `NULL` for this parameter if you do not want to create or set the audit information. These are the values for audit type:

- 0: The selected form is not to be audited. (`AR_AUDIT_NONE`).
- 1: Audit fields and copy fields on the selected form are copied to the audit shadow form (`AR_AUDIT_COPY`).
- 2: Audit fields and copy fields on the selected form are copied to the audit log form (`AR_AUDIT_LOG`).

In addition to the audit type, `auditInfo` also stores the form name (if audit type is `AR_AUDIT_COPY` or `AR_AUDIT_LOG`) and the audit qualification criteria.

## defaultVui

The label of the default view. Specify `NULL` for this parameter if you do not want to set this value.

### helpText

The help text associated with the form. This text can be of any length. Specify `NULL` for this parameter if you do not want to change the help text.

### owner

The owning user for the form. Specify `NULL` for this parameter if you do not want to change the owner.

### changeDiary

The additional change diary text associated with the form. This text can be of any length and is appended at the end of any existing text. Existing change diary text cannot be deleted or changed. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to add to the change diary.

### objPropList

If the `objPropList` parameter is set to `NULL`, no object properties will be set. See “Server object property tags” on page 79 for object property tag names and data types.

### setOption

Reserved for future use.

## Return values

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## See also

`ARCreatField`, `ARCreatSchema`, `ARDeleteSchema`, `ARGetListEntry`, `ARGetListAlertUser`, `ARGetSchema`. See `FreeAR` for: `FreeARCompoundSchema`, `FreeAREntryListFieldList`, `FreeARIIndexList`, `FreeARInternalIdList`, `FreeARPermissionList`, `FreeARPropList`, `FreeARSortList`, `FreeARStatusList`.

## ARSetServerInfo

**Description**     Updates the indicated configuration information for the specified server.

**Privileges**    AR System administrator.

**Synopsis**

```
#i ncl ude "ar.h"
#i ncl ude "arerrno.h"
#i ncl ude "arextern.h"

int ARSetServerInfo(
    ARControl Struct *control,
    ARServerInfoList *serverInfo,
    ARStatusList *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**serverInfo**

The information to update on the server (see the following Server options section). The system returns error messages for server options not updated due to error.

**Return values****status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**Server options**

Some server options can be set and retrieved. Others can be retrieved only and still others can be set only.

- For a complete list of options that can be set with the `ARSetServerInfo` call and retrieved with the `ARGetServerInfo` call, see the server options descriptions for “`ARGetServerInfo`” on page 322.
- For a complete list of options that can be retrieved only, see the server option descriptions for “`ARGetServerInfo`” on page 322.
- You can set the value of the `AR_SERVER_INFO_DB_PASSWORD` option, but you cannot retrieve its value.

`AR_SERVER_INFO_DB_PASSWORD`:

The database password associated with the `ARSysTem` (applicable to Oracle, SQL Server, and Sybase databases only). The encrypted password value is stored in the configuration file.

**See also** ar.conf file, ARGetServerInfo. See FreeAR for: FreeARServerInfoList, FreeARStatusList.

## ARSetServerPort

**Description** Specifies the port that your program will use to communicate with the AR System server and whether to use a private server.

**Privileges** All users.

**Synopsis**

```
#i ncl ude "ar. h"
#i ncl ude "arerrno. h"
#i ncl ude "arextern. h"

int ARSetServerPort(
    ARControl Struct          *control ,
    ARNameType                server,
    int                        port,
    int                        rpcProgramNum,
    ARStatusLi st             *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**server**

The name of the server to update.

**port**

The port number that your program will use to communicate with the AR System server. If you do not specify this parameter or provide 0 for the port number, your program will use the port number supplied by the portmapper. This parameter is overridden by the ARTCPPORT environment variable.

**rpcProgNum**

The RPC program number of the server. Specify 390600 to use the admin server, a number from 390621 to 390634 or 390636 to 390669 or 390680-390694 to use a private server, or 0 (default) to use the fast or list server queue. This parameter is overridden by the ARRPC environment variable.

To retrieve server statistics information with the ARGetServerStatistics call, you can also set the RPC Program number to 390619, 390620, or 390635.

## Return values

### **status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## ARSetSessionConfiguration

**Description** Sets an API session variable.

**Privileges** This operation can be performed by all users.

### **Synopsis**

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

```
int ARSetSessionConfiguration(
    ARControlStruct *control,
    unsigned int variablename,
    ARValueStruct *variablevalue,
    ARStatusList *status)
```

### **Input arguments**

#### **control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and session fields are required.

## variableId

Identification of the variable type:

- 1: Maximum size for a single response (AR\_SESS\_CHUNK\_RESPONSE\_SIZE).
- 2: Timeout for normal operations (AR\_SESS\_TIMEOUT\_NORMAL).
- 3: Timeout for long operations (AR\_SESS\_TIMEOUT\_LONG).
- 4: Timeout for extra long operations (AR\_SESS\_TIMEOUT\_XLONG).
- 5: Socket number to lock to (AR\_SESS\_LOCK\_TO\_SOCKET\_NUMBER).
- 6: Flag with Boolean value indicating if the session is pooled (AR\_SESS\_POOLED).
- 7: API program client type (AR\_SESS\_CLIENT\_TYPE).
- 8: Type of VUI view (AR\_SESS\_VUI\_TYPE).
- 9: Flag with Boolean value indicating if the user would like to override the session from the previous IP (AR\_SESS\_OVERRIDE\_PREV\_IP).

## variableValue

Configuration variable value:

`AR_SESS_CHUNK_RESPONSE_SIZE (integer)`

An integer value for the maximum data packet size returned from the server to the client in one transmission.

`AR_SESS_TIMEOUT_NORMAL (integer)`

An integer value for the client timeout value used in fast server operations.

`AR_SESS_TIMEOUT_LONG (integer)`

An integer value for the client timeout value used in list server operations.

`AR_SESS_TIMEOUT_XLONG (integer)`

An integer value for the client timeout value used in definition updating and in import/export operations.

`AR_SESS_LOCK_TO_SOCKET_NUMBER (integer)`

An integer value for the socket number that the client will lock to for all server interaction.

**AR\_SESS\_POOLED (integer)**

An integer value for the flag indicating if the session is pooled. Valid values for this option are 0 (No) and 1 (Yes).

**AR\_CLIENT\_TYPE\_\* (integer)**

An integer value for the client type. For more information, see AR\_CLIENT\_TYPE\_\* in the ar.h file.

**Range of values for user-defined, client-type variables:**

AR\_CLIENT\_TYPE\_END\_OF\_RESERVED\_RANGE: >5000.

**AR\_SESS\_VUI\_TYPE (integer)**

An integer value for the type of VUI. For more information, see AR\_VUI\_TYPE\_\* in the ar.h file.

**AR\_SESS\_OVERRIDE\_PREV\_IP (integer)**

An integer value for the flag indicating if the session can be overridden. Valid values for this option are 0 (No) and 1 (Yes).

**Return values****status**

A list of zero or more notes, warnings, or errors generated from this function call. Refer to “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARGetSessionConfiguration.

## ARSetSupportFile

**Description** Sets a support file in the AR System.

**Privileges** AR System administrator.

**Synopsis**

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

```
int ARSetSupportFile (
```

ARControlStruct	*control,
unsigned int	fileType,
ARNametype	name,
ARInternalId	id2,

```
ARIInternalId  
FILE  
ARStatusList  
fiel d,  
*filePtr,  
*status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

### fileType

The numerical value for the type of file, and the type of object the file is related to. Specify 1 (AR\_SUPPORT\_FILE\_EXTERNAL\_REPORT) for an external report file associated with an active link.

### name

The name of the object the file is associated with, usually a form.

### id2

The associated identifier if the object identifier is not enough.

### fileId

The unique identifier of a file within its object.

### filePtr

A pointer to the support file to be set in the system. If you are using Windows, you must open the file in binary mode.

## Return values

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## See also

[ARCreateAlertEvent](#), [ARCreateSupportFile](#), [ARDeleteActiveLink](#), [ARDeleteSupportFile](#), [ARGetActiveLink](#), [ARGetListSupportFile](#), [ARGetSupportFile](#), [ARSetActiveLink](#).

# ARSetVUI

**Description** Updates the form view (VUI) with the indicated ID on the specified server.

**Privileges** AR System administrator.

**Synopsis**

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

int ARSetVUI (
    ARControlStruct *control,
    ARNameType schema,
    ARInternalId vuiId,
    ARNameType vuiName,
    ARLocalEntityType localE,
    unsigned int *vuiType,
    ARPropList *dPropList,
    char *helpText,
    ARAccessNameType owner,
    char *changeDictionary,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user` and `server` fields are required.

**schema**

The name of the form containing the VUI to update.

**vuid**

The internal ID of the VUI to update.

**vuiName**

The new name for the VUI. The names of all VUIs and fields associated with a given form must be unique. Specify `NULL` for this parameter if you do not want to change the name of the VUI.

## locale

The locale of the VUI. Specify `NULL` for this parameter if you do not want to change the name of the VUI.

## vuiType

The type of VUI. Specify `NULL` for this parameter if you do not want to retrieve the VUI type.

- 0: No VUI type (`AR_VUI_TYPE_NONE`).
- 1: Windows type (`AR_VUI_TYPE_WINDOWS`) - fields in BMC Remedy User.
- 2: Web relative type (`AR_VUI_TYPE_WEB`) - fields in view can be adjusted.
- 3: Web absolute type (`AR_VUI_TYPE_WEB_ABS_POS`) - fields in view are fixed.

## dPropList

A list of zero or more display properties to associate with the VUI. See “`ARCreatVUI`” on page 169 for a description of the possible values. Specify 0 (`AR_DPROP_NONE`) to assign no display properties. Specify `NULL` for this parameter if you do not want to change this list.

## helpText

The help text associated with the VUI. This text can be of any length. Specify `NULL` for this parameter if you do not want to change the help text.

## owner

The owning user for the VUI. Specify `NULL` for this parameter if you do not want to change the owner.

## changeDiary

The additional change diary text to associate with the VUI. This text can be of any length and is appended at the end of any existing text. Existing change diary text cannot be deleted or changed. The server adds the user making the change and a time stamp when it saves the change diary. Specify `NULL` for this parameter if you do not want to associate change diary text with this object.

## Return values

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateVUI, ARDeleteVUI, ARGetVUI, ARGetListVUI. See FreeAR for: FreeARPropList, FreeARStatusList.

## ARSignal

**Description** Causes the server to reload information.

**Privileges** All users.

**Synopsis**

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

int ARSignal (
    ARControlStruct *control,
    ARSignalList *signalList,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**signalList**

The list of signals. The options for this parameter are:

AR\_SIGNAL\_CONFIG\_CHANGED

Causes the server to reload information from its configuration file (ar.conf on UNIX, ar.cfg on Windows).

AR\_SIGNAL\_GROUP\_CACHE\_CHANGED

Causes the server to reload group and data dictionary information from the database.

AR\_SIGNAL\_LICENSE\_CHANGED

Causes the server to reload license information.

**Return values**

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## ARTermination

<b>Description</b>	Performs environment-specific cleanup routines and disconnects from the specified AR System session. All API programs that interact with the AR System should call this function upon completing work in a given session. Calling this function is especially important in environments that use floating licenses. If you do not disconnect from the server, your license token is unavailable for other users for the defined time-out interval.
<b>Privileges</b>	All users.
<b>Synopsis</b>	<pre>#include "ar.h" #include "arerrno.h" #include "arextern.h" #include "arstruct.h"  int ARTermination(     ARControlStruct          *control,     ARStatusList              *status)</pre>
<b>Input arguments</b>	<p><b>control</b></p> <p>The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The <code>user</code> and <code>server</code> fields are required.</p>
<b>Return values</b>	<p><b>status</b></p> <p>A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the <i>Error Messages</i> guide.</p>
<b>See also</b>	ARInitialization. See FreeAR for: FreeARStatusList.

## ARValidateFormCache

<b>Description</b>	Retrieves key information from a cached definition (such as form name, form change time, active link change time, active link guide change time, and menu change time) and returns information about any changes to the definition given this information. This function call also returns information about updates to the current user performing a test. This information causes the cache to reload to insure accurate definitions.
--------------------	---

This call is intended to allow BMC Remedy User to gather all the data that is needed to validate its local cache for a form in a single call.

**Privileges** All users.

**Synopsis**

```
#i ncl ude "ar.h"
#i ncl ude "arerrno.h"
#i ncl ude "arextern.h"
#i ncl ude "arstruct.h"

int ARVal(int dateFormCache(
    ARControlStruct *control,
    ARNameType form,
    ARTimestamp mostRecentActLink,
    ARTimestamp mostRecentMenu,
    ARTimestamp mostRecentGUI,
    *formLastModifiled,
    *numActLinksOnForm,
    *numActLinksSince,
    *menuSinceList,
    *groupsLastChanged,
    *userLastChanged,
    *guiSinceList,
    *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**form**

The name of the form to validate changes in.

**mostRecentActLink**

A time stamp you provide for the most recent modification time that you know about of an active link for the form.

**mostRecentMenu**

A user supplied time stamp that provides the most recent modification time for the form menu.

## mostRecentGuide

A user supplied time stamp that provides the most recent modification time for an active link guide for the form.

### Return values

## formLastModified

A time stamp that identifies the last modification time of the form. Specify `NULL` for this parameter if you do not want to retrieve this value.

## numActLinkOnForm

An integer that indicates the number of active links associated with the form that are accessible by the current user. Specify `NULL` for this parameter if you do not want to retrieve this value.

## numActLinkSince

An integer that indicates the number of active links associated with the form that are accessible to the current user and that have been modified since the time indicated by `mostRecentActLink`. Specify `NULL` for this parameter if you do not want to retrieve this value.

## menuSinceList

A list of menus changed since the time specified in `mostRecentMenu`. Specify `NULL` for this parameter if you do not want to retrieve this value.

## groupsLastChanged

A time stamp identifying the last change made to any Group on the server. Specify `NULL` for this parameter if you do not want to retrieve this value.

## userLastChanged

A time stamp identifying the last change made to the current user. Specify `NULL` for this parameter if you do not want to retrieve this value.

## guideSinceList

A list of active link guides changed since the time specified in the `mostRecentGuide` argument. Specify `NULL` for this parameter if you do not want to retrieve this value.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

See also ARGetListActiveLink, ARGetListCharMenu, ARGetListContainer, ARGetSchema, ARGetServerInfo. See FreeAR for: FreeARStatusList.

## ARValidateLicense

**Description** Confirms whether the current server holds a valid license of the specified type.

**Privileges** All users.

**Synopsis**

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

int ARValidateLicense(
    ARControlStruct *control,
    ARLicenseNameType licenseType,
    ARLicenseValiadStruct *licenseValid,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**licenseTypeList**

A list of the type of license that this call validates, such as license key, license type, and expiration date. The system does not support a `NULL` value in this option.

**Return values**

**licenseValid**

License information, such as validity of license, license key, license type, and expiration date.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateLicense, ARDeleteLicense, ARGetListLicense, ARValidateMultipleLicenses. See FreeAR for: FreeARLicenseValidStruct, FreeARStatusList.

## ARValidateMultipleLicenses

**Description** Checks whether the current server holds a valid license for several specified license types. This function performs the same action as ARVal i dateLi cense but it is easier to use and more efficient than validating licenses one by one.

**Privileges** All users.

**Synopsis**

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

int ARVal i dateMul ti pl eLi censes(
    ARControl Struct *control,
    ARLi censeNameLi st *Li censeTypeLi st,
    ARLi censeVal i dLi st *Li censeVal i dLi st,
    ARStatusLi st *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**licenseTypeList**

A list of the type of license that this call validates, such as license key, license type, and expiration date. The system does not support a NULL value in this option.

**Return values**

**licenseValidList**

A list of license information, such as validity of license, license key, license type, and expiration date.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARCreateLicense, ARDeleteLicense, ARGetListLicense, ARValidateLicense, ARValidateMultipleLicenses. See FreeAR for: FreeARLicenseNameList, FreeARLicenseValidList, FreeARStatusList.

## ARVerifyUser

**Description** Checks the cache on the specified server to determine whether the specified user is registered with the current server.

**Privileges** All users.

**Synopsis**

```
#i ncl ude "ar. h"
#i ncl ude "arerrno. h"
#i ncl ude "arextern. h"
#i ncl ude "arstruct. h"

int ARVerifyUser(
    ARControlStruct *control,
    ARBool user,
    ARBool admin,
    ARBool subAdmin,
    ARBool customFlag,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user and server fields are required.

**Return values**

[\*\*adminFlag\*\*](#)

A flag indicating whether the specified user is a member of the Administrator group. The system returns 1 (TRUE) if the user is a member of the Administrator group. The system returns 0 (FALSE) if the user is not a member of the Administrator group, is invalid, or is unknown. Specify NULL for this parameter if you do not want to retrieve this flag.

[\*\*subAdminFlag\*\*](#)

A flag indicating whether the specified user is a member of the Subadministrator group. The system always returns 1 (TRUE) if the user is a member of the Subadministrator group. The system returns 0 (FALSE) if the user is not a member of the Subadministrator group, is invalid, or is unknown. Specify NULL for this parameter if you do not want to retrieve this flag.

## customFlag

A flag indicating whether the specified user is a member of the Customize group. The system always returns 1 (TRUE) if the user is a member of the Customize group. The system returns 0 (FALSE) if the user is not a member of the Customize group, is invalid, or is unknown. Specify `NULL` for this parameter if you do not want to retrieve this flag.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** [FreeARStatusList](#).

# FreeAR

**Description** Recursively free all allocated memory associated with a particular AR System data structure. All structure components must be in allocated memory to use these functions.

**Privileges** All users.

### Synopsis

```
#i ncl ude "ar.h"
#i ncl ude "arfree.h"

voi d FreeARAccessNameLi st(
    ARAccessNameLi st          *val ue,
    ARBool ean                 freeStruct)

voi d FreeARActi veLi nkActi onLi st(
    ARActi veLi nkActi onLi st *val ue,
    ARBool ean                 freeStruct)

voi d FreeARActi veLi nkActi onLi stLi st(
    ARActi veLi nkActi onLi st *val ue,
    ARBool ean                 freeStruct)

voi d FreeARActi veLi nkActi onStruct(
    ARActi veLi nkActi onStruct *val ue,
    ARBool ean                 freeStruct)
```

```
voi d FreeARArchivel nfoStruct(  
    ARArchivel nfoStruct  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
voi d FreeARArith0pAssi gnStruct(  
    ARArith0pAssi gnStruct  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
voi d FreeARArith0pStruct(  
    ARArith0pStruct  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
voi d FreeARAssi gnFil dStruct(  
    ARAssi gnFil dStruct  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
voi d FreeARAssi gnSQLStruct(  
    ARAssi gnSQLStruct  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
voi d FreeARAssi gnStruct(  
    ARAssi gnStruct  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
voi d FreeARAttachStruct(  
    ARAttachStruct  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
voi d FreeARAutomati onStruct(  
    ARAutomati onStruct  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
voi d FreeARBool eanLi st(  
    ARBool eanLi st  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
voi d FreeARBool eanLi stLi st(  
    ARBool eanLi stLi st  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
voi d FreeARBool eanMatri x(  
    ARBool eanMatri x  
    ARBool ean  
    *val ue,  
    freeStruct)
```

```
void FreeARBufStruct(  
    ARBufStruct *value,  
    ARBool ean  
)  
  
void FreeARByteList(  
    ARByteList *value,  
    ARBool ean  
)  
  
void FreeARCalIGuiDeStruct(  
    ARCalIGuiDeStruct *value,  
    ARBool ean  
)  
  
void FreeARCharMenuItemStruct(  
    ARCharMenuItemStruct *value,  
    ARBool ean  
)  
  
void FreeARCharMenuList(  
    ARCharMenuList *value,  
    ARBool ean  
)  
  
void FreeARCharMenuSSStruct(  
    ARCharMenuSSStruct *value,  
    ARBool ean  
)  
  
void FreeARCharMenuStruct(  
    ARCharMenuStruct *value,  
    ARBool ean  
)  
  
void FreeARCOMMethodList(  
    ARCOMMethodList *value,  
    ARBool ean  
)  
  
void FreeARCOMMethodParmList(  
    ARCOMMethodParmList *value,  
    ARBool ean  
)  
  
void FreeARCOMMethodParmStruct(  
    ARCOMMethodParmStruct *value,  
    ARBool ean  
)  
  
void FreeARCOMMethodStruct(  
    ARCOMMethodStruct *value,  
    ARBool ean  
)
```

```
voi d FreeARCompoundSchema(  
    ARCompoundSchema          *val ue,  
    ARBool ean                freeStruct)  
  
voi d FreeARCompoundSchemaLi st(  
    ARCompoundSchemaLi st     *val ue,  
    ARBool ean                freeStruct)  
  
voi d FreeARCOMVal ueStruct(  
    ARCOMVal ueStruct        *val ue,  
    ARBool ean                freeStruct)  
  
voi d FreeARContai nerI nfoLi st(  
    ARContai nerI nfoLi st   *val ue,  
    ARBool ean                freeStruct)  
  
voi d FreeARContai nerOwnerObj Li st(  
    ARContai nerOwnerObj Li st   *val ue,  
    ARBool ean                freeStruct)  
  
voi d FreeARContai nerOwnerObj Li stLi st(  
    ARContai nerOwnerObj Li stLi st   *val ue,  
    ARBool ean                freeStruct)  
  
voi d FreeARContai nerTypeLi st(  
    ARContai nerTypeLi st     *val ue,  
    ARBool ean                freeStruct)  
  
voi d FreeARCoordLi st(  
    ARCoordLi st             *val ue,  
    ARBool ean                freeStruct)  
  
voi d FreeARCurrencyLi st(  
    ARCurrencyLi st          *val ue,  
    ARBool ean                freeStruct)  
  
voi d FreeARCurrencyStruct(  
    ARCurrencyStruct         *val ue,  
    ARBool ean                freeStruct)  
  
voi d FreeARDDEStruct(  
    ARDDEStruct              *val ue,  
    ARBool ean                freeStruct)
```

```
voi d FreeARDi aryLi st(
    ARDi aryLi st
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeARDi spl ayl nstanceLi st(
    ARDi spl ayl nstanceLi st
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeARDi spl ayl nstanceLi stLi st(
    ARDi spl ayl nstanceLi stLi st
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeARDi spl ayl nstanceStruct(
    ARDi spl ayl nstanceStruct
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeARDi spl ayLi st(
    ARDi spl ayLi st
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeAREntryl dLi st(
    AREntryl dLi st
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeAREntryl dLi stLi st(
    AREntryl dLi stLi st
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeAREntryl nfoLi st(
    AREntryl nfoLi stLi st
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeAREntryLi stFi el dLi st(
    AREntryLi stFi el dLi st
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeAREntryLi stFi el dLi stLi st(
    AREntryLi stFi el dLi stLi st
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeAREntryLi stFi el dVal ueLi st(
    AREntryLi stFi el dVal ueLi st
    ARBool ean
    *val ue,
    freeStruct)
```

```
voi d FreeAREntryListList( *val ue,
                            AREntryListList st
                            ARBool ean
                            freeStruct)

voi d FreeARFi el dAssi gnList( *val ue,
                                   ARFi el dAssi gnList st
                                   ARBool ean
                                   freeStruct)

voi d FreeARFi el dAssi gnStruct( *val ue,
                                   ARFi el dAssi gnStruct
                                   ARBool ean
                                   freeStruct)

voi d FreeARFi el dInfoList( *val ue,
                             AREntryFileInfo dInfoList st
                             ARBool ean
                             freeStruct)

voi d FreeARFi el dInfoStruct( *val ue,
                               AREntryFileInfo dInfoStruct
                               ARBool ean
                               freeStruct)

voi d FreeARFi el dList mi tList( *val ue,
                                   ARFi el dList mi tList st
                                   ARBool ean
                                   freeStruct)

voi d FreeARFi el dList mi tStruct( *val ue,
                                   ARFi el dList mi tStruct
                                   ARBool ean
                                   freeStruct)

voi d FreeARFi el dMappingList( *val ue,
                               ARFi el dFi el dMappingList st
                               ARBool ean
                               freeStruct)

voi d FreeARFi el dValueList( *val ue,
                             ARFi el dValueList st
                             ARBool ean
                             freeStruct)

voi d FreeARFi el dValueListList( *val ue,
                                   ARFi el dValueListList st
                                   ARBool ean
                                   freeStruct)

voi d FreeARFi el dValueOrAri thStruct( *val ue,
                                         ARFi el dValueOrAri thStruct
                                         ARBool ean
                                         freeStruct)
```

```
void FreeARFi ei dVal ueStruct(  
    ARFi ei dVal ueStruct  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
void FreeARFi l terActi onLi st(  
    ARFi l terActi onLi st  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
void FreeARFi l terActi onNoti fyAdvanced(  
    ARFi l terActi onNoti fyAdvanced  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
void FreeARFi l terActi onStruct(  
    ARFi l terActi onStruct  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
void FreeARFul l TextI nfoLi st(  
    ARFul l TextI nfoLi st  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
void FreeARFul l TextI nfoRequestLi st(  
    ARFul l TextI nfoRequestLi st  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
void FreeARFuncti onAssi gnStruct(  
    ARFuncti onAssi gnStruct  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
void FreeARGotoGui deLabel Struct(  
    ARGotoGui deLabel Struct  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
void FreeARGroupI nfoLi st(  
    ARGroupI nfoLi st  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
void FreeARHostI DTyPeLi st(  
    ARHostI DTyPeLi st  
    ARBool ean  
    *val ue,  
    freeStruct)  
  
void FreeARI ndexLi st(  
    ARI ndexLi st  
    ARBool ean  
    *val ue,  
    freeStruct)
```

```
voi d FreeARI ndexLi stLi st(
    ARI ndexLi stLi st
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeARI nternal l dLi st(
    ARI nternal l dLi st
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeARI nternal l dLi stLi st(
    ARI nternal l dLi stLi st
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeARLi censel nfoLi st(
    ARLi censel nfoLi st
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeARLi censel nfoStruct(
    ARLi censel nfoStruct
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeARLi censeNameStruct(
    ARLi censeNameStruct
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeARLi censeVal i dLi st(
    ARLi censeVal i dLi st
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeARLi censeVal i dStruct(
    ARLi censeVal i dStruct
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeARLocStruct(
    ARLocStruct
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeARMacroParmLi st(
    FreeARMacroParmLi st
    ARBool ean
    *val ue,
    freeStruct)

voi d FreeARNameLi st(
    ARNameLi st
    ARBool ean
    *val ue,
    freeStruct)
```

```
void FreeAROpenDI(gStruct(
    AROpenDI gStruct *value,
    ARBool ean freeStruct))

void FreeARObjectInfoList(
    ARObjectListInfoList *value,
    ARBool ean freeStruct)

void FreeARObjectInfoStruct(
    ARObjectInfoStruct *value,
    ARBool ean freeStruct)

void FreeARPermissList(
    ARPermissList *value,
    ARBool ean freeStruct)

void FreeARPermissListList(
    ARPermissListList *value,
    ARBool ean freeStruct)

void FreeARPropList(
    ARPropList *value,
    ARBool ean freeStruct)

void FreeARPropListList(
    ARPropListList *value,
    ARBool ean freeStruct)

void FreeARPropStruct(
    ARPropStruct *value,
    ARBool ean freeStruct)

void FreeARPushFieldList(
    ARPushFieldList *value,
    ARBool ean freeStruct)

void FreeARQualifierFieldList(
    ARQualifierFieldList *value,
    ARBool ean freeStruct)

void FreeARQualifierFieldStruct(
    ARQualifierFieldStruct *value,
    ARBool ean freeStruct)
```

```
void FreeARReferenceList(
    ARReferenceList *value,
    ARBool ean
    freeStruct)

void FreeARReferenceListList(
    ARReferenceListList *value,
    ARBool ean
    freeStruct)

void FreeARReferenceTypeList(
    ARReferenceTypeList *value,
    ARBool ean
    freeStruct)

void FreeARReferenceStruct(
    ARReferenceStruct *value,
    ARBool ean
    freeStruct)

void FreeARRelOpStruct(
    ARRelOpStruct *value,
    ARBool ean
    freeStruct)

void FreeARRoleInfoList(
    ARRoleInfoList *value,
    ARBool ean
    freeStruct)

void FreeARServerInfoList(
    ARServerInfoList *value,
    ARBool ean
    freeStruct)

void FreeARServerInfoRequestList(
    ARServerInfoRequestList *value,
    ARBool ean
    freeStruct)

void FreeARServerNameList(
    ARServerNameList *value,
    ARBool ean
    freeStruct)

void FreeARSigналList(
    ARSignalList *value,
    ARBool ean
    freeStruct)

void FreeARSortList(
    ARSortList *value,
    ARBool ean
    freeStruct)
```

```
void FreeARSortListList(
    ARSortListList *value,
    ARBool ean
    freeStruct)

void FreeARSQLStruct(
    ARSQLStruct *value,
    ARBool ean
    freeStruct)

void FreeARStatisticsResultList(
    ARStatisticsResultList *value,
    ARBool ean
    freeStruct)

void FreeARStatusHistoryList(
    ARStatusHistoryList *value,
    ARBool ean
    freeStruct)

void FreeARStatusList(
    ARStatusList *value,
    ARBool ean
    freeStruct)

void FreeARStructItemList(
    ARStructItemList *value,
    ARBool ean
    freeStruct)

void FreeARSupportFileInfoList(
    ARSupportFileInfoList *value,
    ARBool ean
    freeStruct)

void FreeARSupportFileInfoStruct(
    ARSupportFileInfoStruct *value,
    ARBool ean
    freeStruct)

void FreeARTextStringList(
    ARTextStringList *value,
    ARBool ean
    freeStruct)

void FreeARTimestampList(
    ARTimestampList *value,
    ARBool ean
    freeStruct)

void FreeARUnsignedIntList(
    ARUnsignedIntList *value,
    ARBool ean
    freeStruct)
```

```
void FreeARUserList(
    ARUserList *value,
    ARBool ean
)
void FreeARUserListStruct(
    ARUserListStruct *value,
    ARBool ean
)
void FreeARUserListStruct(
    ARUserListStruct *value,
    ARBool ean
)
void FreeARValueList(
    ARValueList *value,
    ARBool ean
)
void FreeARValueListList(
    ARValueListList *value,
    ARBool ean
)
void FreeARValueStruct(
    ARValueStruct *value,
    ARBool ean
)
void FreeARVuiInfoList(
    ARVuiInfoList *value,
    ARBool ean
)
void FreeARVuiInfoStruct(
    ARVuiInfoStruct *value,
    ARBool ean
)
void FreeARWaitStruct(
    ARWaitStruct *value,
    ARBool ean
)
void FreeARWorkflowConnectList(
    ARWorkflowConnectList *value,
    ARBool ean
)
void FreeARWorkflowConnectStruct(
    ARWorkflowConnectStruct *value,
    ARBool ean
)
```

```
void FreeARXMLInputDoc(  
    ARXMLInputDoc *value,  
    ARBoolean freeStruct)  
  
void FreeARXMLOutputDoc(  
    ARXMLOutputDoc *value,  
    ARBoolean freeStruct)  
  
void FreeARXMLParserHandle(  
    ARXMLParserHandle *value,  
    ARBoolean freeStruct)  
  
void FreeARXMLParsedStream(  
    ARXMLParsedStream *value,  
    ARBoolean freeStruct)  
  
void FreeListARCharMenuStruct(  
    ListARCharMenuStruct *value,  
    ARBoolean freeStruct)
```

## Input arguments

### value

A pointer to the structure you want to free. The system does not perform an operation if the structure is a list with zero items or you specify `NULL` for this parameter.

### freeStruct

A flag indicating whether you need to free the top-level structure. If you allocated memory for the top-level structure, specify 1 (`TRUE`) to free both the structure and its contents. If you used a stack variable for the top-level structure, specify 0 (`FALSE`) to free only the contents of the structure.



Chapter

# 5

# Creating and executing AR System C API programs

This chapter explains how to initialize and shutdown the system and how to perform other common tasks related to the C API programs.

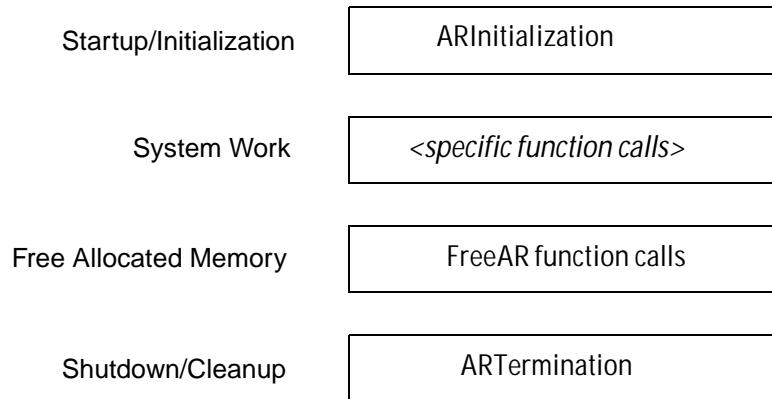
The following topics are provided:

- Program structure (page 434)
- Performing common tasks (page 436)
- Specifying fields or keywords (page 440)
- Error checking (page 441)
- Executing C API programs in workflow (page 443)
- Program design tips (page 445)
- Multithreaded C API clients (page 445)
- Impersonating a user (page 446)

# Program structure

The AR System C API programs consist of the following sections:

Figure 5-1: Organization of C API programs



<b>Startup/Initialization</b>	Call ARInitialization to perform server- and network-specific initialization operations for connecting to the AR System servers (required).
<b>System Work</b>	Call one or more C API functions to perform the specific work of your program.
<b>Free Allocated Memory</b>	Call one or more of the FreeAR (or use the free function) to free all allocated memory associated with a specific data structure (see “Freeing allocated memory” on page 112).
<b>Shutdown/Cleanup</b>	Call ARTermination to perform environment-specific cleanup routines and disconnect from the AR System servers (required). If you are using floating licenses and do not disconnect from the server, your license token is unavailable for other users for the defined time-out interval.

The following code fragment provides an example of the generic program structure. Because almost every AR System C API function has a control parameter as its first input argument, the template includes code for loading the ARControl Struct structure (see “Login and session information” on page 45).

---

**Note:** ARI initialization does not use login information from the control structure, so you can load it into the control structure either before or after you issue the ARI initialization call. In the example, it is done after.

---

```
{
ARControl Structcontrol;
ARStatusLi ststatus;
int      rtn;
...
...
/*
/*Perform Startup*/
/*
if (ARI ni ti al i zati on(&control , &status) >= AR_RETURN_ERROR){
    printf("\n **** ini ti al i zati on error ***\n");
    Pri ntARStatusLi st(&status);
    exit(3);
}
FreeARStatusList(&status, FALSE);

/*
/*Load ARControl Struct*/
/*
strncpy(control . user, "Demo", AR_MAX_ACCESS_NAME_SI ZE); /* use Demo
user */
control . user[AR_MAX_ACCESS_NAME_SI ZE] = '\0';
strcpy(control . password, ""); /* coul d load from fi le*/
memset(&control . local ei nfo, 0, si zeof(ARLocal i zati onl nfo)); /* use
default local e*/
strncpy(control . server, argv[1], AR_MAX_SERVER_SI ZE);
control . server[AR_MAX_SERVER_SI ZE] = '\0';
strcpy(control . authString, ""); /* coul d load from fi le */

/*
/*Perform System Work*/
/*
rtn = ARGetEntry(&control , ... , &status);
if( rtn ... )
...
...

/*
/*Perform Cl eanup*/
/*
ARTermi nati on(&control , &status);
FreeARStatusList(&status, FALSE);
}
```

# Performing common tasks

Each object in the AR System has an `ARGetList<object>` function that returns a list of records that match the specified criteria. Depending on the object, this list contains zero or more names or IDs (or both) that uniquely identify the record. The following table shows the unique identifier for each AR System object.

AR System object	Return parameter	Parameter type	Unique identifier
Active link	nameList	ARNameList *	Name
Character menu	nameList	ARNameList *	Name
Container	conList	ARContainerInfoList *	Name
Entry	entryList	AREntryListList *	ID
Escalation	nameList	ARNameList *	Name
Field	idList	ARInternalIdList *	ID
Filter	nameList	ARNameList *	Name
Group	groupList	ARGroupInfoList *	ID
Schema	nameList	ARNameList *	Name
Server	serverList	ARServerNameList *	Name
User	userList	ARUserInfoList *	Name
View (VUI)	idList	ARInternalIdList *	ID

These names or IDs can then be passed as input arguments to the `ARGet<object>`, `ARSet<object>`, or `ARDelete<object>` functions to perform the corresponding operation on those records.

Unqualified `ARGetList<object>` calls generate a complete list of records (for a particular object). You can select a subset of these records by using a qualification. When retrieving a list of entries, you can specify any set of selection conditions by using `ARQualifierStruct` (see “Representing qualifications” on page 55). For other objects, you can retrieve lists limited to records modified after a specified date and time value. For fields, views, filters, escalations, and active links, you can also limit it to items associated with a specified schema.

The following examples use common tasks to illustrate these concepts. Each of the first two examples include a brief task description, an outline of the conceptual steps involved, and a diagram of the C API functions and parameters used to accomplish the task. These examples also demonstrate the general problem-solving approach of breaking tasks into conceptual steps. This approach might help you identify the sequence of C API functions needed to solve your programming problem.

The third example consists of a small sample program that retrieves and prints a list of available AR System servers.

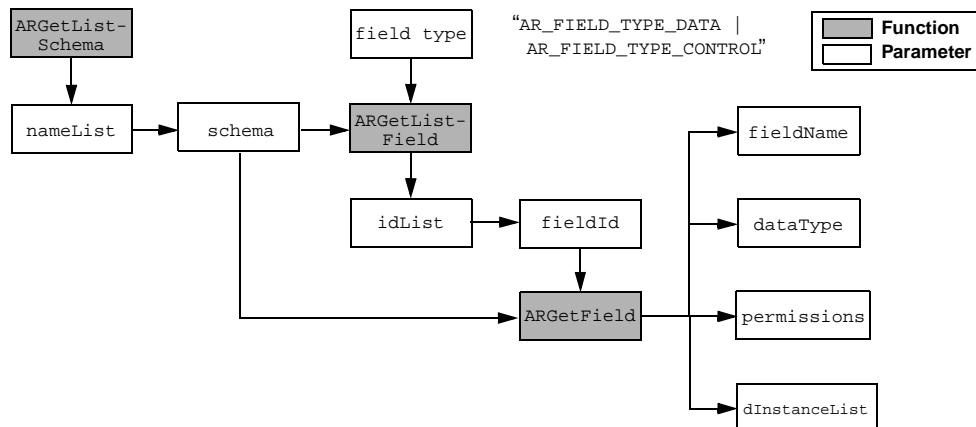
## Example 1 Retrieving field properties

The goal in this example is to retrieve selected properties for a particular schema field. In this case, the field is a data or control field, and the properties shown are the field name, data type, access control information, and display properties. The following high-level steps are illustrated in the following figure.

#### ► To retrieve selected properties for a field

- 1 Retrieve a list of available schemas (ARGetListSchema), and select the desired schema name (schema) from the list (nameList).
  - 2 Retrieve a list of data, trim, and control fields (fieldList dType) associated with the schema (ARGetListFields).
  - 3 Select the desired field (fieldList d) from the list (fieldList), and retrieve the specified properties for that field (ARGetField).

Figure 5-2: Retrieving selected properties from a field



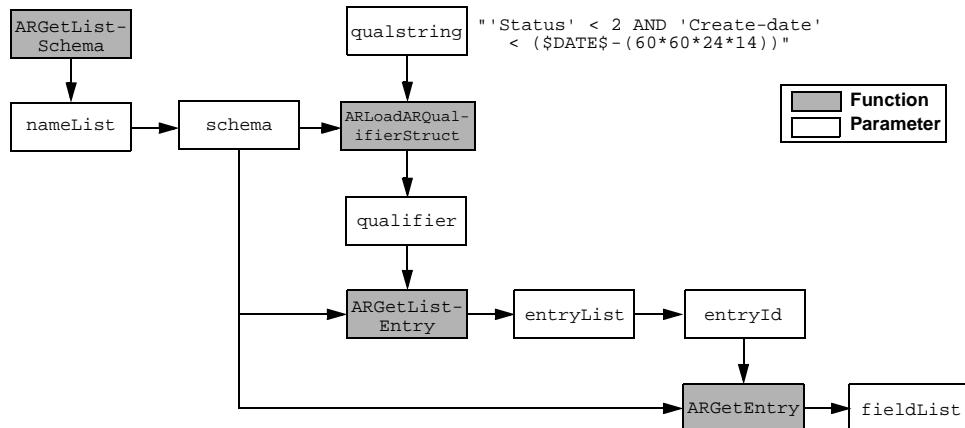
## Example 2 Retrieving selected entries

The goal in this example is to retrieve the set of schema entries that match specified query conditions. The selection criteria are those entries that are either New or Open and were created more than two weeks ago. The following high-level steps are illustrated in the following figure.

### ► To retrieve selected entries from a schema

- 1 Retrieve a list of available schemas (ARGetListSchema), and select the desired schema name (schema) from the list (nameList).
- 2 Create a string with the desired query conditions (qualString), and place it into the proper structure (ARLoadARQualifierStruct).
- 3 Retrieve a list of entries (ARGetListEntry) from the schema that match the query conditions (qualifier).
- 4 If desired, select a particular entry (entryId) from the list (entryList), and retrieve the field data (fieldList) for that entry (ARGetEntry).

Figure 5-3: Retrieving the field data



## Example 3 Retrieving a schema list

```

#include "ar.h"
#include "arextern.h"
#include "arfree.h"

void printStatusList(ARStatusList *theList)
{
    unsigned int i;
    for (i = 0; i < theList->numItems; i++)

```

```

    {
        if (theList->statusList[i].appendedText == NULL ||
            theList->statusList[i].appendedText[0] == '\0')
            printf("%s", theList->statusList[i].messageText);
        else
            printf("%s : %s", theList->statusList[i].messageText,
                   theList->statusList[i].appendedText);
        printf(" (ARERR %d)\n", theList->statusList[i].messageNum);
    }
}

int main(void)
{
    ARControlStruct control = {0, 0, "Demo", "", "", 0, "", "yourservername"};
    ARNameList formNameList;
    unsigned int i;
    ARStatusList status;
    if (ARInitialization(&control, &status) >= AR_RETURN_ERROR)
    {
        printStatusList(&status);
        FreeARStatusList(&status, FALSE);
        return 1;
    }

    FreeARStatusList(&status, FALSE);

    if (ARGetListSchema(&control, 0,
                        AR_LIST_SCHEMA_ALL | AR_INCREMENTAL,
                        NULL, NULL, NULL,
                        &formNameList, &status) >= AR_RETURN_ERROR)
        printStatusList(&status);
    else
    {
        for (i = 0; i < formNameList.numItems; i++)
            printf("%s\n", formNameList.nameList[i]);
    }

    FreeARStatusList(&status, FALSE);
    FreeARNameList(&formNameList, FALSE);
    if (ARTermination(&control, &status) >= AR_RETURN_ERROR)
        printStatusList(&status);
    FreeARStatusList(&status, FALSE);
    return 0;
}

```

# Specifying fields or keywords

The AR System include several actions that involve text:

- Sending a notification (filters and escalations)
- Sending a message (filters and active links)
- Running a process (filters, escalations, and active links)
- Specifying a macro parameter (active links)

Because BMC Remedy User and BMC Remedy Administrator run in a localized environment, you can include a field or keyword value in a character string by specifying the name of the desired field or keyword.

A C API program, however, can operate in multiple environments. Because field names and keywords vary across environments, you must specify the field ID or keyword code (which are environment-independent) to include one of these values in a character string. The following table shows the syntax for specifying a field or keyword value.

Field	\$id\$	ID of the desired field (for example, \$7\$)
Keyword	\$-code\$	Code associated with the desired keyword in the ar.h file (for example, \$-5\$ for the current schema). Use \$--1\$ to specify NULL. A hyphen is required to distinguish a keyword code from a field ID.

**Note:** Field IDs and keyword codes are not required when specifying a qualification string for the `ARLoadARQualifierStruct` function because it automatically translates all values for you.

# Error checking

All the C API functions (except `FreeAR`) return a status parameter (`ARstatusList`) that consists of zero or more notes, warnings, or errors generated from the function call (see “Status information” on page 48).

Each message consists of a value indicating the type of message, the message number, and the message text (in the language specified in the control structure). More serious errors are listed first with lesser warnings and informational messages following. Within each category, messages are listed in reverse chronological order, with the most recent message first.

The return of the function itself is an integer value indicating the success or failure of the call (see the following table). This value represents the status associated with the first (most recent and most serious) message in the list.

0	<code>AR_RETURN_OK</code>	Operation successful—status might contain one or more <i>informational</i> messages.
1	<code>AR_RETURN_WARNING</code>	Operation successful, but a problem was encountered—status contains one or more <i>warning</i> messages and might also contain informational messages.
2	<code>AR_RETURN_ERROR</code>	Operation failed—status contains one or more <i>error</i> messages and might also contain warning or informational messages.
3	<code>AR_RETURN_FATAL</code>	Operation failed—status <i>might</i> contain one or more messages.
4	<code>AR_RETURN_BAD_STATUS</code>	Invalid status parameter—operation canceled.

You can ignore warnings and informational messages, although reporting warnings is often helpful. Because the C API returns all errors in a common structure, you can perform all error handling by using a single, generic routine. The following example provides a sample routine for checking return values.

```
ARStatusLi ststatus;
char      api call [ar_MAX_API CALL_SI ZE]; /* function name*/
int       rtn; /* function return value*/

rtn = ARGetEntry(&control, ..., &status);
strcpy(api call, "ARGetEntry");
swi tch( rtn ){
    case AR_RETURN_OK:
        printf("\t%s: Successful\n", api call);
        PrintARStatusList( &status );
        break;
    case AR_RETURN_WARNING:
        printf("\t%s: Warning\n", api call);
        PrintARStatusList( &status );
        break;
    case AR_RETURN_ERROR:
        printf("\t%s: Error\n", api call);
        PrintARStatusList( &status );
        exit(3);
        break;
    case AR_RETURN_FATAL:
        printf("\t%s: Fatal\n", api call);
        exit(3);
        break;
    case AR_RETURN_BAD_STATUS:
        printf("\t%s: Bad Status\n", api call);
        exit(3);
        break;
    default:
        printf("\t%s: Invalid return value: %d\n", api call, rtn);
        exit(3);
}
```

Because the `status` structure uses allocated memory, you must free that memory after every C API call by using `FreeARStatusList`. You can call these functions regardless of whether the `status` structure contains any messages, because they perform no action if there are no messages to free (for more information, see “Freeing allocated memory” on page 112).

# Executing C API programs in workflow

The most obvious way to execute a C API program is from the command line. You can execute a C API program directly, integrate it with other processes or commands in a shell script, or schedule it for regular execution by using a UNIX cron job or the Windows Scheduler.

The second way to execute a C API program is through workflow. You can call a C API program in a filter, escalation, or active link to perform the Set Fields, Push Fields, or Run Process actions.

## Set fields and push fields

The Set Fields and Push Fields actions assign a specified value to a particular schema field. You can assign a constant, a value from another field, or an operation result value (see “Set fields action” on page 86 or “Push fields action” on page 94). To assign the result of an operating system process or command (including a C API program), specify the following field value:

```
$PROCESS$ <API program> <parameters>
```

The Set Fields and Push Fields operations block the AR System server (the server waits for the operation to complete before performing any other actions). If the Set Fields or Push Fields action calls a C API program that executes on the same AR System server, a deadlock could occur if there are not multiple server threads available to handle calls. When only a single server thread is configured, the server cannot satisfy the request from the API program until it completes the current operation, but the server cannot complete the operation until it executes the API program.

This deadlock causes a time-out, and the Set Fields or Push Fields operation fails. To avoid this problem when running your API program against the same server, configure your server to have multiple fast and list server threads. See the *Optimizing and Troubleshooting* guide for more information about configuring server threads.

---

**Note:** The default process time-out interval is five seconds. You can adjust this setting by using the Server Information window in BMC Remedy Administrator. Valid values are from 1 to 60 seconds.

---

## Run Process

The Run Process action executes the specified operating system process or command. To execute a C API program, specify the following as the command value:

`<API program> <parameters>`

Unlike a Set Fields operation, the Run Process action does *not* block the AR System server. The system places these tasks in the Run Process queue, where they are deferred until the server has completed all database transactions.

If the specified process requires a field value (for example, sending a pager message to the user to whom a particular ticket has been assigned), the server might use different values depending on the type of database operation and whether it was successful. The following table summarizes the possible scenarios.

Database transaction	Submit	Modify
Succeeds	Uses <i>new</i> database value	Uses <i>new</i> database value
Fails	Action not performed	Uses <i>old</i> database value

If the specified process does not require a field value (for example, sending a pager message to a particular user), the server does not execute the queued processes after an unsuccessful database transaction.

# Program design tips

## Use industry-standard tools.

- Use a C or C++ compiler adhering to the ANSI standard. If you are using the API under Windows, you must use Microsoft Visual C++ (version 6.0 or later).
- Use POSIX routines for system calls if the program will run on both UNIX and Windows.

## Use variables (instead of hard coding) for more flexibility and less maintenance.

- Use symbolic constants (#define values from ar.h).
- Use environment variables.
- Use files to store menu or selection values instead of writing them into your program.
- Minimize network traffic.
- Use bulk transfer to retrieve data and to cache the information locally. Omit unnecessary fields when retrieving data.

# Multithreaded C API clients

The AR System API supports multithreaded clients through the use of sessions. Each session maintains its own state information, enabling simultaneous operations against AR System servers. This feature enables more sophisticated client programs to perform multiple operations simultaneously against the same or different servers. You establish a session with a call to `ARI ni ti al i zati on` and terminate it with a call to `ARTermi nati on`. The session identifier returned in the control record from an `ARI ni ti al i zati on` call must be present in the control record for all subsequent API function calls intended to operate within that session. Operations for a session are not restricted to a single thread; however, each session can only be active on one thread at any one time.

## Impersonating a user

 NEW

The ARSetImpersonatedUser API call enables you to set the current user to be any other valid user while still running as aradmin to perform operations as that user. This user's permissions and licensing take effect immediately after this API call is run. The original user must be an administrator. This call can be helpful if you want to debug a problem and see the behavior that the user is experiencing.

When the ARSetImpersonatedUser call is issued, the new user's permissions and licensing take effect. When the server receives an API call with this property, the server validates that the user who is logged in is actually an administrator. The server then skips password validation and creates a user record for the impersonated user. If the call requires licenses, the server acquires the needed licenses on behalf of the user.

When the name of the impersonated user is set to `NULL`, the API automatically issues a `ReleaseCurrentUser` call for the impersonated user. The server switches back to the original administrator user.

---

**Note:** You can issue the ARSetImpersonatedUser call to impersonate a different user instead of setting the name to `NULL`. The permissions and licensing of the new user will take effect, and the previously impersonated user is released.

---

For more information, see “ARSetImpersonatedUser” on page 397.

You can also impersonate a user through the driver program with the `i_muser` command. For more information about the driver program, see “Using the driver program” on page 475.

This chapter describes the Microsoft Automation interfaces supported by BMC Remedy User. You can use an application such as Microsoft Visual Basic to access functionality within BMC Remedy User and to automate repetitive tasks. To best use this information, you should already have a working knowledge of Microsoft Automation programming. For more information about automation programming, see the Microsoft website ([www.microsoft.com](http://www.microsoft.com)).

The OLE capabilities of BMC Remedy User are designed to provide a client-side, single user integration point. For more robust capabilities or server-side integration, use the AR System API.

The following topics are provided:

- Overview (page 449)
- Understanding the automation type library (page 450)
- Building an application to control BMC Remedy User (page 451)
- The BMC Remedy User automation object model (page 455)
- The application object: ICOMAppObj (page 456)
- Login (page 457)
- Logout (page 458)
- GetServerList (page 458)
- GetFormList (page 459)
- OpenForm (page 459)

- LoadForm (page 460)
- GetActiveForm (page 461)
- HasDefaultSession (page 462)
- OpenGuide (page 462)
- RunMacro (page 463)
- The form object: ICOMFormWnd (page 463)
- Submit (page 465)
- Modify (page 465)
- Close (page 465)
- MakeVisible (page 466)
- GetField (page 466)
- GetFieldById (page 466)
- GiveFieldFocus (page 467)
- GiveFieldFocusById (page 467)
- Query (page 467)
- GetServerName (page 468)
- GetFormName (page 468)
- The field object: ICOMField (page 469)
- MakeVisible (page 469)
- MakeReadWrite (page 470)
- Disable (page 470)
- The query result object: ICOMQueryResult (page 470)
- The query result set object: ICOMQueryResultSet (page 471)
- Item (page 471)

---

Note: BMC Remedy User's automation capabilities are a supported feature in a Component Object Model (COM) environment, but not in a Distributed Component Object Model (DCOM) environment.

---

# Overview

BMC Remedy User provides a Microsoft Component Object Model (COM) Automation server, exposing its basic functionality to automation clients located on the same machine. BMC Remedy User automation object model provides the programmer with a basic set of BMC Remedy User functionality. This functionality includes the ability to open a form, perform create, search, and modify operations, get and set field values and properties, open a guide, and run a macro.

---

**Note:** The automation object model provided by BMC Remedy User is not a complete object model in the sense that BMC Remedy User cannot be *fully* driven by an automation controller such as Microsoft Visual Basic.

---

You can write automation client applications that control BMC Remedy User in the foreground, using it for display and user interaction. You can also write automation client applications that control BMC Remedy User in the background, bypassing its display and user interaction. However, if any workflow opens additional windows, those windows will be visible.

Your automation client application can launch a new instance of BMC Remedy User or connect to an instance that is already running. It can also connect to a running AR System application (an instance of BMC Remedy User that manages a prescribed set of forms), but cannot launch one.

Unless otherwise specified by the BMC Remedy User object model, your automation client application can both get and set the interface object's properties. As with the AR System API, the AR System permissions you have when making an automation call are those of the user logged in to the session you establish.

# Understanding the automation type library

Understanding the purpose and use of the *type library* will help you begin assembling custom solutions based on the BMC Remedy User object model.

---

**Important:** The information in this and the following sections is not intended to replace basic training in Microsoft Automation, COM or programming techniques, nor is it meant to provide all the information necessary to become proficient in automation programming. You will benefit most from reading this material if you have had some previous experience using COM.

---

## Accessing type information

Type information consists of compilation data that automation uses to describe objects, properties, and methods to programming tools that access those objects. A compilation of objects and interfaces is called the *type library*. The AR System provides a type library (`aruser.tlb`) installed in the same folder as BMC Remedy User executable file, `aruser.exe`, and must be included in your Microsoft Visual Basic project.

The typical automation controller (such as Microsoft Visual Basic) needs only a description of the automation objects to access those objects. When the automation client executes, it will provide the parameters and call the functions supported by the object as defined by the type library definition.

An automation client can obtain type library information through two mechanisms:

- The first mechanism is programmatic: An automation client can obtain Type Info for the object through the object's `IDispatch` interface.  
Obtaining type library information in this way means that the COM runtime system will start the application that hosts the object.
- The second mechanism is by reading the type library that accompanies the automation server.

This second approach is the approach used by Microsoft Visual Basic.

## Invoking member methods

An automation controller or client calls the methods of a COM object using the following procedure. First, the client must obtain a dispatch ID (usually called a `dispID` by COM programmers) from the type library entry for the object that defines the method to be called. Next, the `dispID` and the parameters to be passed to the method are passed to the objects `IDispatch::Invoke` method. Then, the `Invoke` method asks the COM framework (supplied by Microsoft as part of Windows) to find the method associated with the `dispID` and call it passing the parameters. After the call completes, the return value (if any) is returned to the automation client.

---

**Note:** When an automation server calls a method provided by BMC Remedy User, the COM framework sets up and uses a Remote Procedure Call (RPC).

---

## Building an application to control BMC Remedy User

When using Microsoft Visual Basic or Microsoft Visual C++ to construct a program that accesses the BMC Remedy User object model, you will need to include the type library.

If you construct an automation client without Visual Basic or Microsoft Foundation Class (MFC) support, you will need to either access the type library through the `IDispatch` interface or specify the descriptions in the automation client program. To specify the descriptions, see the sample C++ program at the Customer Support FTP site: <ftp://ftp.remedy.com/pub/ole>. The file for the sample C++ program is called `mfcsample2.exe`.

### Including the type library file

This section presents instructions for including the type library file `aruser.tlb` in your project. The file is installed in the same folder as BMC Remedy User executable file, `aruser.exe`.

#### ► To include the type library in a Microsoft Visual Basic project

- 1 Choose Projects > References.
- 2 Click Browse, and select `aruser.tlb`, located in the same folder as BMC Remedy User.

3 Click Open.

The type library appears as ARUSER in the References dialog.

4 Click the check box next to ARUSER to enable the reference.

► **To include the type library in a Microsoft Visual C++ project using MFC**

1 Choose View > ClassWizard.

2 From the Add Class menu, choose From a Type Library.

3 Select all of the available classes, and click OK.

This creates the aruser.h header file and includes it in your project.

## Handling errors

When a call to an automation interface is unsuccessful, BMC Remedy User returns an appropriate error code. Your client application should implement error-handling routines to catch and display the error. In Visual Basic, precede your OLE calls with On Error statements. In Visual C++, automation calls can be encapsulated with Try and Catch statements.

## Studying a sample program

The following sample Visual Basic program demonstrates the use of some common BMC Remedy User automation calls, providing a user interface that lets a user search the Group form with a combination of query-by-example and a search string. The program was coded in Microsoft Visual Basic 6.0. Form1, the search form, has query-by-example fields for Group Name, Group Type, and Long Group Name, a search string field, and a results field. One button clears the search and the other performs it. Form2, the login form, has user and password fields and an OK button. The code portion of both forms is included here.

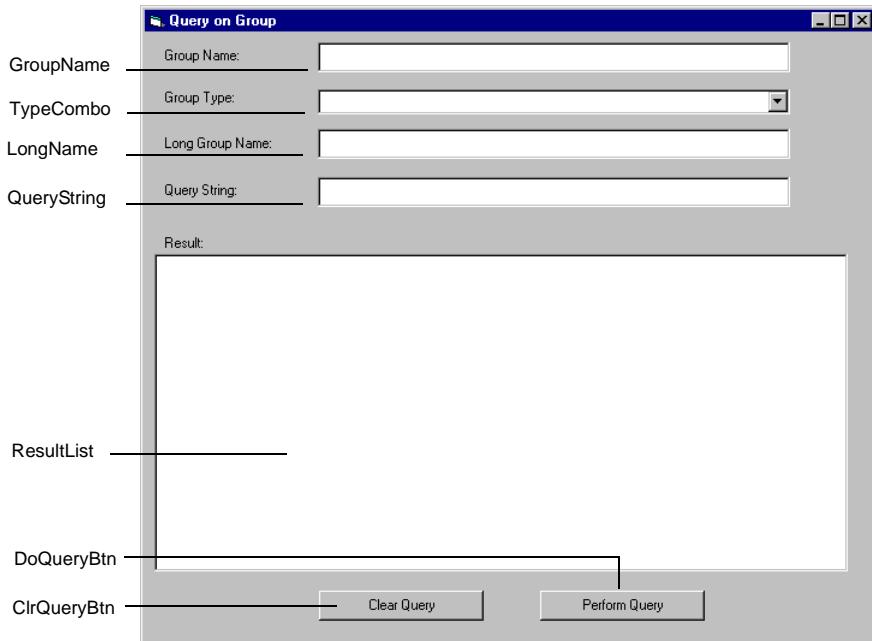
The source for this sample is available from the BMC Remedy Customer Support at the following URL: <ftp://ftp.remedy.com/pub/ole/>. The file for the sample Visual Basic program is called query\_on\_group.zip.

**WARNING:** This sample is supplied “as is.” There is no expressed or implied support for this sample. Trouble-free operation is not guaranteed because there is no expressed or implied testing or validation that has been performed on the sample provided.

## Form1

The following figure shows the start-up form.

Figure 6-1: Start-up form



```

Public App As COMAppObj
Public ARForm As ICOMFormWnd
Public NameField As ICOMField, TypeField As ICOMField,
    LongNameField As ICOMField
Public EntryList As ICOMQueryResultSet
Public sessionID As Long
Public Username As String, Password As String
Private Sub Form_Load()
    Dim gotDefaultSession As Boolean
    Dim terminateString, AutomationError As String
    Dim i As Integer
    ' Initialization
    Set LogInForm = frmLogin

```

```

sessi onl d = 0
termi nateString = Chr(10) & Chr(13) & "Program will be
terminated"
' See if we can connect to an already running instance of Remedy
' User. If not, we will try to launch a new instance.
On Error GoTo LaunchTheApp
Set App = GetObj ect(, "Remedy. User")
' use the following syntax to call an application
' Set App = GetObj ect("drive:\path\name of application-
server name")
GoTo ContinueLoadi ng
LaunchTheApp:
On Error GoTo ErrorHandler
Set App = New COMAppObj
ContinueLoadi ng:
' Test if we need to log in. When Remedy User comes up,
' it will not have a valid session if the user's option is to
' prompt for log in.
On Error GoTo ErrorHandler
gotDefaultSessi on = App. HasDefaultSessi on
If gotDefaultSessi on = False Then
    frmLogi n. Show vbModal , Me
    sessi onl d = App. Logi n(Username, Password, False)
    MsgBox ("Logi n succeeded!")
End If
Dim serverLi st As Vari ant
' Find out which servers are available
serverLi st = App. GetServerLi st(sessi onl d)
For i = 0 To UBound(serverLi st)
    frmServerLi st. cboServerLi st. AddItem (serverLi st(i))
Next
' Let the user select the server
frmServerLi st. Show vbModal , Me
If frmServerLi st. cboServerLi st. Text = "" Then End
On Error GoTo ErrorHandler
Set ARForm = App. OpenForm(sessi onl d,
    frmServerLi st. cboServerLi st. Text, "Group", ARQuery, True)
Set NameFi eld = ARForm. GetFi eld("Group Name")
Set TypeFi eld = ARForm. GetFi eld("Group Type")
Set LongNameFi eld = ARForm. GetFi eld("Long Group Name")
Exit Sub
ErrorHandler:
Automati onError = "Automati on Error: " & Hex(Err. Number) & " " &
Err. Descri ption
MsgBox (Automati onError & termi nateString)
Err. Cl ear
End ' termi nate program
End Sub
Private Sub Form_Unload(Cancel As Integer)
' Clean up before exiting
Set App = Nothing
Set ARForm = Nothing
Set NameFi eld = Nothing
Set TypeFi eld = Nothing

```

```

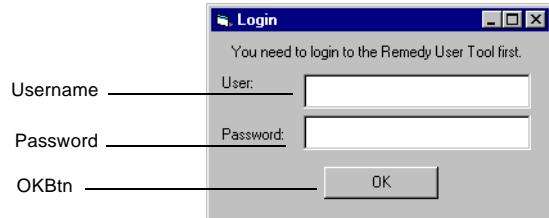
Set LongNameField = Nothing
Set EntryList = Nothing
Cancel = 0
End Sub

```

## Form2

This form is displayed when the user needs to log in.

Figure 6-2: Login form



```

Private Sub cmdOK_Click()
    txtUserName.Refresh
    txtPassword.Refresh
    Username = txtUserName.Text
    Password = txtPassword.Text
    Me.Hide
End Sub

```

## The BMC Remedy User automation object model

The BMC Remedy User Object Model describes how BMC Remedy User can be accessed by an automation client. These objects are used either to collect or expose data in a way that is familiar to AR System application developers. Each BMC Remedy User object has member functions that uniquely apply to that object.

This section describes the automation objects supported by BMC Remedy User Tool and their member functions.

Unless otherwise specified for an interface, your client application can both get and set its properties. As with the AR System API, the AR System permissions you have when making an automation call are those of the user logged in to the session you use.

The remainder of this section will provide specific information about the BMC Remedy User object model.

# The application object: ICOMAppObj

The ICOMAppObj object is the root object of BMC Remedy User's object model. You use it to establish a session and to access form entries. You should explicitly log out of sessions you create to permit proper resource cleanup. See "Methods" on page 456 for a summary of the methods provided by the Application object.

You create an ICOMAppObj object in several different ways, depending on your programming environment and whether you are launching a new instance of BMC Remedy User or connecting with either a running instance of BMC Remedy User or an AR System application. To launch a new instance, call the function in your programming environment that creates and returns a Microsoft COM object. In this case, ask COM to create a COMAppObj. To connect with a running instance, call the function in your environment that gets a reference to a COM object. In this latter case, pass Remedy.User as the ProgID. To connect with a running AR System application, call the function in your environment that gets a reference to a COM object; for the ProgID, pass the full path to BMC Remedy User's .exe file with the application name and the server name appended to it. Use the following syntax:

```
Set Application = GetObject(, "Remedy.User.1")
```

If you are opening BMC Remedy User as an application, use the following syntax:

```
Set Application = GetObject(" <Remedy_User_install_id>\n<app_name>-<server_name>" )
```

**Properties** None.

**Methods** This object has the following member methods.

Method name	Description
Login	Establishes a user session. An automation client can omit this call to instead use the existing session. A newly launched instance of BMC Remedy User will only have an existing session if the Preferences are not set to always prompt for login.
Logout	Terminates the session with the specified ID. An automation client cannot terminate a session established by the desktop user.
GetServerList	Returns the available server names for the specified session.

Method name	Description
GetFormList	Returns the available form names for the specified session and server.
OpenForm	Returns an ICOMFormWnd object to query or submit entries in the specified form.
LoadForm	Returns an ICOMFormWnd object to display or modify an entry in the specified form.
GetActiveForm	Returns an ICOMFormWnd object to the currently active form on the desktop, if one exists.
HasDefaultSession	Returns a flag indicating whether a session is currently active.
OpenGuide	Opens the specified guide on the specified server.
RunMacro	Runs the specified macro, substituting the specified parameters. The macro is run in visible mode. If you run this method against an invisible instance of BMC Remedy User, it will be made visible to the desktop user. If BMC Remedy User is visible but minimized, its window will be brought to the foreground. The desktop user must dismiss any dialog boxes triggered by the macro before your automation client can regain control.

## Login

**Description** Establishes a user session. An automation client can omit this call to instead use the existing session. A newly launched instance of BMC Remedy User will have an existing session only if the Preferences are not set to always prompt for login.

**Synopsis**

```
sessionID = Login(name, password, visiblePrompt)
           char          *name;
           char          *password;
           long          visiblePrompt;
           long          *sessionID;
```

**Input arguments**

**name**  
The user to log in for this session.

**password**

The user's password.

**visiblePrompt**

A code specifying whether errors will (1) or will not (0) be displayed to the desktop user.

**Return values** **sessionID**

The ID of the session established.

**See also** Logout.

## Logout

**Description** Terminates the session with the specified ID. An automation client cannot terminate a session established by the desktop user or a BMC Remedy User instance that has been forced to become visible. An automation client also cannot terminate the default session even if it established the session.

**Synopsis** Logout(sessionID)

long \*sessionID;

**Input arguments** **sessionId**

The ID of the session to terminate.

**See also** Login.

## GetServerList

**Description** Returns the available server names for the specified session.

**Synopsis** retVal = GetServerList(sessionID)

long sessonID;  
variant \*retVal;

**Input arguments** **sessionId**

The ID of the session for which to find available servers. Specify 0 for this parameter to use the current session.

**Return values** [retVal](#)

An array containing the list of available servers.

**See also** [GetFormList](#), [GetServerName](#).

## GetFormList

**Description** Returns the available form names for the specified session and server.

**Synopsis** `retVal = GetFormList(sessionId, serverName)`

long	sessionId;
char	*serverName;
variant	*retVal;

**Input arguments** [sessionId](#)

The ID of the session for which to find available forms. Specify 0 for this parameter to use the current session.

[serverName](#)

The name of the server on which to find available forms.

**Return values** [retVal](#)

An array containing the list of available forms.

**See also** [GetServerList](#), [GetServerName](#), [OpenForm](#).

## OpenForm

**Description** Returns an `I COMFormWnd` object. The `I COMFormWnd` can be used to search or save entries in the specified form.

**Synopsis** `retVal = OpenForm(sessionId, server, form, mode, visible)`

long	sessionId;
char	*server;
char	*form;
long	mode;
Bool	visible;
<code>I COMFormWnd</code>	*retVal;

<b>Input arguments</b>	<p><b>sessionId</b> The ID of the session for which to open the form. Specify 0 for this parameter to use the current session.</p> <p><b>server</b> The name of the server on which to open the form.</p> <p><b>form</b> The name of the form to open.</p> <p><b>mode</b> A flag identifying whether to open the form in search (2) or new (1) mode. You can use the constant ARQuery, which has a value of 2, or ARSubmit which has a value of 1.</p> <p><b>visible</b> A flag identifying whether the form should be opened invisible F (False) or visible T (True). If the form is visible, the desktop user must dismiss any dialog boxes that appear before your automation client can regain control.</p>
<b>Return values</b>	<b>RetVal</b> A reference to the <code>I COMFormWnd</code> object for the form window that was opened.
<b>See also</b>	GetFormList, GetFormName.

## LoadForm

**Description** Returns an `I COMFormWnd` object for the window that was opened. After the form is opened, the request specified by the `entryId` parameter is loaded into the form. The request can then be accessed for display or modify purposes.

**Synopsis**

```
RetVal = LoadForm(sessionId, server, form, entryId, mode,
                  visible)
```

long	sessionId;
char	*server;
char	*form;
char	*entryId;
long	mode;
Boolean	visible;
<code>I COMFormWnd</code>	*RetVal;

<b>Input arguments</b>	<p><b>sessionId</b> The ID of the session for which to open the form. Specify 0 for this parameter to use the current session.</p> <p><b>server</b> The name of the server on which to open the entry.</p> <p><b>form</b> The name of the form containing the entry.</p> <p><b>entryId</b> The Entry ID of the entry to open.</p> <p><b>mode</b> A flag identifying whether to open the form for display (3) or modify (4).</p> <p><b>visible</b> A flag identifying whether the entry should be opened invisible F (False) or visible T (True). If the entry is visible, the desktop user must dismiss any dialog boxes that appear before your automation client can regain control.</p>
<b>Return values</b>	<p><b>RetVal</b> A reference to the entry window that was opened.</p> <p><b>See also</b> OpenForm, GetFormName.</p>

## GetActiveForm

<b>Description</b>	Returns an ICOMFormWnd object to the currently active form in BMC Remedy User, if one exists.
<b>Synopsis</b>	<pre>RetVal = GetActiveForm()            ICOMFormWnd          *RetVal;</pre>
<b>Return values</b>	<p><b>RetVal</b> A reference to the current entry window.</p> <p><b>See also</b> LoadForm, OpenForm.</p>

# HasDefaultSession

**Description**    Return a flag indicating whether a session is currently active.

**Synopsis**      `retval = HasDefaultSession()`

long \*retval;

Return values      `retVal`

A flag indicating whether a session is (1) or is not (0) active in the `ICOMAppObj` object.

**See also**    [Login](#), [Logout](#) [GetServerList](#).

# OpenGuide

**Description** Open the specified guide on the specified server.

**Synopsis**      OpenGui de(sessionId, guiName, serverName)

long session;

char \*gui deName

```
char *serverName;
```

Input arguments	sessionId
The ID of the session.	

The ID of the session for which to open the guide. Specify 0 for this parameter to use the current session.

## guideName

The name of the guide to open.

serverName

The name of the server on which to open the guide.

**See also** OpenForm.

# RunMacro

**Description** Run the specified macro, substituting the specified parameters. The macro is run in visible mode. If you run this method against an invisible instance of BMC Remedy User, it will be made visible to the desktop user. If BMC Remedy User is visible but minimized, its window will be brought to the foreground. The desktop user must dismiss any dialog boxes triggered by the macro before your automation client can regain control.

**Synopsis** RunMacro(sessionId, macroName, parmCount, parmList)

long	sessionId;
char	*macroName;
long	parmCount;
variant	*parmList;

**Input arguments**

### sessionId

The ID of the session for which to run the macro. Specify 0 for this parameter to use the current session.

### macroName

The name of the macro to run. BMC Remedy User must have this macro in its search path.

### parmCount

The number of parameters in parmList.

### parmList

A list of parameters to pass to the macro, in the form of a variable-length array of strings. Each string is in the format parameter=value.

## The form object: ICOMFormWnd

The ICOMFormWnd object represents a form window in one of four modes. You use it to create, search, display, or modify entries. The following table shows a summary of the methods provided by the Form object.

**Properties** None.

**Methods** This object has the following member methods.

Method name	Description
Submit	Submits a new entry to the current form. Field values are taken from the currently open form window.
Modify	Modifies the current entry. Field values are taken from the currently open form window.
Close	Closes the current form. If the window is visible, this method cannot be performed. Only the desktop user can close visible windows.
MakeVisible	Makes a hidden form window visible. This method has no effect on a window that is already visible. If the window is visible, the desktop user must dismiss any dialog boxes that appear before your automation client can regain control.
GetField	Returns an ICOMField object referencing the field with the specified name.
GetFieldById	Returns an ICOMField object referencing the field with the specified ID.
GiveFieldFocus	Sets the input focus to the field with the specified name.
GiveFieldFocusById	Sets the input focus to the field with the specified ID.
Query	Performs the specified query against the current form with the current session.
GetServerName	Returns the name of the server hosting this form.
GetFormName	Returns the name of this form.

## Submit

**Description** Save a new entry to the current form. Field values are taken from the currently open form window.

**Synopsis**

```
pEntryId = Submit()  
char *pEntryId;
```

**Return values**

[pEntryId](#)

The Entry ID of the new entry. This parameter is only returned if the operation was successful.

**See also** [Modify](#), [Close](#), [MakeVisible](#), [Query](#).

## Modify

**Description** Modify the current entry. Field values are taken from the currently open form window.

**Synopsis**

```
Modify()
```

**See also** [Submit](#), [Close](#), [MakeVisible](#), [Query](#).

## Close

**Description** Close the current form. If the window is visible, this method cannot be performed. Only the desktop user can close visible windows.

**Synopsis**

```
Close()
```

**See also** [Submit](#), [Modify](#), [MakeVisible](#), [Query](#).

## MakeVisible

<b>Description</b>	Make a hidden form window visible. This method has no effect on a window that is already visible. If the window is visible, the desktop user must dismiss any dialog boxes that appear before your automation client can regain control.
<b>Synopsis</b>	MakeVi si bl e()
<b>See also</b>	Submit, Modify, Close, Query.

## GetField

<b>Description</b>	Returns an I COMFi el d object for the field with the specified name.
<b>Synopsis</b>	<pre>retval = GetFi el d(field) char *field; I COMFi el d *retval;</pre>
<b>Input arguments</b>	<b>field</b> The name of the field you want a reference to.
<b>Return values</b>	<b>retval</b> An I COMFi el d object for the field.
<b>See also</b>	GetFieldById, GiveFieldFocus, GiveFieldFocusById, Value.

## GetFieldById

<b>Description</b>	Returns an I COMFi el d object for the field with the specified ID.
<b>Synopsis</b>	<pre>retval = GetFi el dById(fieldId) long fieldId; I COMFi el d *retval;</pre>
<b>Input arguments</b>	<b>fieldId</b> The ID of the field you want a reference to.

**Return values** [retVa](#)

An ICOMField object to the field.

**See also** [GetField](#), [GiveFieldFocus](#), [GiveFieldFocusById](#), [Value](#).

## GiveFieldFocus

**Description** Sets the input focus to the field with the specified name.

**Synopsis** Gi veFi el dFocus(fi el d)

char fi el d;

**Input arguments**

[field](#)

The name of the field to receive the input focus.

**See also** [GetField](#), [GiveFieldFocus](#), [GiveFieldFocusById](#), [Value](#).

## GiveFieldFocusById

**Description** Sets the input focus to the field with the specified ID.

**Synopsis** Gi veFi el dFocusByI d(fi el dI d)

long fi el dI d;

**Input arguments**

[fieldId](#)

The ID of the field to receive the input focus.

**See also** [GiveFieldFocus](#), [GetField](#), [GetFieldById](#), [Value](#).

## Query

**Description** Performs the specified search against the current form with the current session.

**Synopsis** retVal = Query(queryStri ng)

char	*queryStri ng;
ICOMQueryResul tSet	*retVal ;

<b>Input arguments</b>	<b>queryString</b>
	The search criteria for your search, in the same format used in the BMC Remedy User advanced search bar. If the current session is visible, any field values entered in the window will also be used in the search criteria.
<b>Return values</b>	<b>RetVal</b>
	An I COMQueryResultSet object used to the list of requests found by your search.
<b>See also</b>	Submit, Modify, Close, MakeVisible.

## GetServerName

<b>Description</b>	Returns the name of the server hosting this form.
<b>Synopsis</b>	pName = GetServerName()  char *pName;
<b>Return values</b>	<b>pName</b>  The name of the server.
<b>See also</b>	GetFormName, GetServerList, GetFormList, server.

## GetFormName

<b>Description</b>	Returns the name of this form.
<b>Synopsis</b>	pName = GetFormName()  char *pName;
<b>Return values</b>	<b>pName</b>  The name of the form.
<b>See also</b>	GetServerName, GetServerList, GetFormList, OpenForm.

# The field object: ICOMField

The ICOMField object represents a field in the current form. You use it to retrieve or set the field's value, change the field's visibility, specify whether the field is read-only, and enable or disable the field. The following table shows a summary of the member methods provided by the Field object.

Properties	Value
	A string representing the field's value
Methods	This object has the following member methods.
Method name	Description
MakeVisible	Makes the field visible or invisible to the desktop user.
MakeReadWrite	Makes the field read-write or read-only.
Disable	Disables the field. The methods of a disabled field's interface cannot be called, and it is grayed and unavailable to the desktop user.

## MakeVisible

**Description** Make the field visible or invisible to the desktop user.

**Synopsis** MakeVisible(visible)

long visible;

**Input arguments** **visible**  
A flag specifying whether the field is invisible (0) or visible (1).

**See also** MakeReadWrite, Disable, GetField, GetFieldById.

## MakeReadWrite

Make the field read-write or read-only.

**Synopsis** MakeReadWrite(readOnly y)

I long

readOnly y;

**Input  
arguments**

**readOnly**

A flag specifying whether the field is read-write (0) or read-only (1).

**See also** MakeVisible, Disable, GetField, GetFieldById.

## Disable

**Description** Disable the field. The methods of a disabled field's interface cannot be called, and it is disabled (grayed) and unavailable to the desktop user.

**Synopsis** Disable()

**See also** MakeVisible, MakeReadWrite, GetField, GetFieldById.

## The query result object: ICOMQueryResult

The ICOMQueryResult object represents a request found by a search. This object is obtained from the ICOMQueryResultSet object. You use this object to retrieve the Entry ID and short description of a request.

**Properties** **entryId**

A string representing the ID of the request. This property is read-only.

**Description**

A string representing the short description of the request. This property is read-only.

**Methods** This object has no methods.

# The query result set object: ICOMQueryResultSet

The ICOMQueryResultSet object represents a list of requests found by a search. You use it to retrieve individual entries from the list, determine how many entries were found by the search, and retrieve the name of the form and server where the search was performed.

## Properties

### Count

A long representing the number of items in the list.

### form

A string representing the name of the form where the search was performed.

### server

A string representing the name of the server where the search was performed.

## Methods

This object has one member method.

# Item

**Description** Retrieves a reference to an item from the list.

## Synopsis

```
retval = Item(pos)  
long pos;  
ICOMQueryResult *retval;
```

## Input arguments

### pos

The position in the list of the item you want a reference to. The first item is numbered 1.

## Return values

### retval

An ICOMQueryResult object for the specified result item.

## See also

Count, entryId, Description.



Chapter

# 7

# Debugging and maintenance

This chapter explains how to solve problems with your API program by using logging and the `driver` program.

The following topics are provided:

- Logging AR System activity (page 474)
- Using the driver program (page 475)

# Logging AR System activity

Log files are a useful debugging tool when your C API program does not produce the expected results. Turning on API logging causes the server to record all API calls it receives in the log file you specify. Each entry in the log file identifies the function name, key parameter values, and the return value. Similarly, turning on filter logging causes the system to record all filter actions in a specified log file.

---

**Note:** Choosing File > Server Information in BMC Remedy Administrator enables you to turn logging on and off. To avoid log file overflow, turn logging off when you have finished your debugging activities. For additional information about log files, see the *Configuring and Optimizing and Troubleshooting* guides.

---

The system automatically renames your log files to <*logfile*>.bak each time you start a new trace. For example, the file arapi.log is renamed arapi.log.bak. The system also renames your log files if the AR System server fails while logging is turned on. You can configure the server to append existing files or rename them and create new log files.

Each log file entry begins with one of the tags in the following table. The tag identifies the action associated with the entry and helps you determine the sequence of events when a log file contains output from multiple traces.

Tag	Source action
<ALRT>	Alert
<API >	API function call
<DSO>	DSO
<ESCL>	Escalation
<FLTR>	Filter
<SQL>	SQL command
<THRD>	Thread
<USER>	Login and logout

Logging the actions of your API program can help you determine why your program is not performing the operation you expect. In some cases, you might discover that a filter or escalation is interfering with your program, thereby causing unexpected results.

## Using the driver program

Installing the API package creates a series of directories in the AR System installation directory. In UNIX environments, the `src` directory contains subdirectories with source code for the `dri ver` sample program. In Windows environments, the API includes source code for the `dri ver` program only (see “Sample Source code” on page 36).

The `dri ver` program provides a command line interface for calling every API function, and viewing return values of certain calls (where appropriate). It also includes print routines for every data structure in the API, making it a useful manual debugging tool. The `dri ver` program is not interactive, but you can use it to log in to the AR System server and observe the results and behavior of the equivalent function calls you make in your API programs.

After compiling the source code or locating the prebuilt program supplied with the API, you can use `dri ver` for a number of purposes:

- To identify function input parameters and load them with appropriate values
- To examine the content and structure of function output parameters
- To experiment with different parameter values

### ► To use the driver program

- 1 Make sure your AR System server is running.
- 2 For Windows, double-click the `dri ver.exe` icon. For UNIX, change directories to the `<ar_install_dir>/api /src/dri ver` directory and type `dri ver` at the prompt.
- 3 Specify the login parameters with the `log` command.  
Use an Administrator login so that you have administrative privileges. Demo is the default system administrator.
- 4 Initialize an API session with the `init` command.

- 5 Type the abbreviation of the function call at the command line, and supply the appropriate input parameter values. (For a list of abbreviations, type ?)

When you are working with the specific commands, see Chapter 4, “AR System C API calls,” to enter the appropriate values for the function parameters. Also, if you are working with specific entries, use leading zeros to see the entry ID of those entries.

---

**Note:** The `dri ver` program is provided as sample source code and is not a supported AR System utility.

---

## Using print.C routines

One of the most helpful components of the `dri ver` program is the set of print routines located in the `print. C` file. These routines enable you to print the contents of any data structure in the API. The routines provide code examples for accessing the various structure members. Printing the contents of a structure before and after an API call is a useful debugging tool.

---

**Note:** See the function definitions in `print. C` to determine the specific parameters and their types for these routines.

---

The following examples illustrate the use of these routines.

### Code fragment that calls three print.C functions

```
main()
{
    ...
    PrintARFi el dVal ueLi st(&fi el dLi st);
    PrintReal ("header", &val ue);
    PrintAREntryl dLi st("header", "header2", &val ue);
    ...
}
```

### Code fragment that shows the output after the AR System executes the command `print_entry fred EX000003`

(where the server name is `fred` and the entry ID is `EX000003`)

```
numl tems = 9
ID= FIELD VALUE
-----
1 = (char) EX000003
2 = (char) snoopy
3 = (timestamp) 11/22/94
4 = (char) snoopy
5 = (char) Demo
6 = (timestamp) 11/23/94
7 = (selection) 2
8 = (char) TESTING
15(S.H.) 5 stat hist items
0 - 1/18/95Demo
1 - 1/19/95Demo
2 -
3 -
4 - 1/26/95Demo
```

The `print.h` file contains a complete list of these routines. Some of the more commonly used routines are:

- `Pri ntAREntryI dLi st`
- `Pri ntAREntryLi stFi el dStruct`
- `Pri ntAREntryLi stStruct`
- `Pri ntARI nternal I dLi st`
- `Pri ntARPermi ssi onLi st`
- `Pri ntARStatusHi storyLi st`
- `Pri ntARStatusStruct`
- `Pri ntAREntryLi stFi el dLi st`
- `Pri ntAREntryLi stLi st`
- `Pri ntARFi el dVal ueLi st`
- `Pri ntARNameLi st`
- `Pri ntARQual i fi erStruct`
- `Pri ntARStatusLi st`

## Using the driver program from the command line

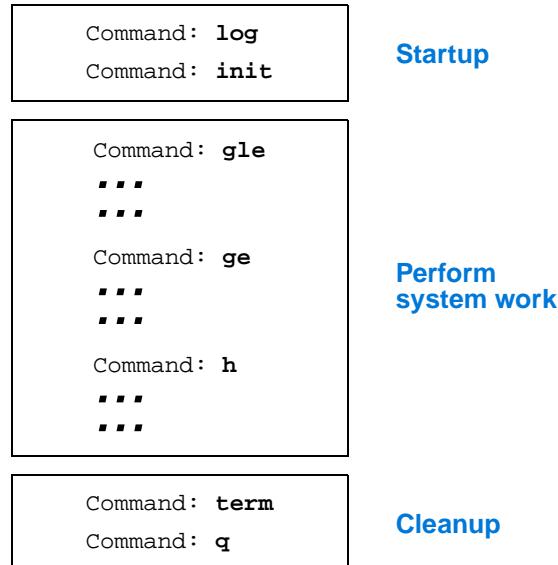
After compiling the source code or locating the prebuilt program supplied with the API, you are ready to use the `dri ver` program. When you execute the program, the system displays the list of `dri ver` commands. The following figure shows a subset of the commands.

Figure 7-1: Driver program commands

AR System API Driver					
Active Link	Escalation	Filter	Entry	Entry	
get <gal>	get <ges>	get <gf>	get <ge>	merge <me>	
set <sal>	set <ses>	set <sf>	set <se>	stats <stat>	
create <cal>	create <ces>	create <cf>	create <ce>	get BLOB <geb>	
delete <dal>	delete <des>	delete <df>	delete <de>	getmult <gme>	
getlist<glal>	getlist<gles>	getlist<glf>	getlist <gle>	getlistblk<gleb>	
getmult<gmal>	getmult<gmes>	getmult<gmf>	getlistw/f<glewf>		
Char Menu	Schema	License	Schema Field	VUI	
get <gc>	get <gs>	val <vl>	get <gsf>	get <gv>	
set <sc>	set <ss>	valmult<vml>	set <ssf>	set <sv>	
create <cc>	create <cs>	create <cl>	create <csf>	create <cv>	
delete <dc>	delete <ds>	delete <dl>	delete <dsf>	delete <dv>	
getlist<glc>	getlist<gls>	getlist<gll>	getlist <glsf>	getlist<glsv>	
getmult<gmc>	gls w/a<glsa>	import <iml>	getmult <gmsf>	getmult <gmv>	
expand <ec>	getmult<gms>	export <exl>	delmult <dmsf>		
exp ssm<essm>	getlist ext<glxsc>		getmult ext cands<gmxfc>		
Container	Alert	Info	Control/Logging	Thread/Timer	
get <gco>	create <cae>	get svr <ssi>	record <rec>	launch <lt>	
set <SCO>	register <rfa>	set svr <ssi>	stop rec <srec>	launch wait <ltw>	
create <cco>	deregistr<dfa>	get FT <gft>	open out <oout>	release wait<rwt>	
delete <dco>	gla user <glau>	set FT <sft>	close out <cout>	sleep <st>	
getlist<glco>	get count<gac>	get stat<gss>	execute <ex>	random sleep<rst>	
getmult<gmc>	decode <dcam>		bgn loop <bl>	msec sleep <mst>	
			end loop <el>		
Init/Term	Encode/Decode	Misc Lists	Misc	Misc	
init <init>	enquiry<ecqal>	server<svr>	ver user <ver>	get file <gfl>	
term <term>	dequery<dcqal>	group <glg>	export <exp>	set file <sfl>	
help <h, ?>	enassig<ecasn>	user <glu>	import <imp>	get errmsg <gem>	
exit <e, q>	deassig<dcasn>	sql <qlsql>	unimport <unimp>	set logging<slog>	
login <log>	echstry<echst>	sql al<sqlal>	exec proc<proc>	close conn <cnc>	
dr ver <dver>	ecdiary<ecdia>	role <glr>	exec p al<epal>	valid cache<vfc>	
bgntran<bhet>	ecdiate <ecdat>		load qual<lgs>	signal <sig>	
endtran<ebet>	dcdate <dcdat>		set port <ssp>	getmult ep <gmep>	
Complex Entry	Session	Config	Localized Val	App States	Currency Ratio
get <xmlge>	get conf<gsc>	get <glv>	get <gas>	getmult sets<gmcrs>	
set <xmlse>	set conf<ssc>	getmult<gmlv>	set <sas>	get ratio <gcr>	
create<xmlce>			getlist <glas>		
Command:					

As in all API programs, you must provide the necessary login information and perform initialization operations for connecting to the AR System server. You can then use the commands to call any number of API functions.

Figure 7-2: Order of driver commands




---

**Note:** Use the `help` command (`h` or `?`) to display the driver commands.

---

When you are finished, you must terminate your interaction with the AR System server and exit the driver program. The following figure illustrates these high-level steps by using sample driver commands.

Unlike calling the API functions directly, the driver interface prompts you for each input parameter. You can also specify driver login information:

- By using the following command line options:

-u	user
-p	password
-l	language
-s	server

For example:

```
dri ver -u Demo -p "fun4me" -s fred
```

- By using the `log` command before or after using the `init` command.

## Creating and using driver scripts

This section describes how to write and execute a driver script.

### ► To create a driver script

- 1 Start the `dri ver` program.
- 2 Enter `rec`, and provide `<script file>` when prompted.
- 3 Enter `log`, and provide login information when prompted.
- 4 Enter `init` (initialize AR System connection).
- 5 Enter the desired `dri ver` commands.
- 6 Enter `term` (terminate AR System connection).
- 7 Enter `srec` (stop recording).

Because `dri ver` scripts are stored as text files, you can edit them by using a text editor (which is sometimes easier than rerecording). Blank lines indicate default or skipped values and should not be deleted.

You have the following options for executing a `dri ver` script:

#### Interactive

- Start the `dri ver` program.
- Enter `ex`, and provide `<script file>` when prompted.

#### Command Line

- Enter `dri ver < script file name>`.  
The `dri ver` program will terminate at the end of the script.

#### Command Line or

#### Shortcut

- Enter `dri ver -x script file name`.

The `dri ver` program will keep running at the end of the script unless the script terminates it with a quit (`q`) command.

To create a shortcut or application icon that will execute your script from a double-click, use this syntax for the shortcut's Target or Command property. It also works directly from the command line.

Chapter

# 8

# AR System plug-ins

This chapter contains information about AR System plug-ins and their corresponding application program interfaces (APIs). AR System plug-ins and APIs extend the functionality of the AR System to external data sources.

For more information, see the *Integrating with Plug-ins and Third-Party Products* guide.

The following topics are provided:

- Overview (page 483)
- LDAP plug-ins (page 483)
- Plug-ins that you create (page 483)
- Plug-in logging facility (page 496)
- Plug-in API calls (page 497)
- ARPluginCreateInstance (page 498)
- ARPluginDeleteInstance (page 498)
- ARPluginEvent (page 499)
- ARPluginIdentify (page 499)
- ARPluginInitialization (page 500)
- ARPluginSetProperties (page 501)
- ARPluginTermination (page 502)
- AREA API (page 502)
- AREA API calls (page 505)
- AREAFreeCallback (page 505)

- AREANeedToSyncCallback (page 506)
- AREAVerifyLoginCallback (page 506)
- ARDBC API (page 508)
- ARDBC API calls (page 509)
- ARDBCCommitTransaction (page 509)
- ARDBCCreateEntry (page 510)
- ARDBCDeleteEntry (page 511)
- ARDBCGetEntry (page 513)
- ARDBCGetEntryBLOB (page 515)
- ARDBCGetEntryStatistics (page 516)
- ARDBCGetListEntryWithFields (page 519)
- ARDBCGetListSchemas (page 521)
- ARDBCGetMultipleFields (page 522)
- ARDBCRollbackTransaction (page 523)
- ARDBCSetEntry (page 524)
- ARF API calls (page 526)
- ARFilterApiCall (page 526)

# Overview

The AR System offers two ready to use plug-ins that you access through BMC Remedy Administrator: AREA Lightweight Directory Access Protocol (LDAP) plug-in and ARDBC LDAP plug-in. Both these plug-ins access LDAP services.

The AR System also supports three plug-ins and corresponding application program interfaces (APIs) that you create yourself.

---

**Note:** On Microsoft Windows platforms, plug-ins created previously (for pre 7.0 servers usage) must be recompiled with Microsoft Visual Studio .NET 2003 to be used successfully within the AR System 7.0 environment.

---

## LDAP plug-ins

These plug-ins enable you to extend the functionality of the AR System to external data sources. External data is data that is not contained in the AR System database. Both these plug-ins access LDAP services.

To use these plug-ins, you must:

- Install them when you install the AR System (or install them later by re-running the installation program). See the *Installing* guide.
- Use the AREA LDAP and ARDBC LDAP forms to configure them. See the *Integrating with Plug-ins and Third-Party Products* guide.
- Create forms and map fields. (ARDBC LDAP only). See the *Form and Application Objects* and *Workflow Objects* guides.

## Plug-ins that you create

You can create the following types of AR System plug-ins.

### [AR System External Authentication \(AREA\)](#)

Accesses network directory services and other authentication services to verify user login name and password. When you use the AREA plug-in, you do not have to maintain duplicate user authentication data in the AR System directories because the AR System server can access user identification information and user passwords from external sources.

**Note:** If, however, you have users with fixed licenses or users that use BMC Remedy applications, such as the IT Service Management applications, you must maintain user authentication data in AR System directories because users must exist in the user form for the application.

---

## AR System Database Connectivity (ARDBC)

Accesses external sources of data. You can integrate ARDBC with external data sources through their own APIs. The ARDBC plug-in, which you access through the vendor form, enables you to perform the following tasks on external data:

- Search
- Create, delete, modify, and set entries
- Populate search-style character menus
- Implement workflow

## AR System filter (ARF)

Offers an alternate method to send information requests to and from external servers. Use of ARF increases transaction speed and enables the AR System server to return to its workflow faster. ARF also applies to escalations.

The following example files, which can be used to create a filter API DLL or shared library, are located in the `<install_dir>/Arserver/Api /arfil terapi /example` directory:

- `arfil terapi samp. c`
- `arfil terapi samp. dep`
- `arfil terapi samp. vcproj`
- `arfil terapi samp. mak`

To create and use the AR System plug-ins, you must install plug-in related files and components, set up the environment, create the plug-ins, and start the plug-in service (also referred to as plug-in server or the `arpl ugi n` server) from the command line. The `arpl ugi n` server, a companion server to the AR System server, runs plug-in requests from the server and acts as a buffer between the client APIs and the server.

## Installing plug-in files and components

When you install the AR System, you automatically install the arpl ugi n server into the same directories that contain the AR System files. During the installation, you must select the option to install the API component of the AR System server, which includes the plug-in related APIs. When you install the API component, you also install the header files you use to compile and create the shared libraries. See the `arpl ugi n.h` file for plug-in definitions and declarations.

See the *Installing* guide for more installation information.

## Setting up the environment

This section describes how to set up your environment for use with plug-ins.

### ► To set up the plug-in environment

- 1 Set the plug-in related values in the `ar.conf` (UNIX) or `ar.cfg` (Windows) configuration file.

The `arpl ugi n` server reads the configuration file at run time and loads the plug-ins that the configuration file specifies.

- 2 Create a dynamic linked library (DLL) if you use a Windows platform, or a shared object library if you use a UNIX platform. Place the library in the directory that contains the AR System.

## Creating plug-ins

You use the AR System C API to extend *client* functionality. To use the C API, you write code that references the C API and a dynamically linked library (DLL) if you use a Windows platform, or a shared object library if you use a UNIX platform. The AR System provides the C API, header files, and libraries, which are installed into the same directory as other AR System files.

You use AR System plug-ins to extend *server* functionality. To use AR System plug-ins, you must create a DLL or shared object that conforms to the specifications in the header files. BMC Remedy provides files that contain empty function definitions that you can use to build your plug-in.

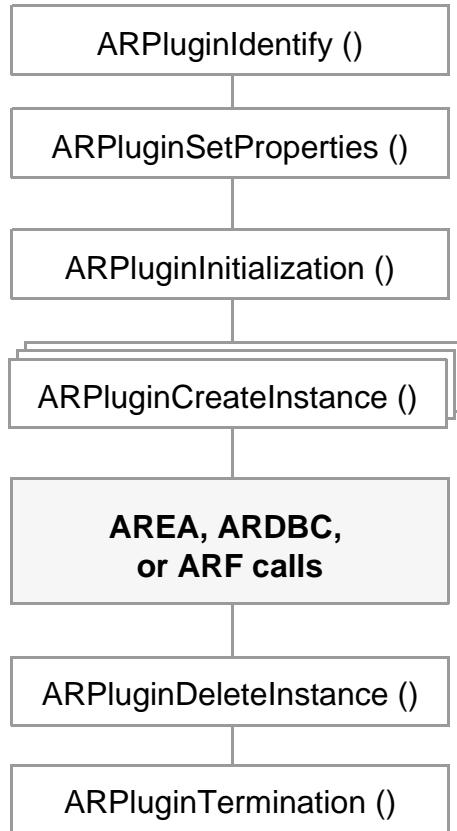
## ► To create plug-ins

- 1 Write a C or C++ program that includes:
  - Plug-in API calls for initialization, termination, object creation, and object destruction (see “Plug-in API calls” on page 497).
  - One of three type-specific API calls (see “AREA API” on page 502, “ARDBC API” on page 508, or “ARF API calls” on page 526).
  - Code to implement the calls— Use sample Microsoft Developer Studio projects (Windows) or makefiles (UNIX), which you install with the AR System, to build your program into your DLL or shared object library.
- 2 Place your plug-in in the directory that contains the other AR System files and add an entry to the configuration file that identifies the plug in, for example:
  - Windows—Add an entry in the ar.cfg file: <your\_plug\_in>.dll
  - UNIX—Add an entry in the ar.conf file: <your\_plug\_in>

At run time, the arplugin server reads the configuration file and loads the specified plug-ins.

The following figure shows the general structure of a plug-in program.

Figure 8-1: Plug-in program structure



- 3 Windows—Re-start the BMC Remedy Action Request System Server service.  
UNIX—Re-start the BMC Remedy AR System server by using the stop and start scripts.
- 4 Run the `arpl ugi n` or `arpl ugi n.exe` program from the command line (see “Running the arplugin server” on page 489). The `arpl ugi n` program reads the configuration file at run time and creates the plug-ins that the configuration file specifies.

You must create a separate plug-in for each of the three types of plug-ins. For example, you cannot create one plug-in that supports both AREA and ARDBC.

---

**Note:** If you do not add the plug-in information to the configuration file before you run `arpl ugi n`, the `arpl ugi n` server does not load any plug-ins. If you consequently modify the configuration file, you must restart the `arpl ugi n` server to create the plug-ins.

---

## Plug-in conventions

When you implement a plug-in, use the following naming conventions and memory management practices.

### Naming conventions

Plug-in names must be unique. Use the following naming format:

`<company_name>. <pl ug-i n_type>. <uni que_pl ugi n_i denti fi er_name>`

For example, if your company name is *ACME*, the type of plug-in you want to create is an *ARDBC* plug-in, and the unique plug-in identifier is *pl ugi nexampl e1*, your plug-in name could be:

ACME. ARDBC. pl ugi nexampl e1

Plug-in names cannot include space or tab characters. In addition, the value cannot contain the word AREA by itself, as this word is reserved for AREA plug-ins. However, you *can* use the word AREA as the `<pl ug-i nType>` value.

### Memory management

Do not free memory that the `arpl ugi n` server passes to your functions as arguments or that you return to the `arpl ugi n` server. Plug-ins can allocate and deallocate memory associated with the `obj ect` argument for each call. The `arpl ugi n` server will not deallocate this memory.

### Input values and return values

In a C API program, developers specify input values for the functions and receive return values. In a plug-in program, the `arpl ugi n` server provides the input values to the plug-ins and the plug-ins provide return values.

If the AR System server returns `AR_RETURN_WARN` or `AR_RETURN_OK` to the `arpl ugi n` log file after a call is issued, the `arpl ugi n` server considers that call to be successful. The `arpl ugi n` server considers the call to be unsuccessful if the server returns `AR_RETURN_ERROR` or `AR_RETURN_FATAL`.

If you do not implement a call, the arpl ugi n server will take a default action. The default action might be to proceed or return an error message.

## Protection for global information

You must protect any global information or resources that you access through plug-in related API calls with appropriate mutual exclusion locks to make sure of multithread safety. Global information and resource protection applies to all plug-in related calls except ARPI ugi nl denti fy, ARPI ugi nl ni ti al i zati on, ARPI ugi nSetProperti es, and ARPI ugi nTermination, which are always called by one thread at a time.

At run time, the arpl ugi n server reads the configuration file and creates the plug-ins that the configuration file specifies. Once the arpl ugi n server creates the plug-ins, they remain active until a system failure or until you modify the plug-in configuration information and restart the arpl ugi n server.

For more information about configuration, see the configuration file information in the *Configuring* guide.

## Running the arplugin server

To run the arpl ugi n server, type the following at the command prompt or include it in armonitor.conf or armonitor.cfg file:

**UNIX:** arpl ugi n [-i <directory>] [-s]

**Windows:** arpl ugi n [-i <directory>] [-m] [-s]

Option	Operating system	Explanation
-i <directory>	UNIX and Windows	Specifies path to the AR System server installation directory.
-m	Windows	Specifies manual running mode for the arpl ugi n server. The default mode is to run the arpl ugi n server as a Windows service.
-s	UNIX and Windows	Specifies the AR System server name.

## Accessing the plug-ins

After you have completed the preparatory steps, your plug-ins are ready to be accessed under the following circumstances:

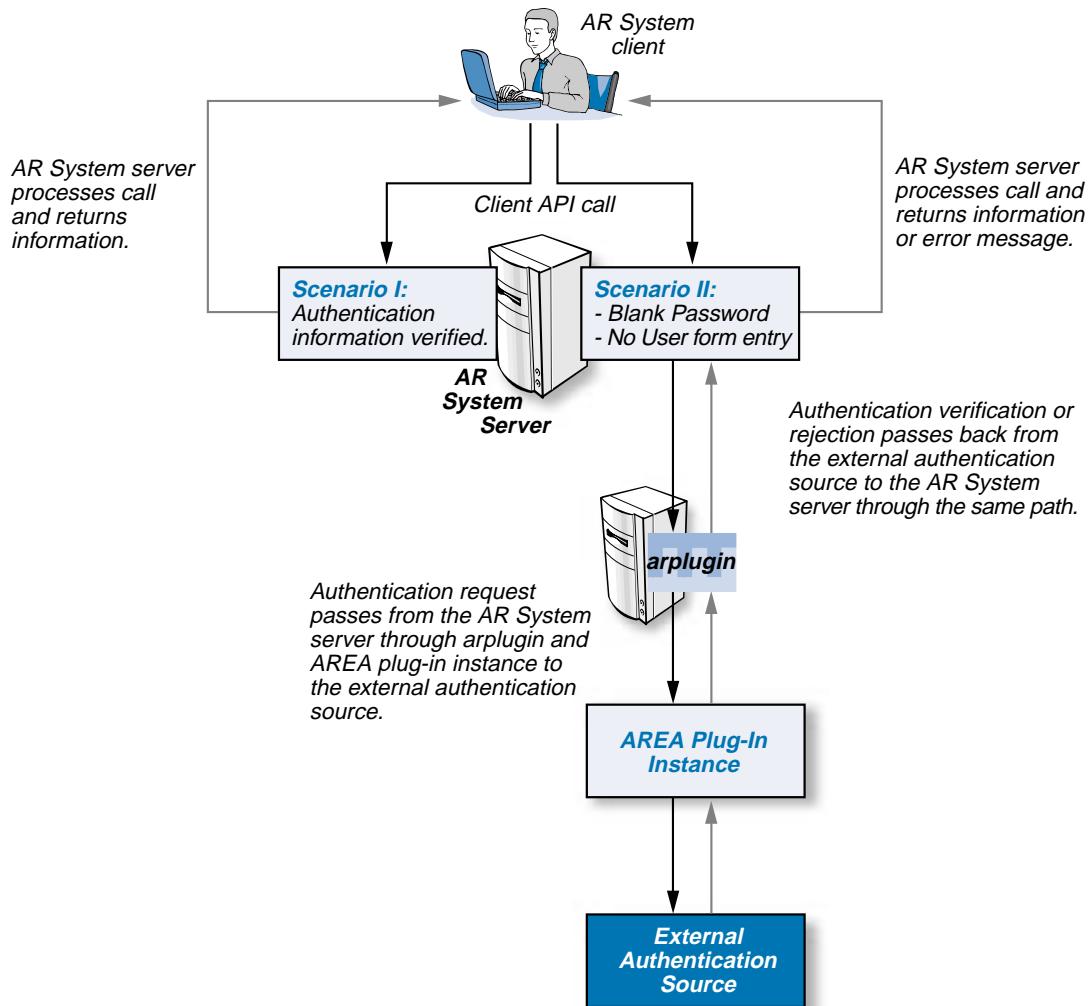
- ARDBC plug-ins are accessed when an API client or workflow operation references data stored in an external data source.
- ARF plug-ins are accessed when server-side workflow, such as filters and escalations, reference filter API calls.
- AREA plug-ins are accessed when the AR System server needs to authenticate the identity of a user.

The AR System server uses AREA plug-ins in a manner that is transparent to the user. Plug-ins connect to the AR System server through the `arpl_ugi_n` server.

## AREA plug-in

The following figure shows an example of the flow of requests and data for an AREA plug-in.

Figure 8-2: Flow of requests and data for an AREA plug-in



When users first log in to the AR System through a client, or when a client issues an API call to the AR System, the AR System server verifies the user name and password.

If the server verifies that the user name and password are in the User form, it authenticates the information and processes the login or API call.

When the user information is not in the User form or the user password is blank in the User form, the AR System server sends an authentication request to the arpl ugi n server. The request passes from the arpl ugi n server through the AREA plug-in instance to the external authentication source. The external authentication source sends authentication information back through the same path to the AR System server.

If the authentication source verifies that the user information is valid, the AR System server processes the API call or allows the user to log in.

When the authentication information is not verified (that is, the information is incorrect, incomplete, or cannot be found on the external data source), the AR System server returns an error message to the client.

The arpl ugi n server can only load one AREA plug-in instance at a time. An AREA plug-in can be configured to access one or more data sources.

---

**Note:** The behavior described is dependent on how you configure the plug-in, such as whether you enable the Cross Reference Blank Password and the Authenticate Unregistered users options.

---

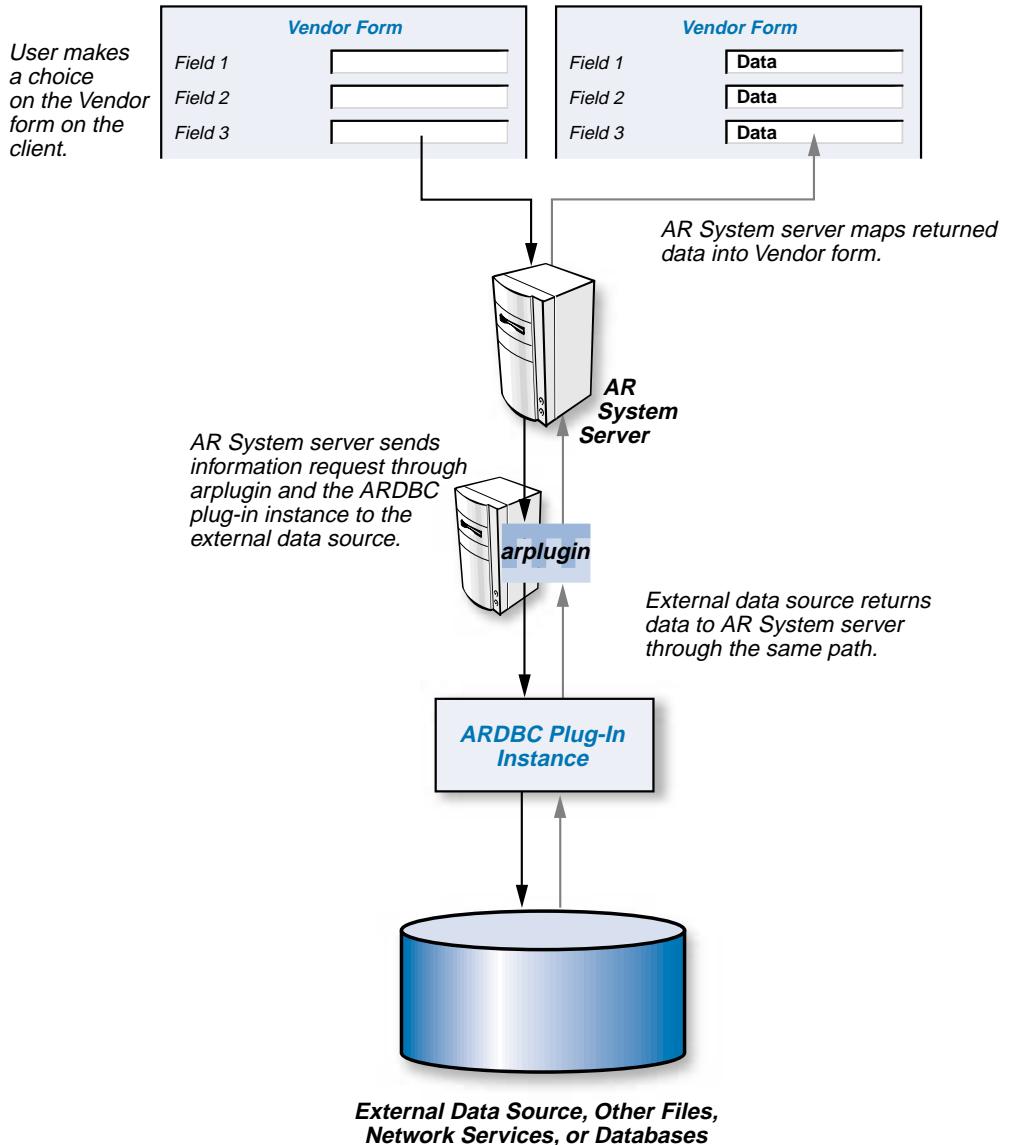
## ARDBC plug-in

When users enter requests in the vendor form on AR System API clients, such as BMC Remedy User or BMC Remedy Mid Tier, the AR System server sends the requests to the arpl ugi n server, which routes the requests to the ARDBC plug-in instance. The plug-in retrieves the data (if any) from the external data source and returns it in the opposite direction. The AR System server maps the external data to fields in the vendor form and the form displays the data. See the *Form and Application Objects* for more information about external forms.

The arpl ugi n server can load more than one ARDBC plug-in at a time.

The following figure explains the flow of requests and information for the ARDBC plug-in.

Figure 8-3: Flow of requests and information for the ARDBC plug-in



## ARF plug-in

You create AR filter (ARF) API set field actions in filters and escalations with BMC Remedy Administrator.

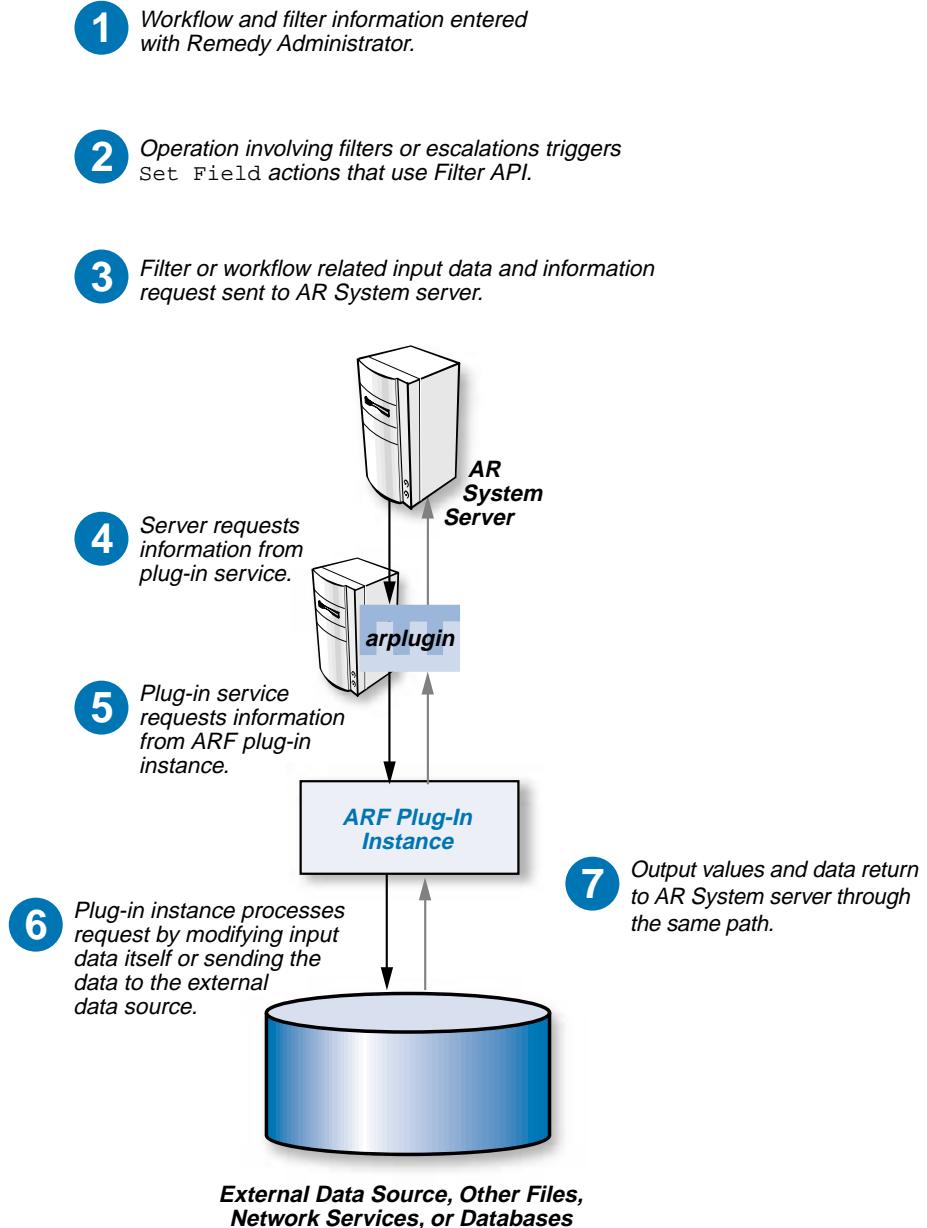
At run time, the AR System server sends AR filter API requests to the arpl ugi n server, which directs the requests to the appropriate plug-in. The plug-in processes the input arguments and can return values that can be used in the set fields action.

When you enter ARF API requests in the Create Filter form, the AR System server sends the requests to the arpl ugi n server, which sends them to ARF. ARF either processes the data or request itself or retrieves output data from the external data source and returns it in the opposite direction.

For information about example files that can be used to create a filter API DLL or shared library, see “AR System filter (ARF)” on page 484

This figure shows the flow of requests and information for the ARF plug-in.

Figure 8-4: Flow of requests and information for the ARF plug-in



The arplugin server can load more than one ARF plug-in at a time.

# Plug-in logging facility

The plug-in logging facility allows plug-ins to write information to the arpl ugi n server's log file. The plug-in is provided with a function to call via ARPI ugi nSetProperties.

Additionally, the plug-in can specify a logging level associated with each piece of information. This allows the plug-in to write different classes of information to the log file under different circumstances.

## Log function

```
typedef int (*AR_PLUGIN_LOG_FUNCTION)(  
    ARPI ugi nl denti fi cati on          *id,  
    int                                     logLevel,  
    char                                    *text);
```

### Input arguments

#### **id**

The plug-in type, name, and version.

#### **logLevel**

The log level to which the information applies. The text is not written to the log file unless the arpl ugi n log level is equal to or lower than logLevel.

#### **text**

The message that is written to the arplugin log file.

### Return values

There are no return values.

## Logging levels

The following is a list of the arplugin logging levels with recommended uses.

#### **AR\_PLUGIN\_LOG\_OFF (10000)**

No plug-in messages are logged.

#### **AR\_PLUGIN\_LOG\_SEVERE (1000)**

Messages that report fundamental problems that inhibit the plug-in from working. An example of this might be the inability to open a required resource during plug-in initialization.

## [AR\\_PLUGIN\\_LOG\\_WARNING \(900\)](#)

Messages that might be a precursor to a severe problem or identify an incorrect configuration setting. An example of this might be a bad configuration setting where the plug-in logs a warning and reverts to a default value.

## [AR\\_PLUGIN\\_LOG\\_INFO \(800\)](#)

Messages that identify intermittent milestones or events that do not have negative repercussions. This should not be used on information that is likely to occur frequently.

## [AR\\_PLUGIN\\_LOG\\_CONFIG \(700\)](#)

Messages that describe the current configuration settings of the plug-in.

## [AR\\_PLUGIN\\_LOG\\_FINE \(600\)](#)

This setting, along with the "finer" and "finest" logging levels, is primarily used while debugging problems. For every decision that is made throughout processing, identify the result of the decision.

## [AR\\_PLUGIN\\_LOG\\_FINER \(500\)](#)

Supplements "fine" messages with supporting data.

## [AR\\_PLUGIN\\_LOG\\_FINEST \(400\)](#)

Messages at this level typically contain information that would help someone who has the plug-in source code in front of them. All messages logged at the higher levels should have meaning for a customer or a technical support engineer who may not have access to the source code. This is where you would have messages that reference internal function names or structures.

## [AR\\_PLUGIN\\_LOG\\_ALL \(100\)](#)

All plug-in messages are logged.

# Plug-in API calls

Implement the following plug-in API calls in every type of plug-in.

## ARPluginCreateInstance

**Description** This optional call creates a plug-in instance and can also create instance-specific data. Simple plug-ins might not need to define this function because they do not need to create instance-specific data.

### Synopsis

```
#include "arpl_ugi.h"

int ARPI_ugi_nCreateInstance(
    void* object,
    ARStatusList* status)
```

**Return arguments**

#### object

Pointer to the plug-in instance that this call creates, if any.

#### status

A list of zero or more notes, warnings, or errors generated as a result of completing this call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARPluginDeleteInstance, ARPluginIdentify, ARPluginInitialization, ARPluginTermination.

## ARPluginDeleteInstance

**Description** When a thread encounters an exception, the arpl\_ugi\_n server uses this call to free resources associated with the instance. You do need to define this function unless you need to free resources associated with this instance.

### Synopsis

```
#include "arpl_ugi.h"

int ARPI_ugi_nDeleteInstance(
    void* object,
    ARStatusList* status)
```

**Return arguments**

#### object

Pointer to the plug-in instance to be deleted. This is the object that the server returned when it executed the ARPI\_ugi\_nCreateInstance call.

**status**

A list of zero or more notes, warnings, or errors generated as a result of completing this call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARPluginCreateInstance, *ARPluginIdentify*, ARPluginInitialization, *ARPluginTermination*.

## ARPluginEvent

**Description** This optional call is used to inform the plug-in of events that have occurred. The plug-in can use this to react to changes in configuration. You must protect any global information or resources accessed here with appropriate mutual exclusion locks to make sure of multithread safety.

**Synopsis**

```
#include "area.h"

int ARPluginEvent(
    ARPropList eventID,
    ARStatusList *status)
```

**Input arguments**

**eventID**

The type of event that has occurred:

AR\_PLUGIN\_EVENT\_CONFIG

The AR System server has informed the arplugin server of a change to the server configuration file.

**status**

A list of zero or more notes, warnings, or errors generated as a result of completing this call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## ARPluginIdentify

**Description** This call provides the type, name, and version of a plug-in. The arplugin server issues this call when it loads a plug-in.

---

**Important:** If you do not define this function, the `arpl ugi n` server does not load the plug-in.

---

**Synopsis**

```
#include "arpl ugi n. h"

int ARPI ugi ni denti fy(
    ARPI ugi ni denti fi cati on          *id,
    ARStatusLi st                         *status)
```

**Return arguments****id**

The plug-in type, name, and version.

**type**

The type of plug-in, in the form:

`AR_PLUGI N_TYPE_<pl ug-i nType>`

**name**

The unique name of the plug-in.

**version**

The plug-in version, which is appended to the plug-in type:

`AR_PLUGI N_<type>_VERSI ON`

**status**

A list of zero or more notes, warnings, or errors generated as a result of completing this call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** `ARPluginCreateInstance`, `ARPluginDeleteInstance`, `ARPluginInitialization`, `ARPluginTermination`.

## ARPluginInitialization

**Description** This optional call allocates and initializes global resources when the `arpl ugi n` server loads the plug-in.

**Synopsis**

```
#include "arpl ugi n. h"

int ARPI ugi ni ni ti al i zati on(
    ARStatusLi st                         *status)
```

---

<b>Return arguments</b>	<b>status</b> A list of zero or more notes, warnings, or errors generated as a result of completing this call. See “Error checking” on page 441 for a description of all possible values or see the <i>Error Messages</i> guide.
<b>See also</b>	<i>ARPluginCreateInstance</i> , <i>ARPluginDeleteInstance</i> , <i>ARPluginIdentify</i> , <i>ARPluginTermination</i> .

## ARPluginSetProperties

<b>Description</b>	This optional call provides information to the plug-in, such as plug-in service configuration parameters or services that the plug-in server provides to the plug-ins it loads.  <b>ARPluginSetProperties</b> is called immediately after <b>ARPluginIdentify</b> . You must protect any global information or resources accessed here with appropriate mutual exclusion locks to make sure of multithread safety.
--------------------	--

### Synopsis

```
#include "area.h"

int ARPluginSetProperties(
    ARPropList           *propList,
    ARStatusList         *status)
```

<b>Input arguments</b>
------------------------

#### propList

A list of **ARPropStruct** structures. The following properties are supported:

##### AR\_PLUGIN\_PROP\_LOG\_FUNCTION

An **ARByteList** containing a function pointer to the **arplugin** logging facility. The plug-in might use this function pointer to log plug-in specific information to the **arplugin** log file. See “Plug-in logging facility” on page 496.

### Return values

#### status

A list of zero or more notes, warnings, or errors generated as a result of completing this call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

### See also

The *Configuring* guide.

# ARPluginTermination

Description	This optional call deallocates global resources for a plug-in instance when the plug-in instance completes its function.
Synopsis	#include "arpl_ugi.h" <pre>int ARPluginTermination(     ARStatusList *status)</pre>
Return arguments	<b>status</b> A list of zero or more notes, warnings, or errors generated as a result of completing this call. See “Error checking” on page 441 for a description of all possible values or see the <i>Error Messages</i> guide.
See also	<i>ARPluginCreateInstance</i> , <i>ARPluginDeleteInstance</i> , <i>ARPluginIdentify</i> , <i>ARPluginInitialization</i> .

# AREA API

This section describes the AREA API calls and explains related information.

## AREA API data structure

The AREA API data structure returns user login information from the AREA plug-in to the AR System server through AREAResponseStruct structure. AREAResponseStruct contains information about login status (whether the login attempt succeeded, failed, or the user is unknown), license information for modifying entries and performing full text search (FTS), and message information that can be displayed to a user or logged to a file (aruser.log).

If you have configured the system to cross-reference a blank password, the AREAResponseStruct element that the AREA plug-in returns overrides the related value in the user entry on the User form. For example, if the plug-in returns a value for the groups element of AREAResponseStruct, then those groups will override the groups in the user entry on the User form.

The AREA plug-in cannot return a fixed license (AR\_LICENSE\_TYPE\_FIXED). Users must have a fixed license selected in their entry in the User form.

The AREAResponseStruct structure is defined in the area.h file.

An AREA plug-in implementation must use the data structure to return a response. The structure contains the following elements:

licenseMask	An integer value indicating the type of licenses for that user. To preserve license values in the User form, set the value of licenseMask to zero (0).
	AREA_LICENSE_MASK_WRITE Used for setting write license
	AREA_LICENSE_MASK_FULL_TEXT Setting for FTS license.
	AREA_LICENSE_MASK_RESERVED1 Reserved
	AREA_LICENSE_MASK_ALL All of these options (for convenience)
licenseWritel	An integer value indicating the type of license for that user.
	AR_LICENSE_TYPE_NONE Indicates no write license (read license)
	AR_LICENSE_TYPE_FLOATING Indicates floating write license
licenseFTS	An integer value indicating the FTS license type or that user.
	AR_LICENSE_TYPE_NONE Indicates no FTS license
	AR_LICENSE_TYPE_FLOATING Indicates floating FTS license
licenseRes1	An integer value indicating the type of licenses for that user. This is reserved.
licenseApps	A character string of the application licenses, separated by semicolons, to which the user is licensed.
groups	A character string of the group names, separated by semicolons, to which the user belongs. To preserve the group information in the User form, set the value of groups to NULL.
notifyMech	An integer value indicating the type of notification for that user. This can be either notification by email or through BMC Remedy Alert. To preserve the notify mechanism in the User form, set the value of notifyMech to zero (0).
	AR_NOTIFY_VIA_NONE Indicates no notifications will be sent
	AR_NOTIFY_VIA_NOTIFICATIONER Notifications sent through BMC Remedy Alert
	AR_NOTIFY_VIA_EMAIL Notifications sent through email
	AR_NOTIFY_VIA_DEFAULT Notifications sent according to setting for the user in the User form
email	The email address for the user. To preserve the email address in the User form, set the value of email to NULL.

---

loginStatus	Tells whether the login was successful, or whether the user is unknown.  AREA_LOGIN_SUCCESS Successful login (authenticated) AREA_LOGIN_UNKNOWN_USER User is not known to the system AREA_LOGIN_FAILED Failed to authenticate login
messageText	The message text that might appear to users. For example: "You failed to log in." or "Your password will expire in less than 30 days." If the login attempt fails, this will appear as an AR_RETURN_ERROR. On successful login, this will appear as AR_RETURN_OK, and the message will appear as a note, not an error.
logText	Messages that only an administrator would use; these would appear in the aruser.log file.
modTime	The time that the user's information was last changed. This is represented in the number of seconds from January 1, 1970 (UNIX time). If the time is 0 (zero), this communicates that the timestamp parameter has not changed from the last time that a time stamp was specified.

The licenseMask information is necessary, because the license information normally contained in the User form is overridden when you use external authentication. This is relevant in situations where some of your AR System users are listed in the User form, and some will be authenticated externally.

The AREA response data structure follows:

```
typedef struct AREAResponseStruct {
    unsigned int licenseMask; /* AREA_LICENSE_MASK_* */
    unsigned int licenseWrite; /* AR_LICENSE_TYPE_* from AR API */
    unsigned int licenseFTS; /* AR_LICENSE_TYPE_* from AR API */
    unsigned int licenseRes1; /* AR_LICENSE_TYPE_* from AR API */
    char *licenseApps; /* AR_LICENSE_TYPE_* from AR API */
    char *groups; /* semi-colon separated list */
    unsigned int notifyMech; /* AR_NOTIFY_* from AR API */
    char *email;
    unsigned int loginStatus; /* AREA_LOGIN_* */
    char *messageText; /* text seen by user */
    char *logText; /* text placed in user log */
    ARTimestamp *modTime;
} AREAResponseStruct;
```

---

**Note:** If you use AREA plug-ins, the plug-ins can selectively override field values that users entered in the User form.

---

# AREA API calls

This section contains AREA API calls.

## AREAFreeCallback

**Description** This optional call frees the data associated with the AREAResponseStruct, which the AREAVerifyLoginCallback call returns. Although the API plugin server normally frees the storage that other API calls return, it does not free the storage that this API call returns. Instead, the plugin deallocates the storage through this call, allowing the plugin to optionally cache information.

The API plugin server calls this function after it calls the AREAVerifyLoginCallback call. The AREAVerifyLoginCallback call returns a pointer to the response structure as a parameter.

You must protect any global information or resources accessed here with appropriate mutual exclusion locks to make sure of multithread safety. An AREA plugin implementation must return a response. A NULL response indicates a failed login attempt.

---

**Important:** You must free whichever storage is allocated for the purpose of this response only. For example, if you are getting the response data from a prebuilt cache constructed during initialization, then you would not free anything here. If you are not getting the data from a prebuilt cache, then you would want to deallocate response data here.

---

### Synopsis

```
#include "area.h"

int AREAFreeCallback(
    void* object,
    AREAResponseStruct* response)
```

### Input arguments

#### object

Pointer to an instance of an AREA plugin that the API plugin instance call returns.

## response

A pointer to the response structure that the AREAVerifyLoginCallback call returns.

See also [AREAVerifyLoginCallback](#).

# AREANeedToSyncCallback

<b>Description</b>	This call determines whether the user information in the server cache is invalid. The AR System server periodically checks with the AREA plug-in to determine if any in-memory copies of user information have expired. The arpl_ugi_n server issues this call if it receives an authentication request from the server at least five minutes since the last authentication request.  You must protect any global information or resources accessed here with appropriate mutual exclusion locks to make sure of multithread safety. A return value that is not zero will instruct the AR System server to expire its internally stored user information.
--------------------	---

## Synopsis

```
#include "area.h"

int AREANeedToSyncCallback(
    void* object)
```

## Input arguments

### object

Pointer to an instance of an AREA plug-in that the ARPI\_ugi\_nCreateInstance call returns.

# AREAVerifyLoginCallback

<b>Description</b>	The arpl_ugi_n server issues this call when the AR System server makes a request to authenticate a user. The AR System server passes the unencrypted user name and password as parameters. You must protect any global information or resources accessed here with appropriate mutual exclusion locks to make sure of multithread safety.  An AREA plug-in implementation must return a response. A NULL response indicates a failed login attempt.  The AR System server detects the IP address of the client and sends it to the plug-in, which rejects or accepts the requests.
--------------------	--

**Synopsis**

```
#include "area.h"

int AREAVerifyLoginCallback(
    void* object,
    ARAccessNameType user,
    ARPasswordType password,
    ARAccessNameType networkAddr,
    ARAuthType authString,
    AREAResponseStruct** response)
```

**Input arguments****object**

Pointer to an instance of an AREA plug-in that the `ARPI_ugi_nCreateInstance` call returns.

**user**

User name.

**password**

User password. If the password is a NULL pointer, as opposed to an empty string, then the AR System is attempting to send the user a notification and needs to determine the user's notification mechanism and email address. When the password is NULL, the AR System is not requesting authentication of the user's identity. The plug-in implementation should return notification mechanism and email address information for the user and return a successful login attempt.

**networkAddr**

The internet provider (IP) address of the AR System client. The AR System server determines the client's IP address and passes the address, user name, and password to the plug-in.

For connections using AR System external authentication (AREA), the IP address of the client machine can be used as authentication, even when using a firewall or a load balancer. However, you cannot use the IP address as authentication when making mid tier connections. In this case, the IP address passed is that of the machine where the mid tier is installed instead of the client machine.

**authString**

A string that the AR System API client user enters into the login screen. There are no format restrictions for this string. The AREA plug-in uses this string for authentication purposes.

Return values [response](#)

A pointer to a response structure containing user information returned from the plug-in. The structure should be freed using `AREAFreeCallback` when no longer needed.

See also [AREAFreeCallback](#).

## ARDBC API

The ARDBC API enables you to:

- Implement calls on external data that are analogous to set entry, get entry, create entry, delete entry, and get list C API calls.
- Use external data to implement push field and set field filter, escalation, and active link actions.
- Create, modify, and search for external data through API clients, such as BMC Remedy User.
- Construct query-style character menus.
- Present forms, views, and active links on external data in the same manner as internal data. The data source is transparent to the user.

---

**Note:** If you implement an ARDBC plug-in, document what your plug-ins allow users to do. For example, you might implement a read-only plug-in, which does not allow a user to create, set, or delete entries. If the data source with which you integrate does not support a particular functionality, do not implement that function and, instead, allow the default behavior to occur. For example, if your data source does not support transactions, do not define `ARDBCCommitTransaction` and `ARDBCRollbackTransaction` calls.

---

## ARDBC API data structure

The ARDBC API uses the same data structures as the C API, in addition to the `ARVendorFieldStruct` structure. The `ARVendorFieldStruct` structure is defined in the `ardbc.h` header file.

# ARDBC API calls

ARDBC has the following API calls.

## ARDBCCCommitTransaction

<b>Description</b>	This optional call commits the changes of one or more previous ARDBC calls to the external data source. The client typically calls this function after it performs one or more actions that read or modify the external data source. After the client issues this call, the AR System server begins the next transaction.
--------------------	---

---

**Important:** If you do not define this function and the `arpl ugi n` server receives a *commit transaction* request, the call proceeds without error.

---

### Synopsis

```
#include "ardbc.h"

int ARDBCCCommitTransaction(
    void* object,
    ARInternalId translId,
    ARStatusList* status)
```

### Input arguments

**object**  
Pointer to an instance of an ARDBC plug-in that the `ARPI ugi nCreateInstance` call returns.

### transId

The transaction identification number (transaction ID) that the `arpl ugi n` server designates for each transaction. The transaction ID is unique to the thread that the AR System server uses to access the ARDBC plug-in. There is only one open transaction per server thread.

### Return values

### status

A list of zero or more notes, warnings, or errors generated as a result of completing this call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

### See also

`ARDBCRollbackTransaction`.

# ARDBCCreateEntry

**Description** Creates an entry in the external data source. The arplugin server issues this call upon request from the AR System server.

The AR System server issues a request to the arplugin server when:

- An AR System API client issues an ARCreateEntry or an ARMergeEntry call to the server.
- Users enter information to create or merge entries into an external form.
- Users import entries into an external form.
- The Distributed Server Option (DSO) transfers or merges entries into an external form.
- A push fields filter or an escalation action creates a new entry in an external form.

---

**Important:** If you do not define this function, this call does not create an entry and the AR System server will receive an error message from the arplugin server.

---

## Synopsis

```
#include "ardbc.h"

int ARDBCCreateEntry(
    void *object,
    char *tableName,
    ARVendorFieldList *vendorFieldList,
    ARInternalId *internalId,
    ARFieldValuelist *fieldValuelist,
    AREntryIdList *entryIdList,
    ARStatusList *status)
```

## Input arguments

### object

Pointer to an instance of an ARDBC plug-in that the ARPluginCreateInstance call returns.

### tableName

The name of the external table on which the plug-in creates an entry.

## vendorFieldList

A list of external form fields. The plug-in uses this list to map AR System field identification numbers to corresponding external form field names. All other input arguments and return values must use the AR System field IDs.

## transId

The transaction identification number (transaction ID) that the arplugin server designates for each transaction. The transaction ID is unique to the thread that the AR System server uses to access the ARDBC plug-in. There is only one open transaction per server thread.

## fieldValueList

A list of the data in a new entry. The list consists of one or more field and value pairs that the server specifies in any order.

## Return values

### entryId

The unique identifier for the entry that this call creates. This must correspond to a value in the external data source and must be unique, non-NULL, and either character or numeric data. In the case of an external data source, the entry ID can be longer than 15 characters. Therefore, the entry ID can consist of one or more values of type AREEntryIdType and is represented by the AREEntryIdList structure.

### status

A list of zero or more notes, warnings, or errors generated as a result of completing this call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

See also [ARDBCDeleteEntry](#), [ARDBCCGetEntry](#), [ARDBCCGetEntryBLOB](#), [ARDBCSGetEntry](#).

# ARDBCDeleteEntry

**Description** Deletes a single entry in an external data source. The arplugin server issues an ARDBCDeleteEntry call when it receives a delete entry request from the AR System server.

The AR System server issues a delete entry request to the arplugin server when:

- An AR System API client issues an ARDeleteEntry call to the server.
- Users delete entries in an external form.
- The server receives a delete entry command from a workflow process to delete an entry.

---

**Important:** If you do not define this function, the AR System server will receive an error message and will not delete the entry.

---

## Synopsis

```
#include "ardbc.h"

int ARDBCDeleteEntry(
    void* object,
    const char* tableName,
    const ARVendorFieldList* vendorFieldList,
    const ARInternalId* transId,
    const AREntryIdList* entryId,
    const ARStatusList* status)
```

### Input arguments

#### object

Pointer to an instance of an ARDBC plug-in that the ARPluginCreateInstance call returns.

#### tableName

The name of the external table on which the plug-in deletes this entry.

#### vendorFieldList

A list of external form fields. The plug-in uses this list to map AR System field identification numbers to corresponding external form field names. All other input arguments and return values must see the AR System field IDs.

#### transId

The transaction identification number (transaction ID) that the arplugin server designates for each transaction. The transaction ID is unique to the thread that the AR System server uses to access the ARDBC plug-in. There is only one open transaction per server thread.

## entryId

The unique identifier for the entry that this call deletes. This must correspond to a value in the external data source and must be unique, non-NUL, and either character or numeric data. In the case of an external data source, the entry ID can be longer than 15 characters. Therefore, the entry ID can consist of one or more values of type AREEntryIdType and is represented by the AREEntryIdList structure.

### Return values

#### [status](#)

A list of zero or more notes, warnings, or errors generated as a result of completing this call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

### See also

[ARDBCCreateEntry](#), [ARDBCGetEntry](#), [ARDBCGetEntryBLOB](#), [ARDBCSetsEntry](#).

## ARDBCGetEntry

**Description** Retrieves a single entry from an external data source table. The arplugin server issues this call when the AR System server receives an ARGetEntry request from a client.

### Synopsis

```
#include "ardbc.h"

int ARDBCGetEntry(
    void* object,
    char* tableName,
    ARVendorIdList* vendorIdList,
    ARIinternalIdList* entryIdList,
    ARIinternalIdList* idList,
    ARFieldValuelist* fieldList,
    ARStatusList* status)
```

### Input arguments

#### [object](#)

Pointer to the instance of an ARDBC plug-in that the ARPluginCreateInstance call returns.

**tableName**

The name of the external table from which the plug-in retrieves this entry.

**vendorFieldList**

A list of external form fields. The plug-in uses this list to map AR System field identification numbers to corresponding external form field names. All other input arguments and return values must see the AR System field IDs.

**transId**

The transaction identification number (transaction ID) that the arplugin server designates for each transaction. The transaction ID is unique to the thread that the AR System server uses to access the ARDBC plug-in. There is only one open transaction per server thread.

**entryId**

The unique identifier for the entry that this call retrieves. This must correspond to a value in the external data source and must be unique, non-NULL, and either character or numeric data. In the case of an external data source, the entry ID can be longer than 15 characters. Therefore, the entry ID can consist of one or more values of type AREEntryIdType and is represented by the AREEntryIdList structure.

**idList**

A list of zero or more AR System field identification numbers (IDs) for the fields that this call retrieves. If the idList value is NULL or there are zero field ID's in the list, the call retrieves all the fields.

**Return values****fieldList**

A list of the data in the entry that this call retrieves. The list consists of one or more field and value pairs that the AR System server specifies in any order.

**status**

A list of zero or more notes, warnings, or errors generated as a result of completing this call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also**

ARDBCCreateEntry, ARDBCDeleteEntry, ARDBCGetEntryBLOB, ARDBCSetEntry.

# ARDBCGetEntryBLOB

<b>Description</b>	This call opens an attachment. The AR System server receives an ARGetEntryBLOB API call from the client when users request to open an attachment from an external form. The server then sends a request to the arplugin server, which issues an ARDBCGetEntryBLOB call.
--------------------	---

**Important:** If you do not define this function and the arplugin server receives a *get entry BLOB* request, the AR System server will receive an error message and will not retrieve the data.

## Synopsis

```
#include "ardbc.h"

int ARDBCGetEntryBLOB(
    void* object,
    const char* tableName,
    const ARVendorFieldList* vendorFieldList,
    const ARInternalId* transId,
    const AREntryIdList* entryIdList,
    const ARInternalId* fieldIdList,
    const ARLocStruct* loc,
    const ARStatusList* status)
```

## Input arguments

### object

Pointer to an instance of an ARDBC plug-in that the ARPluginCreateInstance call returns.

### tableName

The name of the external table that contains the attachment to be opened.

### vendorFieldList

A list of external form fields. The plug-in uses this list to map AR System field identification numbers to corresponding external form field names. All other input arguments and return values must see the AR System field IDs.

### transId

The transaction identification number (transaction ID) that the arplugin server designates for each transaction. The transaction ID is unique to the thread that the AR System server uses to access the ARDBC plug-in. There is only one open transaction per server thread.

### entryId

The unique identifier for the entry that this call retrieves. This must correspond to a value in the external data source and must be unique, non-NULL, and either character or numeric data. In the case of an external data source, the entry ID can be longer than 15 characters. Therefore, the entry ID can consist of one or more values of type AREEntryIdType and is represented by the AREEntryIdList structure.

### fieldId

The field ID for the field that this call retrieves.

#### Return values

##### loc

A pointer to an ARLocStruct structure that contains the name of the buffer.

##### status

A list of zero or more notes, warnings, or errors generated as a result of completing this call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

#### See also

ARDBCCreateEntry, ARDBCDeleteEntry, ARDBCGetEntry, ARDBCSetsEntry.

## ARDBCGetEntryStatistics

#### Description

This call gathers statistical information about a set of data. The server issues this call when it receives a request from the AR System server to get entry statistics.

---

**Important:** If you do not define this function and the server receives a *get entry statistics* request, the AR System server will receive an error message and will not retrieve the data.

---

#### Synopsis

```
#include "ardbc.h"

int ARDBCGetEntryStatistics(
    void* object,          *object,
    char* tableName,        *tableName,
    ARVendorFieldList* vendorFieldList,
    ARInternalId translid,
```

ARQualifierStruct	*qualifier,
ARFieldValueOrArrayListStruct	*target,
unsigned int	statistic,
ARIinternalList	*groupByList,
ARStatisticsResultList	*results,
ARStatusList	*status)

**Input arguments****object**

Pointer to an instance of an ARDBC plug-in that the `ARPIugiCreateInstance` call returns.

**tableName**

The name of the external table from which to retrieve the statistics.

**vendorFieldList**

A list of external form fields. The plug-in uses this list to map AR System field identification (field ID) numbers to corresponding external form field names.

---

**Note:** All other ARDBC input arguments and return values see AR System field IDs, not external form IDs.

---

**transId**

The transaction identification number (transaction ID) that the application server designates for each transaction. The transaction ID is unique to the thread that the AR System server uses to access the ARDBC plug-in. There is only one open transaction per server thread.

**qualifier**

A query that determines the set of entries to retrieve. The qualification can include one or more fields and any combination of conditional, relational, and arithmetic operations (numeric data only).

## target

The arithmetic operation that defines the statistic to compute. The statistic can include one or more fields and any combination of arithmetic operations. The system generates an error if the user does not have read permission for a field or a field does not exist. If you specify AR\_STAT\_OP\_COUNT for the statistic parameter, assign a tag value of 0 to omit this parameter.

## statistic

A value indicating the statistic type.

- 1: The total number of matching entries (AR\_STAT\_OP\_COUNT).
- 2: The sum of values for each group (AR\_STAT\_OP\_SUM).
- 3: The average value for each group (AR\_STAT\_OP\_AVERAGE).
- 4: The minimum value for each group (AR\_STAT\_OP\_MINIMUM).
- 5: The maximum value for each group (AR\_STAT\_OP\_MAXIMUM).

## groupByList

A list of zero or more fields to group the results by. The system computes a result for each group of entries having the same value in the specified field. Specifying more than one field creates groups within groups, each of which returns a separate statistic. Specify `NULL` for this parameter (or zero fields) to compute a single result for all matching entries.

## Return values

### results

A list of zero or more results. If you specify one or more fields for the groupByList parameter, each item in the list represents a group. Each result structure contains the field values that define the group and the statistic for that group.

### status

A list of zero or more notes, warnings, or errors generated as a result of completing this call.

See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## See also

ARDBCGetEntry.

# ARDBCGetListEntryWithFields

**Description** Retrieves a list of qualified entries with field data. The application server issues this call when the AR System server receives an ARGetListEntryWithFields or an ARGetListEntry request from a client.

---

**Important:** If you do not define this function and the application server receives a *get list entry with fields* request, the AR System server returns a list of zero entries.

---

## Synopsis

```
#include "ardbc.h"

int ARDBCGetListEntryWithFields(
    void* object,
    const char* tableName,
    const ARVendorFieldList* vendorFieldList,
    const ARInternalId* internalId,
    const ARQualifierStruct* qualifier,
    const ARSortList* sortList,
    const AREntryListFieldList* entryListFields,
    unsigned int firstRetrieved,
    unsigned int maxRetries,
    const AREntryListFieldValuelist* entryList,
    unsigned int numMatches,
    ArStatusList* status)
```

## Input arguments

**object**  
Pointer to an instance of an ARDBC plug-in that the ARPI::CreateInstance call returns.

### tableName

The name of the external table from which the plug-in retrieves the entry.

### vendorFieldList

A list of external form fields. The plug-in uses this list to map AR System field identification numbers to corresponding external form field names. All other input arguments and return values must see the AR System field IDs.

### transId

The transaction identification number (transaction ID) that the AR System server designates for each transaction. The transaction ID is unique to the thread that the AR System server uses to access the ARDBC plug-in. There is only one open transaction per server thread.

### qualifier

A query that determines the set of entries that this call retrieves. Values for this argument might include one or more fields, and any combination of conditional, relational, and arithmetic operations (for numeric data types only).

### sortList

A list of zero or more fields that identifies the order in which the call sorts the entries.

### getListFields

A list of zero or more fields that the call retrieves with each entry.

### firstRetrieved

The first entry to be retrieved. The call ignores entries that occur before the entry that this argument specifies.

### maxRetrieve

The maximum number of entries to retrieve. You can configure this option at both the AR System client and AR System server levels. The value that the AR System server passes to this call is the lower of the two values.

## Return values

### entryList

A list of zero or more retrieved entries.

### numMatches

The total number of entries that match the criteria in the qualifier argument. This value is the same as the number of entries that the call returns if the number of matching entries is less than or equal to the value in the maxRetrieve argument.

### status

A list of zero or more notes, warnings, or errors generated as a result of completing this call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

See also ARDBCGetListSchemas.

## ARDBCGetListSchemas

### Description

The call retrieves a list of external schemas. The `arpl ugi n server` issues this call when the AR System server receives an `ARGetListExtSchemaCandidates` request from the client to list candidate schemas.

To test your ARDBC plug-in in a vendor form, you must implement `ARDBCGetListSchemas`. If you do not implement this function, the Available Window Names and Available Vendor Tables fields in the Vendor form will not contain the name of your plug-in.

---

**Important:** If you do not define this function and the `arpl ugi n server` receives a *get list schemas* request, the AR System server returns a list of zero forms.

---

### Synopsis

```
#include "ardbc.h"

int ARDBCGetListSchemas(
    void* object,           /*obj ect,
    ARCompoundSchemaList* schema,   /*schema,
    ARStatusList* status);      /*status)
```

### Input arguments

#### object

Pointer to an instance of an ARDBC plug-in that the `ARPluginCreateInstance` call returns.

### Return values

#### schema

A list of retrieved schemas.

The name specified in `schema->compoundSchema[n].u.vendor.vendorName` must match the plug-in name specified in `ARPluginIdentify` in the id-name field.

#### status

A list of zero or more notes, warnings, or errors generated as a result of completing this call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

### See also

`ARDBCGetListEntryWithFields`, `ARDBCGetMultipleFields`.

# ARDBCGetMultipleFields

**Description** Retrieves a list of fields from an external data source. The plug-in issues this call when the AR System server receives an

ARGetMultipleFields API call from a client. For example, an BMC Remedy Administrator client might provide a user with a list of fields instead of requiring the user to type in a field name.

To test your ARDBC plug-in in a vendor form, you must implement ARDBCGetMultipleFields. If you do not implement this function, the Available Column fields in the Vendor form will not contain the name of your plug-in.

---

**Important:** If you do not define this function and the application server receives a *get multiple fields* request, the AR System server returns a list of zero fields.

---

## Synopsis

```
#include "ardbc.h"

int ARDBCGetMultipleFields(
    void* object,
    ARCompoundSchema* schema,
    ARFieldMappingList* mapping,
    ARFieldType* dataType,
    ARStatusList* status)
```

## Input arguments

### object

Pointer to an instance of an ARDBC plug-in that the ARPI CreateInstance call returns.

### schema

An ARCompoundSchema structure that describes all external forms.

## Return values

### mapping

A list of zero or more ARFieldMapping structures that describe external form fields.

## limit

A list of zero or more ARFieldList structures, each of which corresponds to the fields that the AR System server returns.

## dataType

A list of zero or more integers that represent the data types of the retrieved fields. See the ar.h file for the integer definitions, such as the AR\_DATA\_TYPE\_CHAR definition.

## status

A list of zero or more notes, warnings, or errors generated as a result of completing this call.

See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARDBCGetEntry, ARDBCGetListEntryWithFields, ARDBCGetListSchemas.

# ARDBCRollbackTransaction

**Description** This call rolls back the changes of one or more previous ARDBC calls to the external data source. The arplug-in server issues this call when one or more actions that read or modify the external data source result in a failed operation or an error. After the arplug-in server issues this function, the AR System server begins the next transaction.

---

**Important:** If you do not define this function and the arplug-in server receives a *roll back transaction* request, the AR System server does not process the request and does not return an error.

---

## Synopsis

```
#include "ardbc.h"

int ARDBCRollbackTransaction(
    void* object,           /*obj ect,
    ARInternalId translid, /*transl d,
    ARStatusList* status); /*status)
```

**Input arguments**

## object

Pointer to an instance of an ARDBC plug-in that the ARPlug-inCreateInstance call returns.

**transId**

The transaction identification number (transaction ID) that the arpl ugi n server designates for each transaction. The transaction ID is unique to the thread that the AR System server uses to access the ARDBC plug-in. There is only one open transaction per server thread.

**Return values****status**

A list of zero or more notes, warnings, or errors generated as a result of completing this call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARDBCCCommitTransaction.

## ARDBCSetEntry

**Description** This call modifies the contents of a single entry or BLOB. The arplugin server issues this call when:

- The AR System server receives an ARSetEntry or ARMergeEntry API call from the client.
- Users enter information to create or merge entries into an external form.
- A push fields filter or an escalation action modifies an entry.

If the specified entry does not exist, the plug-in creates it. An ARDBCSetEntry operation on a non-existent entry can occur when the AR System server receives an ARMergeEntry API call from a client.

---

**Important:** If you do not define this function, the AR System server will receive an error message and this call does not set the entry.

---

**Synopsis**

```
#include "ardbc.h"

int ARDBCSetEntry(
    void *object,
    *tabl eName,
    *vendorFi el dLi st,
    transId,
    *entryId,
    *fi el dVal ueLi st,
    *status)
```

Input arguments	<b>object</b>
	Pointer to the instance of an ARDBC plug-in that the ARPluginCreateInstance call returns.
	<b>tableName</b>
	The name of the external table on which the plug-in sets this entry.
	<b>vendorFieldList</b>
	A list of external form fields.
	The plug-in uses this list to map AR System field identification numbers to corresponding external form field names. All other input arguments and return values must see the AR System field IDs.
	<b>transId</b>
	The transaction identification number (transaction ID) that the arplugin server designates for each transaction.
	The transaction ID is unique to the thread that the AR System server uses to access the ARDBC plug-in. There is only one open transaction per server thread.
	<b>entryId</b>
	The unique identifier for the entry that this call sets. This must correspond to a value in the external data source and must be unique, non-NULL, and either character or numeric data. In the case of an external data source, the entry ID can be longer than 15 characters. Therefore, the entry ID can consist of one or more values of type AREntryIdType and is represented by the AREntryIdList structure.
	<b>fieldValueList</b>
	A list of the data in a new entry. The list consists of one or more field and value pairs that the server specifies in any order.
Return values	<b>status</b>
	A list of zero or more notes, warnings, or errors generated as a result of completing this call. See “Error checking” on page 441 for a description of all possible values or see the <i>Error Messages</i> guide.
See also	ARDBCCreateEntry, ARDBCDeleteEntry, ARDBCGetEntry, ARDBCGetEntryBLOB.

# ARF API calls

This section describes the ARF API calls and explains related information.

As the ARF plug-in creator, you must document the input and output values that the plug-in receives, to allow application developers to define appropriate filter and escalation values.

## ARF API data structure

The ARF API uses the same data structures that the C API uses.

### ARFilterApiCall

**Description** The ARPLugin server calls this function when it receives filter API set filters actions data or input values from the AR System server during workflow operations.

You must protect any global information or resources accessed here with appropriate mutual exclusion locks to make sure of multithread safety. You must include this call in all filter plug-ins.

#### Synopsis

```
#include "arfilterapi.h"

int ARFilterApiCall (
    void* object,
    ARValueList* inValues,
    ARValueList* outValues,
    *status)
```

#### Input arguments

##### object

Pointer to the plug-in instance that the ARPLuginCreateInstance call returns.

##### inValues

An array of input values that the filter action specifies. BMC Remedy Administrator defines the filter action and what data the AR System server passes to the ARF plug-in.

#### Return arguments

##### outValues

An array of output values that the plug-in creator defines.

## status

A list of zero or more notes, warnings, or errors generated as a result of completing this call. See “Error checking” on page 441 for a3 description of all possible values or see the *Error Messages* guide. The AR System server passes the status results on to the corresponding filter or escalation.

### See also

[ARPluginCreateInstance](#), [ARPluginDeleteInstance](#), [ARPluginIdentify](#),  
[ARPluginInitialization](#), [ARPluginTermination](#).



Chapter

# 9

# XML support

The AR System API provides routines to transform AR System objects into XML format and XML back into AR System objects. This chapter contains descriptions of the transformation routines and XML Schemas that describe the XML format of the AR System objects.

The following topics are provided:

- Transforming XML and AR System objects (page 530)
- Object API calls (page 531)
- Function descriptions (page 535)
- ARGetActiveLinkFromXML (page 536)
- ARGetContainerFromXML (page 539)
- ARGetDSOMappingFromXML (page 542)
- ARGetDSOPoolFromXML (page 545)
- ARGetEscalationFromXML (page 547)
- ARGetFieldFromXML (page 550)
- ARGetFilterFromXML (page 553)
- ARGetListXMLObjects (page 556)
- ARGetMenuFromXML (page 557)
- ARGetSchemaFromXML (page 559)
- ARGetVUIFromXML (page 563)
- ARParseXMLDocument (page 564)
- ARSetActiveLinkToXML (page 565)

- ARSetContainerToXML (page 569)
- ARSetDSOMappingToXML (page 572)
- ARSetDSOPoolToXML (page 575)
- ARSetEscalationToXML (page 577)
- ARSetFieldToXML (page 580)
- ARSetFilterToXML (page 583)
- ARSetMenuToXML (page 586)
- ARSetSchemaToXML (page 588)
- ARSetVUIToXML (page 592)
- ARSetXMLDocFooterToXML (page 593)
- ARSetXMLDocHeaderToXML (page 594)

## Transforming XML and AR System objects

To transform XML Schema into AR System objects and vice versa, you use the XML Schema definition files (.xsd files) and the XML API calls. The AR System XML Schema definition files describe the format and rules for defining the objects.

You can also use BMC Remedy Administrator to import and export AR System objects in XML from an AR System server. To make sure that you can import and export your own XML files and objects with AR System, check the AR XML Schema definition files for format information. For more information about exporting and importing, see the *Form and Application Objects* guide.

The next section lists the XML Schema definition files. The section “Object API calls” on page 531 lists the object API calls. For individual AR System XML tag descriptions, see the XML definition files.

## Schema definition files

XML Schemas are documents that are used to define and validate the content and structure of XML data.

An XML Schema defines and describes an XML instance document using the XML Schema definition language (XSD). An XML instance document conforms to the standards of the XML Schema. XML Schema elements (elements, attributes, types, and groups) are used to define the structure, content, and relationship of the XML data. XML Schemas can also provide default values for attributes, and elements. For more information, see the World Wide Web Consortium (W3C) website:

<http://www.w3.org/>

The AR System XML Schemas describe the format of the XML that is produced and consumed by AR System. Applications can be built to integrate with the AR System using XML that conforms to the standards of the AR System XML Schemas.

The XML schemas can also be used to validate the XML instance documents. This verifies that all of the elements of data exist, are in the expected sequence, and are all of the correct data type. This helps make sure that AR System will be able to interpret the XML instance document that is received.

## Object API calls

The object API calls are divided into two categories:

- ARGet calls, which transform XML objects into AR System structures.
- ARSet calls, which transform AR System structures into XML objects.

### Object manipulation functions

You can perform `get` and `set` operations for each of the following objects:

#### Active links

- ARGetActi veLi nkFromXML
- ARSetActi veLi nkToXML

## Character menus

- ARGetMenuFromXML
- ARSetMenuToXML

## Containers

- ARGetContainerFromXML
- ARSetContainerToXML

## DSO

- ARGetDSOMappingFromXML
- ARSetDSOMappingToXML
- ARGetDSOPoolFromXML
- ARSetDSOPoolToXML

## Escalations

- ARGetEscalationFromXML
- ARSetEscalationToXML

## Fields

- ARGetFieldFromXML
- ARSetFieldToXML

## Filters

- ARGetFilterFromXML
- ARSetFilterToXML

## Schemas

- ARGetSchemaFromXML
- ARSetSchemaToXML

## VUIs

- ARGetVUIFromXML
- ARSetVUIToXML

## Other calls for object manipulation

The XML API also includes the following calls to object manipulation:

- `ARParseXMLDocument`—creates a parsed XML stream for the `ARGetXML` calls.
- `ARGetListXMLObjects`—retrieves object names and types from an XML document.
- `ARSetXMLDocHeaderToXML` and `ARSetXMLDocFooterToXML`—generates the headers and footers of an XML document.

## Using the object XML calls

You can use the XML calls to transform XML documents to AR System structures, or AR System structures to XML documents. You can also use the `ARGetListXMLObjects` call to get a list of the AR System structures in an XML document.

Make sure that you call the `ARInitiateInitialization` call at the beginning of your API program. See Chapter 4, “AR System C API calls.”

### Transforming XML documents into AR System structures

Use the following AR System XML API calls and structures to transform XML documents into AR System structures:

`ARXMLInputDoc`

Supply this structure with the XML document to parse, in one of the following forms:

- `AR_XML_DOC_CHAR_ST`—null-terminated character string (you must free the memory associated with this action)
- `AR_XML_DOC_FILE_NAME`—file name
- `AR_XML_DOC_URL`—URL

See the `ar.h` file for definitions of this structure.

`ARParseXMLDocument`

This call parses the XML document and returns a parsed XML stream and a list of object names and types.

The combination of object name and object type must be unique for each object in the XML document. For example, you cannot have two schemas named ABC in a single XML document. However, you can have a schema and a filter named ABC in one XML document, as these are different object types.

#### ARGet<*object*>FromXML

These calls retrieve the properties of the object from the parsed XML stream and places the properties into the output parameters. Supply NULL for any property that you do not want to retrieve.

#### FreeARXMLParsedStream

This call frees memory associated with the parsed stream. For more information, see arfree.h.

## Transforming AR System structures into XML documents

Use the following AR System API calls and structures to transform AR System structures into XML documents:

#### ARXMLOutputDoc

This structure contains the XML document generated from the Set calls in one of the following forms:

- AR\_XML\_DOC\_CHAR\_STR—null-terminated character string
- AR\_XML\_DOC\_FILE\_NAME—file name (A new file is created. If the file already exists, the new XML contents are appended to the end of the file.)
- AR\_XML\_DOC\_FILE\_HANDLE—file handle

See the ar.h file for definitions of this structure.

#### ARSetHeaderToXML

This call generates the XML declaration and start tag for the root element (such as <?xml...?><root>).

The server sets the document encoding in the XML declaration to the encoding of your local machine.

#### ARSet<*structure*>ToXML

This call transforms AR System structures into XML.

### ARSetFooterToXML

This call generates the XML footer, which is the ending tag for the root element (</root>).

### FreeARXMLParsedStream

This call frees memory associated with parsed stream.

If you only have a single structure to transform to XML, you can set the `xml.DocHdrFtrFlg` flag to TRUE (T) to automatically generate the XML document header and footer, instead of calling the `ARSetHeaderToXML` and `ARSetFooterToXML` functions. If you transform multiple structures to XML and combine them together, set this flag to FALSE (F).

## Listing AR System structures

Use the `ARGetListXMLObjects` call to list the AR System objects in an XML document and retrieve the following object characteristics:

- The object name.
- The object type, which is in the form `AR_STRUCTITEM_XML_<object_type>`, where `<object_type>` is filter, schema, or other object types.
- The object subtype. For container objects, the subtype is in the form `ARCON_<container_type>`, where `<container_type>` is a guide, application, packing list, or filter guide. Currently only containers have a valid subtype. The subtype value for non-container objects is -1.

If you are only retrieving object names, use the object subtype to increase the retrieval speed.

See the `ar.h` file for object values.

## Function descriptions

This section describes each XML API function call and its components.

---

Note: In API function descriptions, when a value is said to be specified, it means the value is found within the synopsis for that particular API function.

---

# ARGetActiveLinkFromXML

**Description** Retrieves an active link from an XML document.

**Privileges** All users.

**Synopsis**

```
#include "arextern.h"

int ARGetActiveLinkFromXML(
    ARControlStruct *control,
    ARXMLParsedStream *parsedStream,
    activeLinkName,
    appBlockName,
    *executionOrder,
    *workflowConnect,
    *accessList,
    *executeOn,
    *controlFieldID,
    *focusFieldID,
    *enabled,
    *query,
    *ifActionList,
    *elseActionList,
    *supportFieldList,
    owner,
    lastModifiedBy,
    *modifiedDate,
    **helpText,
    **changeHistory,
    *objPropList,
    *arDocVersion,
    *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.

**parsedStream**

The parsed XML stream.

## activeLinkName

The active link name. Each active link name must be unique.

## appBlockName

For deployable applications, the application block name of the active link.

## Return values

### executionOrder

A value between 0 and 1000 (inclusive) that determines the active link execution order. When multiple active links are associated with a form, the value associated with each active link determines the order in which they are processed (from lowest to highest). Specify `NULL` for this parameter if you do not want to retrieve this value.

### workflowConnect

The list of form names the active link is linked to. The active link must be associated with a single form or a list of forms that currently exists on the server. Specify `NULL` for this parameter if you do not want to retrieve this value.

### accessList

A list of lists containing zero or more groups who can access the fields retrieved. Access to this information is limited to users with AR System administrator privileges only. Specify `NULL` for this parameter if you do not want to retrieve the permissions.

### executeOn

A bitmask indicating the form operations that trigger the active link. Specify `NULL` for this parameter if you do not want to retrieve this value.

### controlFieldID

The ID of the field that represents the button, toolbar button, or menu item associated with executing the active link. The system returns zero if the `executeMask` does not include the `AR_EXECUTE_ON_BUTTON` condition. Specify `NULL` for this parameter if you do not want to retrieve this value.

### focusFieldID

The ID of the field associated with executing the active link by pressing Return or selecting a character menu item. The system returns zero if the `executeMask` does not include the `AR_EXECUTE_ON_RETURN` or `AR_EXECUTE_ON_MENU_CHOICE` conditions. Specify `NULL` for this parameter if you do not want to retrieve this value.

**enabled**

A flag identifying whether the active link is disabled (0) or enabled (1). Specify `NULL` for this parameter if you do not want to retrieve this value.

**query**

A qualification that determines whether the active link is executed. The system returns zero (`AR_COND_OP_NONE`) if the active link has no qualification. Specify `NULL` for this parameter if you do not want to retrieve this value.

**ifActionList**

The set of actions performed if the condition defined by the `query` parameter is satisfied. This list can contain from 1 to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve this value.

**elseActionList**

The set of actions performed if the condition defined by the `query` parameter is not satisfied. This list can contain from 0 to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve this value.

**supportFileList**

The support file that the active link uses.

**owner**

The active link owner. Specify `NULL` for this parameter if you do not want to retrieve this value.

**lastModifiedBy**

The user who last modified the active link. Specify `NULL` for this parameter if you do not want to retrieve this value.

**modifiedDate**

The date that the active link was last modified.

**helpText**

The help text associated with the active link. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

**changeHistory**

The change history of the active link.

**objPropList**

The object properties of the active link.

**arDocVersion**

The XML document version.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARSetActiveLinkToXML.

## ARGetContainerFromXML

**Description** Retrieves containers from an XML document.

**Privileges** All users.

**Synopsis**

```
#include "arextern.h"

int ARGetContainerFromXML(
    ARControlStruct *control,
    ARXMLParsedStream *parsedStream,
    containerName,           /* containerName */
    appBlockName,            /* appBlockName */
    *permissionList,         /* permissionList */
    *subAdminGrpList,        /* subAdminGrpList */
    *ownerObjList,           /* ownerObjList */
    **label,                 /* label */
    **description,           /* description */
    *containerType,          /* containerType */
    *referenceList,          /* referenceList */
    owner,                   /* owner */
    lastModifiedBy,          /* lastModifiedBy */
    *modifiedDate,           /* modifiedDate */
    **helpText,               /* helpText */
    **changeHistory);        /* changeHistory */
```

```
ARPropList  
unSignedInt  
ARStatusList
```

\*obj PropList,  
\*arDocVersion,  
\*status)

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.

### parsedStream

The parsed XML stream.

### containerName

The container name. Each container name must be unique.

### appBlockName

For a deployable application, this is the application block name of the container.

## Return values

### permissionList

A list of zero or more groups who can access this container. Access to this information is limited to users with AR System administrator privileges only. Specify `NULL` for this parameter if you do not want to retrieve this value.

### subAdminGrpList

The list of groups that have access to the container.

### ownerObjectList

A list of schemas that own this container. This parameter can be `NULL` if the container exists globally.

### label

The label for this container. It can be as many as 255 characters long or `NULL`.

### description

The description for this container. It can be as many as 2000 characters long or `NULL`.

## containerType

The type for this container—either guide (ARCON\_GUI DE), application (ARCON\_APP), or a custom type you have defined.

## referenceList

A list of pointers to the objects referenced by this container. References can be to internal AR System objects (for example, guides reference active links and applications reference forms) or to external objects such as URLs or file names. Specify `NULL` for this parameter if you do not want to associate any objects with this container.

## owner

The container owner.

## lastModifiedBy

The user who last modified the container.

## modifiedDate

The date that the container was last modified.

## helpText

The help text associated with the container. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

## changeHistory

The change history of the container.

## objPropList

The object properties of the container.

## arDocVersion

The XML document version.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

See also ARSetContainerToXML.

# ARGetDSOMappingFromXML

**Description** Retrieves DSO mapping information from an XML document. For more information about DSO mapping, see the *Administering BMC Remedy DSO* guide.

**Privileges** All users.

**Synopsis**

```
#include "arextern.h"

int ARGetDSOMappingFromXML(
    ARControlStruct *control,
    ARXParsedStream *parsedStream,
    mappingName,
    appBlockName,
    fromSchema,
    fromServer,
    toSchema,
    toServer,
    *enabled,
    updateCode,
    *transferMode,
    *mapType,
    *rtnMapType,
    *defaultMap,
    *duplicateAction,
    *patternMatch,
    *requiredFields,
    *retryTime,
    **mapping,
    **rtnMapping,
    **matchQual,
    owner,
    lastModifiedBy,
    *modifiedDate,
    **helpText,
    **changeHistory,
    *objPropList,
    *arDocVersion,
    *status)
```

Input arguments	<b>control</b>
	The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.
	<b>parsedStream</b>
	The parsed XML stream.
	<b>mappingName</b>
	The name of the schema to be exported. Each schema name must be unique.
	<b>appBlockName</b>
	For a deployable application, this is the application block name of the DSO mapping.
Return values	<b>fromSchema</b>
	The name of the source schema. Each schema name must be unique.
	<b>fromServer</b>
	The name of the source server.
	<b>toSchema</b>
	The name of the target schema.
	<b>toServer</b>
	The name of the target server.
	<b>enabled</b>
	A flag identifying whether the DSO mapping is disabled (0) or enabled (1). Specify NULL for this parameter if you do not want to retrieve this value.
	<b>updateCode</b>
	A value that indicates when the system should update the DSO mapping after changing the entry.
	<b>transferMode</b>
	The transfer mode.
	<b>mapType</b>
	The type of mapping to use on transfer.

**rtnMapType**

The type of mapping to use on update or return.

**defaultMap**

A value that indicates the default mapping.

**duplicateAction**

The action to perform on transfer to existing entry.

**patternMatch**

A value indicating that pattern matching is enabled.

**requiredFields**

A value that indicates if required fields can be ignored.

**retryTime**

The maximum number of seconds that the server waits to remap the schema, before it cancels the operation.

**mapping**

A value that indicates custom mapping if the `mapType` value is `custom`.

**rtnMapping**

A value that indicates custom mapping if the `rtnMapType` value is `custom`.

**matchQual**

The matching qualification.

**owner**

The owner of the DSO mapping.

**lastModifiedBy**

The user who last modified the DSO mapping.

**modifiedDate**

The date that the DSO mapping was last modified.

**helpText**

The help text associated with the DSO mapping. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

**changeHistory**

The change history of the DSO mapping.

**objPropList**

The properties of the DSO mapping object.

**arDocVersion**

The XML document version.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## ARGetDSOPoolFromXML

**Description** Retrieves DSO pool information from an XML document.

**Privileges** All users.

**Synopsis**

```
#include "arextern.h"

int ARGetDSOPoolFromXML(
    ARControlStruct*control,
    ARXMLParsedStream*parsedStream,
    ARNameType*poolName,
    ARNameType*appBlockName,
    *enabled,
    *defaultPool,
    *threadCount,
    **connection,
    owner,
    lastModifiedBy,
    *modifiedDate,
    **helpText,
    **changeHistory,
    *objPropList,
    *arDocVersion,
    *status)
```

Input arguments	
	<b>control</b>
	The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.
	<b>parsedStream</b>
	The parsed XML stream.
	<b>poolName</b>
	The name of the DSO pool. Each pool name must be unique.
	<b>appBlockName</b>
	For a deployable application, this is the application block name of the DSO pool.
Return values	
	<b>enabled</b>
	A flag identifying whether the DSO pool is disabled (0) or enabled (1). Specify NULL for this parameter if you do not want to retrieve this value.
	<b>defaultPool</b>
	A value that indicates that this is the default pool.
	<b>threadcount</b>
	The thread count used in this call.
	<b>connection</b>
	This parameter is reserved for future use.
	<b>owner</b>
	The owner of the DSO pool.
	<b>lastModifiedBy</b>
	The user who last modified the DSO pool.
	<b>modifiedDate</b>
	The date that the DSO pool was last modified.

**helpText**

The help text associated with the DSO pool. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

**changeHistory**

The change history of the DSO pool.

**objPropList**

The properties of the DSO pool.

**arDocVersion**

The XML document version.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## ARGetEscalationFromXML

**Description** Retrieves escalations from an XML document.

**Privileges** All users.

**Synopsis** #include "arextern.h"

```
int ARGetEscalationFromXML(
    ARControlStruct *control,
    ARXParsedStream *parsedStream,
    ARNameType escalati onName,
    ARNameType appB l ockName,
    AREscalati onTmStruct *escalati onTi me,
    ARWorkflowConnectStruct *workfl owConnect,
    unsigned int *enabled,
    ARQualifierStruct *query,
    ARFilterActi onList *filterActi onList,
    ARFilterActi onList *elseActi onList,
    ARAccessNameType *owner,
    ARTimestamp lastModifiedBy,
    ARTimestamp modifiedDate,
```

```
char          **hel pText,
char          **changeHi story,
ARPropList    *obj PropList,
unSignedInt   *arDocVersion,
ARStatusList  *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.

### parsedStream

The parsed XML stream.

### escalationName

The name of the escalation. Each escalation name must be unique.

### appBlockName

For a deployable application, this is the application block name of the escalation.

## Return values

### escalationTime

The time specification for evaluating the escalation condition. This parameter can take one of two forms: a time interval that defines how frequently the server checks the escalation condition (in seconds) or a bitmask that defines a particular day (by month or week) and time (hour and minute) for the server to check the condition. Specify `NULL` for this parameter if you do not want to retrieve this value.

### workflowConnect

The list of form names the escalation is linked to. The escalation must be associated with a single form or a list of forms that currently exists on the server. Specify `NULL` for this parameter if you do not want to retrieve this value.

### enabled

A flag identifying whether the escalation is disabled (0) or enabled (1). Specify `NULL` for this parameter if you do not want to retrieve this value.

## query

A query operation performed when the escalation is executed that determines the set of entries to which the escalation actions (defined by the `actionList` parameter) are applied. The system returns 0 (`AR_COND_OP_NONE`) if the escalation has no qualification. Specify `NULL` for this parameter if you do not want to retrieve this value.

## ifActionList

The set of actions performed for each entry that matches the criteria defined by the `query` parameter. This list can contain from 1 to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve this value.

## elseActionList

The set of actions performed if no entries match the criteria defined by the `query` parameter. This list can contain from zero to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve this value.

## owner

The escalation owner.

## lastModifiedBy

The user who last modified the escalation.

## modifiedDate

The date that the escalation was last modified.

## helpText

The help text associated with the escalation. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

## changeHistory

The change history of the escalation.

## objPropList

The object properties of the escalation.

## arDocVersion

The XML document version.

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARSetEscalationToXML.

# ARGetFieldFromXML

**Description** Retrieves fields from an XML document.

**Privileges** All users.

**Synopsis** #include "arextern.h"

```
int ARGetFieldFromXML(
    ARControlStruct *control,
    ARXMLParsedStream *parsedStream,
    const char *fieldName,
    const char *appName,
    const char *fieldID,
    const char *fieldMapping,
    const char *dataType,
    const char *entryMode,
    const char *createMode,
    const char *fieldOption,
    const char *defaultVal,
    const char *permissons,
    const char *fieldList,
    const char *instanceList,
    const char *owner,
    const char *lastModifiedBy,
    const char *modifiedDate,
    const char *helpText,
    const char *changeHistory,
    const char *arDocVersion,
    const char *status)
```

Input arguments	<b>control</b>
	The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.
	<b>parsedStream</b>
	The parsed XML stream.
	<b>fieldName</b>
	The field name. Each field name must be unique.
	<b>appBlockName</b>
	For a deployable application, this is the application block name of the field.
Return values	<b>fieldId</b>
	The internal ID of the field to retrieve.
	<b>fieldMapping</b>
	The underlying form that contains the field (applicable for join forms only). Specify <code>NULL</code> for this parameter if you do not want to retrieve the field mapping.
	<b>dataType</b>
	The data type of the field. See <code>ARCreateField</code> information in Chapter 4, “AR System C API calls,” for a description of the possible values. Specify <code>NULL</code> for this parameter if you do not want to retrieve the data type.
	<b>entryMode</b>
	The entry mode type.
	<b>createMode</b>
	A flag indicating the permission status for the field when users submit entries. See <code>ARCreateField</code> information in Chapter 4, “AR System C API calls,” for a description of the possible values. Specify <code>NULL</code> for this parameter if you do not want to retrieve this flag.

## fieldOption

A bitmask that indicates whether the field should be audited or copied when other fields are audited.

- Bit 0: Value 1: Audit this field. (AR\_FIELD\_BIT0PTION\_AUDIT)
- Bit 1: Value 1: Copy this field when other fields in the form are audited. (AR\_FIELD\_BIT0PTION\_COPY)
- Bit 2: Value 1: Indicates this field is for Log Key 1. (AR\_FIELD\_BIT0PTION\_LOG\_KEY1)
- Bit 3: Value 1: Indicates this field is for Log Key 2. (AR\_FIELD\_BIT0PTION\_LOG\_KEY2)
- Bit 2 and 3: Both value 1: Indicates this field is for Log Key 3. (AR\_FIELD\_BIT0PTION\_LOG\_KEY3)

## defaultValue

The value to apply if a user submits an entry with no field value (applicable for data fields only). The system returns 0 (AR\_DEFAULT\_VALUE\_NONE) if the field has no default. Specify `NULL` for this parameter if you do not want to retrieve the default value.

## permissionList

The list of group access permissions.

## fieldLimit

The value limits for the field and other properties specific to the field's type. See the `ARFieldDefinitionStruct` definition in the `ar.h` file to find the contained structure that applies to the type of field you are creating. Specify `NULL` (or `AR_FIELD_LIMIT_NONE`) for trim and control fields or if you do not want to retrieve the limits and properties.

## displayList

A list of zero or more display properties associated with the field. See “`ARCreateField`” on page 143 for a description of the possible values. The system returns 0 (`AR_DPROP_NONE`) if the field has no display properties. Specify `NULL` for this parameter if you do not want to retrieve this list.

## owner

The owner for the field.

**lastModifiedBy**

The user who last modified the field.

**modifiedDate**

The date that the field was last modified.

**helpText**

The help text associated with the field. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

**changeHistory**

The change history of the field.

**arDocVersion**

The XML document version.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

See also ARSetFieldToXML.

## ARGetFilterFromXML

**Description** Retrieves a filter from an XML document.

**Privileges** All users.

**Synopsis** #include "arextern.h"

```
int ARGetFilterFromXML(
    ARControlStruct          *control,
    ARXMLParsedStream         *parsedStream,
    filterName                filterName,
    appBlockName               appBlockName,
    *executionOrder            *executionOrder,
    *workflowConnect           *workflowConnect,
    *executeOn                 *executeOn,
    *enabled                   *enabled,
```

	ARQualifierStruct ARFilterActi onList ARFilterActi onList ARAccessNameType ARAccessNameType ARTimestamp char char ARPropList unsigned int ARStatusList	*query, *filterList, *elseActi onList, owner, lastModifiedBy, *modifiedDate, **helpText, **changeHistory, *objPropList, *arDocVersion, *status)
<b>Input arguments</b>	<b>control</b>	The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.
	<b>parsedStream</b>	The parsed XML stream.
	<b>filterName</b>	The filter name. Each filter name must be unique.
	<b>appBlockName</b>	For a deployable application, this is the application block name of the filter.
<b>Return values</b>	<b>executionOrder</b>	A value between 0 and 1000 (inclusive) that determines the filter execution order. When multiple filters are associated with a form, the value associated with each filter determines the order in which they are processed (from lowest to highest). Specify NULL for this parameter if you do not want to retrieve this value.
	<b>workflowConnect</b>	The list of form names the filter is linked to. The filter must be associated with a single form or a list of forms that currently exists on the server. Specify NULL for this parameter if you do not want to retrieve this value.

**executeOn**

A bitmask indicating the form operations that trigger the filter. Specify `NULL` for this parameter if you do not want to retrieve this value.

**enabled**

A flag identifying whether the filter is disabled (0) or enabled (1). Specify `NULL` for this parameter if you do not want to retrieve this value.

**query**

A qualification that determines whether the filter is executed. The system returns zero (`AR_COND_OP_NONE`) if the active link has no qualification. Specify `NULL` for this parameter if you do not want to retrieve this value.

**ifActionList**

The set of actions performed for each entry that matches the criteria defined by the `query` parameter. This list can contain from 1 to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve this value.

**elseActionList**

The set of actions performed if the condition defined by the `query` parameter is not satisfied. This list can contain from 0 to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve this value.

**owner**

The owner of the filter.

**lastModifiedBy**

The user who last modified the filter.

**modifiedDate**

The date that the filter was last modified.

**helpText**

The help text associated with the filter. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

**changeHistory**

The change history of the filter.

**objPropList**

The object properties of the filter.

**arDocVersion**

The XML document version.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARSetFilterToXML.

## ARGetListXMLObjects

**Description** Retrieves object names and types from an XML document.

**Privileges** All users.

**Synopsis**

```
#include "arextern.h"

int ARGetListXMLObjектs(
    ARControlStruct *control,
    ARXMLInputDoc *xmlInputDoc,
    ARUnsignedList *objektTypeList,
    ARUnsignedList *objectNameList,
    ARUnsignedList *subTypeList,
    ARUnsignedList *applicationNameList,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user`, `session`, and `server` fields are required.

**xmlDocSource**

The XML document output source.

**Return values**

**objectTypeList**

A list of the object types.

**objectNameList**

A list of the object names.

**subTypeList**

The object subtypes.

**appBlockNameList**

In a deployable application the list of application block names for the object. The array indexes of the application block name list corresponds to the array index of the `parsedObj ects` lists. You can supply NULL for this parameter if you do not want to retrieve it.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## ARGetMenuFromXML

**Description** Retrieves menus from an XML document.

**Privileges** All users.

**Synopsis**

```
#include "arextern.h"

int ARGetMenuFromXML(
    ARControlStruct           *control,
    ARXMLParsedStream          *parsedStream,
    ARNameType                 menuName,
    ARNameType                 appBlockName,
    *refreshCode,
    *menuDefn,
    owner,
    lastModifiedBy,
    *modifiedDate,
    **helpText,
    **changeHistory,
    *objPropList,
    *arDocVersion,
    *status)
```

Input arguments	<b>control</b>
	The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.
	<b>parsedStream</b>
	The parsed XML stream.
	<b>menuName</b>
	The menu name. Each menu name must be unique.
	<b>appBlockName</b>
	For a deployable application, this is the application block name of the menu.
Return values	<b>refreshCode</b>
	A value indicating when the menu is refreshed. See “ARCreateCharMenu” on page 134 for a description of the possible values. Specify <code>NULL</code> for this parameter if you do not want to retrieve this value.
	<b>menuDefn</b>
	The definition of the character menu. Specify <code>NULL</code> for this parameter if you do not want to retrieve this value.
	<b>owner</b>
	The menu owner.
	<b>lastModifiedBy</b>
	The user who last modified the menu.
	<b>modifiedDate</b>
	The date that the menu was last modified.
	<b>helpText</b>
	The help text associated with the menu. Specify <code>NULL</code> for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).
	<b>changeHistory</b>
	The change history of the menu.

**objPropList**

The object properties of the menu.

**arDocVersion**

The XML document version.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## ARGetSchemaFromXML

**Description** Retrieves schemas (forms) from an XML document.

**Privileges** All users.

**Synopsis** #include "arextern.h"

```
int ARGetSchemaFromXML(
    ARControlStruct *control,
    ARXParsedStream *parsedStream,
    const char *schemaName,
    const char *appBlockName,
    ARCompoundSchema *compoundSchema,
    ARPermissionList *permissionList,
    ARInternalList *subAdminGroupList,
    ARListFields *getLists,
    ARSortList *sortList,
    ARIndexList *indexList,
    ARArchivedInfoStruct *archivedInfo,
    ARAuditInfoStruct *auditInfo,
    defaultTVUI *default,
    nextFieldID *nextFieldID,
    coreVersion *coreVersion,
    upgradeVersion *upgradeVersion,
    fieldInfoList *fieldInfoList,
    vuiInfoList *vuiInfoList,
    owner *owner,
    lastModifiedBy *lastModifiedBy,
    modified *modifiedDate,
```

```
char          **helpText,
char          **changeHistory,
ARPropList    *objPropList,
unsigned int  *arDocVersion,
ARStatusList  *status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.

### parsedStream

The parsed XML stream.

### schemaName

The schema name. Each schema name must be unique.

### appBlockName

For a deployable application, this is the application block name of the schema (form).

## Return values

### compoundSchema

The definition of the candidate form or table that contains the candidate field to retrieve.

### permissionList

A list of zero or more groups who can access the form. Access to this information is limited to users with AR System administrator privileges only. Specify `NULL` for this parameter if you do not want to retrieve the group list.

### subAdminGrpList

A list of zero or more groups who can administer this form (and the associated filters, escalations, and active links). Users must belong to both a specified group *and* the Subadministrator group to obtain these privileges. Specify `NULL` for this parameter if you do not want to retrieve the administrator group list.

## getListFields

A list of zero or more fields that identifies the default query list data for retrieving form entries. Specify `NULL` for this parameter if you do not want to retrieve this value.

## sortList

A list of zero or more fields that identifies the default sort order for retrieving form entries. Specify `NULL` for this parameter if you do not want to retrieve the default sort fields.

## indexList

The set of zero or more indexes for the form. Specify `NULL` for this parameter if you do not want to retrieve the index list.

## archiveInfo

If the form is to be archived, this is the archive information for the form. Specify `NULL` for this parameter if you do not want to retrieve the archive information. The following are the options for archive type:

- 1 The form is selected to have entries copied to the archive form and deleted from the main form (`AR_ARCHIVE_FORM`).
- 2 The form is selected to have entries deleted from the main form (`AR_ARCHIVE_DELETE`).
- 16 The form is selected for archive without attachment fields (`AR_ARCHIVE_NO_ATTACHMENTS`).
- 32 The form is selected for archive without diary fields (`AR_ARCHIVE_NO_DIARY`).

In addition to the archive type, `archiveInfo` also stores the form name (if archive type is `AR_ARCHIVE_FORM`), time to trigger the archive, archive qualification criteria, and name of the main form that is being archived.

## auditInfo

If a form is to be audited, this is the audit information for the form. Specify `NULL` for this parameter if you do not want to retrieve the audit information. These are the values for audit type:

- 0: The selected form is not to be audited. (`AR_AUDIT_NONE`).
- 1: Audit fields and copy fields on the selected form are copied to the audit shadow form (`AR_AUDIT_COPY`).

- 2: Audit fields and copy fields on the selected form are copied to the audit log form (AR\_AUDIT\_LOG).

In addition to the audit type, audi\_tInfo also stores the form name (if audit type is AR\_AUDIT\_COPY or AR\_AUDIT\_LOG) and the audit qualification criteria.

#### **defaultVui**

The label for the default view.

#### **nextFieldID**

The next default field identification number, which the server generates if users do not specify it.

#### **coreVersion**

The version number of the core fields.

#### **upgradeVersion**

The upgrade version for autogenerated internal server schemas.

#### **fieldInfoList**

The list of schema fields.

#### **vuiInfoList**

The list of schema views.

#### **owner**

The schema owner.

#### **lastModifiedBy**

The user who last modified the schema.

#### **modifiedDate**

The date that the schema was last modified.

#### **helpText**

The help text associated with the schema. Specify NULL for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

#### **changeHistory**

The change history of the schema.

**objPropList**

The object properties of the schema.

**arDocVersion**

The XML document version.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARSetSchemaToXML.

## ARGetVUIFromXML

**Description** Retrieves views from an XML document.

**Privileges** All users.

**Synopsis** #include "arextern.h"

```
int ARGetVUIFromXML(
    ARControlStruct *control,
    ARXMLParsedStream *parsedStream,
    ARSchemaName *schemaName,
    ARBlockName *blockName,
    ARViewList *vuiList,
    ARFieldList *fieldList,
    ARTimestamp *modifiedDate,
    ARDocVersion *arDocVersion,
    *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user`, `session`, and `server` fields are required.

**parsedStream**

The parsed XML stream.

**schemaName**

The name of the view. Each view name must be unique.

**appBlockName**

For a deployable application, this is the application block name of the view.

**Return values****vuiInfoList**

The list of schema views.

**fieldInfoList**

The list of view fields.

**modifiedDate**

The date that the view was last modified.

**arDocVersion**

The XML document version.

**status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARSetVUIToXML.

## ARParseXMLDocument

**Description** Parses the contents of an XML document.

**Privileges** All users.

**Synopsis** #include "arextern.h"

```
int ARParseXMLDocument(
    ARControlStruct *control,
    ARXMLInputDoc *xmlInputDoc,
    ARStructItemList *objectsToParse,
    ARXMLParsedStream *parsedStream,
    ARStructItemList *parsedObjects,
    ARNameList *appBlockNameList,
    ARStatusList *status)
```

<b>Input arguments</b>	<b>control</b>	The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.
	<b>xmlInputDoc</b>	The XML document to be parsed.
	<b>objectsToParse</b>	The specific list of objects to parse. Specify <code>NULL</code> if you do not want to parse any objects.
<b>Return values</b>	<b>parsedStream</b>	The parsed XML stream.
	<b>parsedObjects</b>	The list of parsed objects.
	<b>appBlockNameList</b>	In a deployable application the list of application block names for the object. The array indexes of the application block name list corresponds to the array index of the <code>parsedObjects</code> lists. You can supply <code>NULL</code> for this parameter if you do not want to retrieve it.
	<b>status</b>	A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the <i>Error Messages</i> guide.

## ARSetActiveLinkToXML

<b>Description</b>	Converts active links to XML.
<b>Privileges</b>	All users.
<b>Synopsis</b>	<pre>#include "arextern.h"  int ARSetActiveLinkToXML(     ARControlStruct*control,     ARXMLOutputDoc*xmlOutputDoc,</pre>

ARBool ean	xml DocHdrFtrFlag,
ARNameType	activeLinkName,
unsi gned int	*executi onOrder,
ARWorkfl owConnectStruct	*workfl owConnect,
ARI nternal LdList	*accessLi st,
unsi gned int	*executeOn,
ARI nternal Ld	*controlFi el dID,
ARI nternal Ld	*focusFi el dID,
unsi gned int	*enabl ed,
ARQual i fierStruct	*query,
ARActi veLi nkActi onLi st	*i fActi onLi st,
ARActi veLi nkActi onLi st	*el seActi onLi st,
ARSupportFi leInfoLi st	*supportFi leLi st,
ARAccessNameType	owner,
ARAccessNameType	lastModifi edBy,
ARTimestamp	*modifi edDate,
char	*helpText,
char	*changeHistory,
ARPropLi st	*obj PropLi st,
unsi gned int	*arDocVersi on,
ARStatusLi st	*status)

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.

### xmlOutputDoc

The XML document output source.

### xmlDocHdrFtrFlag

The header and footer flag for the XML document output source. If there is only one object in the XML document, set this value to (0).

### activeLinkName

The name of the active link to be set.

## executionOrder

A value between 0 and 1000 (inclusive) that determines the active link execution order. When multiple active links are associated with a form, the value associated with each active link determines the order in which they are processed (from lowest to highest). Specify `NULL` for this parameter if you do not want to retrieve this value.

## workflowConnect

The list of form names the active link is linked to. The active link must be associated with a single form or a list of forms that currently exists on the server. Specify `NULL` for this parameter if you do not want to retrieve this value.

## accessList

A list of lists containing zero or more groups who can access the fields retrieved. Access to this information is limited to users with AR System administrator privileges only. Specify `NULL` for this parameter if you do not want to retrieve the permissions.

## executeOn

A bitmask indicating the form operations that trigger the active link. Specify `NULL` for this parameter if you do not want to retrieve this value.

## controlFieldID

The ID of the field that represents the button, toolbar button, or menu item associated with executing the active link. The system returns zero if the `executeMask` does not include the `AR_EXECUTE_ON_BUTTON` condition. Specify `NULL` for this parameter if you do not want to retrieve this value.

## focusFieldID

The ID of the field associated with executing the active link by pressing Return or selecting a character menu item. The system returns zero if the `executeMask` does not include the `AR_EXECUTE_ON_RETURN` or `AR_EXECUTE_ON_MENU_CHOICE` conditions. Specify `NULL` for this parameter if you do not want to retrieve this value.

## enabled

A flag identifying whether the active link is disabled (0) or enabled (1). Specify `NULL` for this parameter if you do not want to retrieve this value.

## query

A query operation performed when the escalation is executed that determines the set of entries to which the escalation actions (defined by the `actionList` parameter) are applied. The system returns 0 (`AR_COND_OP_NONE`) if the escalation has no qualification. Specify `NULL` for this parameter if you do not want to retrieve this value.

## ifActionList

The set of actions performed for each entry that matches the criteria defined by the `query` parameter. This list can contain from 1 to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve this value.

## elseActionList

The set of actions performed if the condition defined by the `query` parameter is not satisfied. This list can contain from 0 to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve this value.

## supportFileList

The support file that the active link uses.

## owner

The active link owner.

## lastModifiedBy

The user who last modified the active link.

## modifiedDate

The date that the active link was last modified.

## helpText

The help text associated with the active link. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

## changeHistory

The change history of the active link.

## objPropList

The object properties of the active link.

## arDocVersion

The XML document version.

### Return values

#### [status](#)

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** [ARGetActiveLinkFromXML](#).

## ARSetContainerToXML

**Description** Converts containers to XML.

**Privileges** All users.

**Synopsis** #include "arextern.h"

```
int ARSetContainerToXML(
    ARControlStruct*control,
    ARXMLOutputDoc*xmlOutputDoc,
    XMLDocHdrFtrFlag contai nerName,
    ARPerm  issi onLi st*permis sionLi st,
    ARSubAdmi nGrpLi st*subAdmi nGrpLi st,
    AROwnerObjLi st*ownerObj ectLi st,
    ARLabel*label,
    ARDescription*description,
    ARContainerType*contai nerType,
    ARReferenceLi st*referenceLi st,
    AROwner*owner,
    ARLastModifi edBy*lastModifi edBy,
    ARModifi edDate*modifi edDate,
    ARHelpText*helpText,
    ARChangeHistory*changeHistory,
    ARObjectPropList*obj PropLi st,
    ARDocVersion*arDocVersi on,
    ARStatus*status)
```

**Input  
arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.

**xmlOutputDoc**

The XML document output source.

**xmlDocHdrFtrFlag**

The header and footer flag for the XML document output source. If there is only one object in the XML document, set this value to (0).

**containerName**

The container name. Each container name must be unique.

**permissionList**

A list of zero or more groups who can access this container. Access to this information is limited to users with AR System administrator privileges only. Specify NULL for this parameter if you do not want to retrieve this value.

**subAdminGrpList**

The list of groups that have access to the container.

**ownerObjectList**

A list of schemas that own this container. This parameter can be NULL if the container exists globally.

**label**

The label for this container. It can be as many as 255 characters long or NULL.

**description**

The description for this container. It can be as many as 2000 characters long or NULL.

**containerType**

The type for this container—either guide (ARCON\_GUI DE), application (ARCON\_APP), or a custom type you have defined.

## referenceList

A list of pointers to the objects referenced by this container. References can be to internal AR System objects (for example, guides reference active links and applications reference forms) or to external objects such as URLs or file names. Specify `NULL` for this parameter if you do not want to associate any objects with this container.

## owner

The container owner.

## lastModifiedBy

The user who last modified the container.

## modifiedDate

The date that the container was last modified.

## helpText

The help text associated with the container. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

## changeHistory

The change history of the container.

## objPropList

The object properties of the container.

## arDocVersion

The XML document version.

## Return values

### status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARGetContainerFromXML.

# ARSetDSOMappingToXML

**Description** Retrieves information about the DSO mapping. For more information about DSO mapping, see the *Administering BMC Remedy DSO* guide.

**Privileges** All users.

**Synopsis**

```
#include "arextern.h"

int ARSetDSOMappingToXML(
    ARControlStruct *control,
    ARXMLOutputDoc *xmlOutputDoc,
    XMLDocHdrFtrFlag mappingName,
    ARNameType fromSchema,
    ARNameType fromServer,
    ARServerNameType toSchema,
    ARServerNameType toServer,
    ARNameType *enabled,
    ARNameType *updateCode,
    ARNameType *transferMode,
    ARNameType *mapType,
    ARNameType *rtnMapType,
    ARNameType *defaultMap,
    ARNameType *duplicateAction,
    ARNameType *patternMatch,
    ARNameType *requiredFields,
    ARNameType *retryTime,
    ARNameType **mapping,
    ARNameType **rtnMapping,
    ARNameType **matchQual,
    ARNameType owner,
    ARNameType lastModifiedBy,
    ARNameType modifiedDate,
    ARNameType helpText,
    ARNameType changeHistory,
    ARNameType objPropList,
    ARNameType arDocVersion,
    ARNameType status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.

**xmlOutputDoc**

The XML document output source.

**xmlDocHdrFtrFlag**

The header and footer flag for the XML document output source. If there is only one object in the XML document, set this value to (0).

**mappingName**

The name of the schema to be exported. Each schema name must be unique.

**fromSchema**

The name of the source schema. Each schema name must be unique.

**fromServer**

The name of the source server.

**toSchema**

The name of the target schema.

**toServer**

The name of the target server.

**enabled**

A flag identifying whether the DSO pool is disabled (0) or enabled (1). Specify NULL for this parameter if you do not want to retrieve this value.

**updateCode**

A value that indicates when the server updates the entry after the entry is made.

**transferMode**

The transfer mode.

**mapType**

The type of mapping to use on transfer.

**rtnMapType**

The type of mapping to use on update or return.

**defaultMap**

A value that indicates the default mapping.

**duplicateAction**

The action to perform on transfer to the existing entry.

**patternMatch**

A value that indicates that pattern matching is enabled.

**requiredField**

A value that indicates if the required fields can be ignored.

**retryTime**

The maximum number of seconds that the server waits to remap the schema, before it cancels the operation.

**mapping**

A value that indicates custom mapping is turned on, if the `mapType` parameter is set to `custom`.

**rtnMapping**

A value that indicates custom mapping is turned on, if the `rtnMapType` parameter is set to `custom`.

**matchQual**

The matching qualification.

**owner**

The owner for the DSO mapping.

**lastModifiedBy**

The user who last modified the DSO mapping.

**modifiedDate**

The date that the DSO mapping was last modified.

**helpText**

The help text associated with the DSO mapping. Specify NULL for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

**changeHistory**

The change history of the DSO mapping.

**objPropList**

The properties of the DSO mapping.

**arDocVersion**

The XML document version.

**Return values****status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## ARSetDSOPoolToXML

**Description** Retrieves information about the DSO pool. For more information about DSO mapping, see the *Administering BMC Remedy DSO* guide.

**Privileges** All users.

**Synopsis**

```
#include "arextern.h"
```

```
int ARSetDSOPoolToXML(
    ARControlStruct *control,
    ARXMLOutputDoc *xmlOutputDoc,
    XMLDocHdrFtrFlag,
    poolName,
    *enabled,
    *defaultPool,
    *threadCount,
    **connection,
    owner,
    lastModifiedBy,
    *modifiedDate,
    *helpText,
```

```
char *changeHi story,
ARPropList *obj PropList,
unsigned int *arDocVersion,
ARStatusList *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.

**xmlOutputDoc**

The XML document output source.

**xmlDocHdrFtrFlag**

The header and footer flag for the XML document output source. If there is only one object in the XML document, set this value to (0).

**poolName**

The name of the DSO pool. Each pool name must be unique.

**enabled**

A flag identifying whether the DSO pool is disabled (0) or enabled (1). Specify NULL for this parameter if you do not want to retrieve this value.

**defaultPool**

The code that indicates whether there is a default pool.

**threadcount**

The thread count used for this call.

**connection**

This parameter is reserved for future use.

**owner**

The owner for the DSO pool.

**lastModifiedBy**

The user who last modified the DSO pool.

**modifiedDate**

The date that the DSO pool was last modified.

**helpText**

The help text associated with the DSO pool. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

**changeHistory**

The change history of the DSO pool.

**objPropList**

The properties of the DSO pool.

**arDocVersion**

The XML document version.

**Return values****status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

## ARSetEscalationToXML

**Description** Converts escalations to XML.

**Privileges** All users.

**Synopsis** #include "arextern.h"

```
int ARsetEscalationToXML(
    ARControlStruct *control,
    ARXMLOutputDoc *xmlOutputDoc,
    XMLDocHdrFtrFlag xmlDocHdrFtrFlag,
    ARescalationName escalationName,
    ARescalationTime escalationTime,
    ARWorkflowConnectStruct *workflowConnect,
    ARenabled enabled,
    ARquery query,
    ARfilterList filterList,
    ARfilterList filterList,
```

ARAccessNameType	owner,
ARAccessNameType	LastModifedBy,
ARTimestamp	*modifedDate,
char	*helpText,
char	*changeHistory,
ARPropList	*objPropList,
unsigned int	*arDocVersion,
ARStatusList	*status)

**Input  
arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.

**xmlOutputDoc**

The XML document output source.

**xmlDocHdrFtrFlag**

The header and footer flag for the XML document output source. If there is only one object in the XML document, set this value to (0).

**escalationName**

The name of the escalation. Each escalation name must be unique.

**escalationTime**

The time specification for evaluating the escalation condition. This parameter can take one of two forms: a time interval that defines how frequently the server checks the escalation condition (in seconds) or a bitmask that defines a particular day (by month or week) and time (hour and minute) for the server to check the condition. Specify NULL for this parameter if you do not want to retrieve this value.

**workflowConnect**

The list of form names the escalation is linked to. The escalation must be associated with a single form or a list of forms that currently exists on the server. Specify NULL for this parameter if you do not want to retrieve this value.

## enabled

A flag identifying whether the escalation is disabled (0) or enabled (1). Specify `NULL` for this parameter if you do not want to retrieve this value.

## query

A query operation performed when the escalation is executed that determines the set of entries to which the escalation actions (defined by the `actionList` parameter) are applied. The system returns 0 (`AR_COND_OP_NONE`) if the escalation has no qualification. Specify `NULL` for this parameter if you do not want to retrieve this value.

## ifActionList

The set of actions performed for each entry that matches the criteria defined by the `query` parameter. This list can contain from 1 to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve this value.

## elseActionList

The set of actions performed if no entries match the criteria defined by the `query` parameter. This list can contain from zero to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve this value.

## owner

The escalation owner.

## lastModifiedBy

The user who last modified the escalation.

## modifiedDate

The date that the escalation was last modified.

## helpText

The help text associated with the escalation. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

## changeHistory

The change history of the escalation.

## objPropList

The object properties of the escalation.

## arDocVersion

The XML document version.

## Return values

## status

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

#### See also

#### **ARGetEscalationFromXML.**

## ARSetFieldToXML

<b>Description</b>	Converts a field to XML.
<b>Privileges</b>	All users.
<b>Synopsis</b>	#include "arextern.h"  int ARSetFieldToXML( ARControlStruct *control, ARXMLOutputDoc *xmlOutputDoc, ARBool can, ARNameType *controlName, ARInternalId *internalId, ARFieldMappingStruct *fieldMapping, unsigned int entryMode, unsigned int dataType, unsigned int createMode, ARValueStruct *defaultValue, ARPermissionList *permissionsList, ARFieldListStruct *fieldListStruct, ARDisplayInstanceList *displayInstanceList, ARAccessNameType *accessNameType, ARAccessNameType *changeHistoryType, ARTimestamp *lastModifiedBy, char *modifiedDate, char *helpText, char *changeHistory);

```
    unsi gned i nt  
    ARStatusLi st
```

```
*arDocVersi on,  
*status)
```

## Input arguments

### control

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user`, `session`, and `server` fields are required.

### xmlOutputDoc

The XML document output source.

### xmlDocHdrFtrFlag

The header and footer flag for the XML document output source. If there is only one object in the XML document, set this value to (0).

### fieldName

The field name. Each field name must be unique.

### fieldId

The internal ID of the field to retrieve.

### fieldMapping

The underlying form that contains the field (applicable for join forms only). Specify `NULL` for this parameter if you do not want to retrieve the field mapping.

### dataType

The data type of the field. See “`ARCreateField`” on page 143 for a description of the possible values. Specify `NULL` for this parameter if you do not want to retrieve the data type.

### entryMode

The entry mode type.

### createMode

A flag indicating the permission status for the field when users submit entries. See “`ARCreateField`” on page 143 for a description of the possible values. Specify `NULL` for this parameter if you do not want to retrieve this flag.

## fieldOption

A bitmask that indicates whether the field should be audited or copied when other fields are audited.

- Bi t 0: Value 1: Audit this field. (AR\_FIELD\_BITOPTION\_AUDIT)
- Bi t 1: Value 1: Copy this field when other fields in the form are audited. (AR\_FIELD\_BITOPTION\_COPY)
- Bi t 2: Value 1: Indicates this field is for Log Key 1. (AR\_FIELD\_BITOPTION\_LOG\_KEY1)
- Bi t 3: Value 1: Indicates this field is for Log Key 2. (AR\_FIELD\_BITOPTION\_LOG\_KEY2)
- Bi t 2 and 3: Both value 1: Indicates this field is for Log Key 3. (AR\_FIELD\_BITOPTION\_LOG\_KEY3)

## defaultValue

The value to apply if a user submits an entry with no field value (applicable for data fields only). The system returns 0 (AR\_DEFAULT\_VALUE\_NONE) if the field has no default. Specify `NULL` for this parameter if you do not want to retrieve the default value.

## permissionList

The list of group access permissions.

## fieldLimit

The value limits for the field and other properties specific to the field's type. See the `ARFieldDefinitionStruct` definition in the `ar.h` file to find the contained structure that applies to the type of field you are creating. Specify `NULL` (or `AR_FIELD_LIMIT_NONE`) for trim and control fields or if you do not want to retrieve the limits and properties.

## displayList

A list of zero or more display properties associated with the field. See “`ARCreateField`” on page 143 for a description of the possible values. The system returns 0 (`AR_DPROP_NONE`) if the field has no display properties. Specify `NULL` for this parameter if you do not want to retrieve this list.

## owner

The owner for the field.

**lastModifiedBy**

The user who last modified the field.

**modifiedDate**

The date that the field was last modified.

**helpText**

The help text associated with the field. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

**changeHistory**

The change history of the field.

**arDocVersion**

The XML document version.

**Return values****status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARGetFieldFromXML.

## ARSetFilterToXML

**Description** Converts a filter to XML.

**Privileges** All users.

**Synopsis** #include "arextern.h"

```
int ARSetFilterToXML(
    ARControlStruct *control,
    ARXMLOutputDoc *xmlOutputDoc,
    XMLDocHdrFtrFlag filterName,
    *executionOrder,
    *workflowConnect,
    *executeOn,
    *enabled,
```

ARBoolean ARNameType
unsigned int ARWorkflowConnectStruct
unsigned int unsigned int

ARQual i fi erStruct	*query,
ARFi l terActi onLi st	*i fActi onLi st,
ARFi l terActi onLi st	*el seActi onLi st,
ARAccessNameType	owner,
ARAccessNameType	LastModi fi edBy,
ARTi mestamp	*modi fi edDate,
char	*hel pText,
char	*changeHi story,
ARPropLi st	*obj PropLi st,
unsi gned i nt	*arDocVersi on,
ARStatusLi st	*status)

**Input  
arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, sessi onl d, and server fields are required.

**xmlOutputDoc**

The XML document output source.

**xmlDocHdrFtrFlag**

The header and footer flag for the XML document output source. If there is only one object in the XML document, set this value to (0).

**filterName**

The filter name. Each filter name must be unique.

**executionOrder**

A value between 0 and 1000 (inclusive) that determines the filter execution order. When multiple filters are associated with a form, the value associated with each filter determines the order in which they are processed (from lowest to highest). Specify `NULL` for this parameter if you do not want to retrieve this value.

**workflowConnect**

The list of form names the filter is linked to. The filter must be associated with a single form or a list of forms that currently exists on the server. Specify `NULL` for this parameter if you do not want to retrieve this value.

## executeOn

A bitmask indicating the form operations that trigger the filter. Specify `NULL` for this parameter if you do not want to retrieve this value.

## enabled

A flag identifying whether the filter is disabled (0) or enabled (1). Specify `NULL` for this parameter if you do not want to retrieve this value.

## query

A qualification that determines whether the filter is executed. The system returns zero (`AR_COND_OP_NONE`) if the filter has no qualification. Specify `NULL` for this parameter if you do not want to retrieve this value.

## ifActionList

The set of actions performed for each entry that matches the criteria defined by the `query` parameter. This list can contain from 1 to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve this value.

## elseActionList

The set of actions performed if the condition defined by the `query` parameter is not satisfied. This list can contain from 0 to 25 actions (limited by `AR_MAX_ACTIONS`). Specify `NULL` for this parameter if you do not want to retrieve this value.

## owner

The owner of the filter.

## lastModifiedBy

The user who last modified the filter.

## modifiedDate

The date that the filter was last modified.

## helpText

The help text associated with the filter. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

## changeHistory

The change history of the filter.

**objPropList**

The object properties of the filter.

**arDocVersion**

The XML document version.

**Return values****status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARGetFilterFromXML.

## ARSetMenuToXML

**Description** Converts menus to XML.

**Privileges** All users.

**Synopsis**

```
#include "arextern.h"

int ARSetMenuToXML(
    ARControlStruct *control,
    ARXMLOutputDoc *xmlOutputDoc,
    XMLDocHdrFtrFlag menuName,
    *refreshCode,
    *menuDefn,
    owner,
    lastModififiedBy,
    *modifiedDate,
    *helpText,
    *changeHistory,
    *objPropList,
    *arDocVersion,
    *status)
```

**Input arguments****control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.

**xmlOutputDoc**

The XML document output source.

**xmlDocHdrFtrFlag**

The header and footer flag for the XML document output source. If there is only one object in the XML document, set this value to (0).

**menuName**

The menu name. Each menu name must be unique.

**refreshCode**

A value indicating when the menu is refreshed. See “ARCreateCharMenu” on page 134 for a description of the possible values. Specify `NULL` for this parameter if you do not want to retrieve this value.

**menuDefn**

The definition of the character menu. Specify `NULL` for this parameter if you do not want to retrieve this value.

**owner**

The menu owner.

**lastModifiedBy**

The user who last modified the menu.

**modifiedDate**

The date that the menu was last modified.

**helpText**

The help text associated with the menu. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

**changeHistory**

The change history of the menu.

**objPropList**

The object properties of the menu.

**arDocVersion**

The XML document version.

**Return values****status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARGetMenuFromXML.

## ARSetSchemaToXML

**Description** Converts schemas (forms) to XML.

**Privileges** All users.

**Synopsis**

```
#include "arextern.h"

int ARSetSchemaToXML(
```

ARControlStruct	*control,
ARXMLOutputDoc	*xmlOutputDoc,
ARBoolEan	xmlDocHdrFtrFlag,
ARNameType	schemaName,
ARCompoundSchema	*compoundSchema,
ARPermissionsOnList	*permissionsOnList,
ARIInternalList	*subAdminGrpList,
AREntryListFieldList	*getListFields,
ARSortList	*sortList,
ARIIndexList	*indexList,
ARArchivelInfoStruct	*archivelInfo,
ARAuditInfoStruct	*auditInfo,
ARNameType	*defaultVUI,
ARIInternalId	*nextFieldID,
unsigned long	*coreVersion,
int	*upgradeVersion,
ARFieldInfoList	*fieldInfoList,
ARVuiInfoList	*vuiInfoList,
ARAccessNameType	owner,

```

ARAccessNameType           lastModifi edBy,
ARTimestamp                *modifi edDate,
char                        *helpText,
char                        *changeHistory,
ARPropList                  *objPropList,
unsigned int                 *arDocVersion,
ARStatusList                *status)

```

## Input arguments

### **control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The `user`, `sessionID`, and `server` fields are required.

### **xmlOutputDoc**

The XML document output source.

### **xmlDocHdrFtrFlag**

The header and footer flag for the XML document output source. If there is only one object in the XML document, set this value to (0).

### **schemaName**

The schema name. Each schema name must be unique.

### **compoundSchema**

The definition of the candidate form or table that contains the candidate field to retrieve.

### **permissionList**

A list of zero or more groups who can access the form. Access to this information is limited to users with AR System administrator privileges only. Specify `NULL` for this parameter if you do not want to retrieve the group list.

### **subAdminGrpList**

A list of zero or more groups who can administer this form (and the associated filters, escalations, and active links). Users must belong to both a specified group *and* the Subadministrator group to obtain these privileges. Specify `NULL` for this parameter if you do not want to retrieve the administrator group list.

## getListField

A list of zero or more fields that identifies the default query list data for retrieving form entries. Specify `NULL` for this parameter if you do not want to retrieve this value.

## sortList

A list of zero or more fields that identifies the default sort order for retrieving form entries. Specify `NULL` for this parameter if you do not want to retrieve the default sort fields.

## indexList

The set of zero or more indexes for the form. Specify `NULL` for this parameter if you do not want to retrieve the index list.

## archiveInfo

If the form is to be archived, this is the archive information for the form. Specify `NULL` for this parameter if you do not want to create or set the archive information. These are the options for archive type:

- 1: The form is selected to have entries copied and deleted to the archive form (AR\_ARCHIVE\_FORM).
- 2: The form is selected to have entries deleted from the main form (AR\_ARCHIVE\_DELETE).
- 32: The form is selected for archive but without attachment field (AR\_ARCHIVE\_NO\_ATTACHMENTS).
- 64: The form is selected for archive without diary fields (AR\_ARCHIVE\_NO\_DIARY).

In addition to the archive type, archive info also stores the form name (if archive type is AR\_ARCHIVE\_FORM), time to trigger the archive, archive qualification criteria, and name of the main form that is being archived.

## auditInfo

If a form is to be audited, this is the audit information for the form. Specify `NULL` for this parameter if you do not want to create or set the audit information. These are the values for audit type:

- 0: The selected form is not to be audited. (AR\_AUDIT\_NONE).

- 1: Audit fields and copy fields on the selected form are copied to the audit shadow form (AR\_AUDI T\_COPY).
- 2: Audit fields and copy fields on the selected form are copied to the audit log form (AR\_AUDI T\_LOG).

In addition to the audit type, audi tlnfo also stores the form name (if audit type is AR\_AUDI T\_COPY or AR\_AUDI T\_LOG) and the audit qualification criteria.

#### **defaultVui**

The label for the default view.

#### **nextFieldID**

The next default field identification number, which the server generates if users do not specify it.

#### **coreVersion**

The version number of the core fields.

#### **upgradeVersion**

The upgrade version for autogenerated internal server schemas. To disable this option, set it to zero (0).

#### **fieldInfoList**

The list of schema fields.

#### **vuiInfoList**

The list of schema views.

#### **owner**

The schema owner.

#### **lastModifiedBy**

The user who last modified the schema.

#### **modifiedDate**

The date that the schema was last modified.

#### **helpText**

The help text associated with the schema. Specify `NULL` for this parameter if you do not want to retrieve the help text (which is useful if you are calling this function to verify whether an instance of this object exists).

**changeHistory**

The change history of the schema.

**objPropList**

The object properties of the schema.

**arDocVersion**

The XML document version.

**Return values****status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARGetSchemaFromXML.

## ARSetVUIToXML

**Description** Converts a view to an XML document.

**Privileges** All users.

**Synopsis** #include "arextern.h"

```
int ARSetVUIToXML(
    ARControlStruct *control,
    ARXMLOutputDoc *xmlOutputDoc,
    ARBool docHdrFtrMask,
    ARNameType schemaName,
    ARVuiInfoList *vuiInfoList,
    ARFieldInfoList *fieldInfoList,
    ARTimestamp modifiedDate,
    unsigned int arDocVersion,
    ARStatusList *status)
```

**Input arguments**

**control**

The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.

**xmlOutputDoc**

The XML document output source.

**xmlDocHdrFtrMask**

The header and footer flag for the XML document output source. If there is only one object in the XML document, set this value to (0).

**schemaName**

The name of the view. Each view name must be unique.

**vuiInfoList**

The list of schema views.

**fieldInfoList**

The list of view fields.

**modifiedDate**

The date that the view was last modified.

**arDocVersion**

The XML document version.

**Return values****status**

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

**See also** ARGetVUIFromXML.

## ARSetXMLDocFooterToXML

**Description** Converts document footers to XML.

**Privileges** All users.

**Synopsis**

```
#include "arextern.h"

int ARSetXMLDocFooterToXML(
    ARControlStruct          *control,
    ARXMLOutputDoc            *xmlOutputDoc,
    ARStatusList               *status)
```

---

<b>Input arguments</b>	<p><b>control</b> The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.</p> <p><b>xmlOutputDoc</b> The XML document output source.</p>
<b>Return values</b>	<p><b>status</b> A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the <i>Error Messages</i> guide.</p>
<b>See also</b>	ARSetXMLDocHeaderToXML.

## ARSetXMLDocHeaderToXML

<b>Description</b>	Converts document headers to XML.
<b>Privileges</b>	All users.
<b>Synopsis</b>	<pre>#include "arextern.h"  int ARSetXMLDocHeaderToXML(     ARControlStruct          *control,     ARXMLOutputDoc            *xmlOutputDoc,     ARStatusList               *status)</pre>

<b>Input arguments</b>	<p><b>control</b> The control record for the operation. It contains information about the user requesting the operation, where that operation is to be performed, and which session is used to perform it. The user, session, and server fields are required.</p> <p><b>xmlOutputDoc</b> The XML document output source.</p>
------------------------	--

Return values [status](#)

A list of zero or more notes, warnings, or errors generated from this function call. See “Error checking” on page 441 for a description of all possible values or see the *Error Messages* guide.

See also [ARSetXMLDocFooterToXML](#).



# A Migrating to the Action Request System 7.0 API

This appendix summarizes the changes since the 6.3 release.

AR System 7.0 is backward-compatible, supporting all requests from client applications that use the 6.3, 6.0, 5.1, 5.0, 4.x, or 3.2 API libraries. If you continue to link to one of these libraries, you do not need to make any changes to continue running your existing programs against AR System 7.0 servers. If you link to the new 7.0 API libraries, you must make changes to your programs and recompile them. The main program structure and processing, however, need not change.

# Existing program changes

The following changes might affect your existing API programs.

## New and changed C data types

The following table contains C data type changes. For more information, see “C data types” on page 41.

Affected C data type	Change or addition
typedef char ARAAuthType[AR_MAX_AUTH_SI_ZE + 1]	Changed type. AR_MAX_AUTH_SI_ZE was increased from 1024 to 2047.
typedef char ArAccessNameType[AR_MAX_ACCESS_NAME_SI_ZE + 1]	Changed type. AR_MAX_ACCESS_NAME_SI_ZE was increased from 30 to 254.
typedef char AREncryptedPasswordType [AR_MAX_ENCRYPTED_PASSWORD_SI_ZE + 1]	New type. AR_MAX_ENCRYPTED_PASSWORD_SI_ZE is defined as 120.
typedef char ARPasswordType[AR_MAX_PASSWORD_SI_ZE + 1]	New type. AR_MAX_PASSWORD_SI_ZE is defined as 30.
typedef char ARTablenameType[AR_MAX_TABLENAME_SI_ZE + 1]	Changed type. AR_MAX_TABLENAME_SI_ZE was increased from 256 to 2047.

## Changed parameters

The following table contains C API parameter changes and the calls that they affect.

Affected calls	Parameter	Change or addition
ARCreateField, ARGetField, ARGetFieldFromXML, ARGetMultipleFields, ARSetField, ARSetFieldToXML	fieldOption	New parameter.
ARCreateSchema, ARGetSchema, ARGetSchemaFromXML, ARSetSchema, ARSetSchemaToXML	auditInfo	New parameter.
ARSetField	setFieldOptions	New parameter.

## Data structure changes and additions

This section describes data structure changes and additions.

### New structures

The following structures are new for AR System 7.0:

- ARAudi tI nfoStruct contains information about auditing type, form names, and qualification criteria for audited forms.
- ARAudi tI nfoList is a list of ARAudi tI nfoStruct items.

## Changed structures

The following table contains C API data structure changes for AR System.

Affected structure	Element	Change or addition
ARAttachListStruct	fullTextOptions	This element was added to the structure.
ARComplexEntryOptions	startAt	This element was added to the structure.
ARControlStruct	AuthenticationString	The type changed from char to ARAuthType.
ARControlStruct	Password	The type changed from ARAccessNameType to ARPasswordType.
ARConnectionStruct	Password	The type changed from ARAccessNameType to ARPasswordType.
ARFielddInfoStruct	fieldOption	This element was added to the structure.
ARLocalizationalInfo	timeZone	The size changed from AR_MAX_LANG_SIZE + 1 to AR_MAX_LOCALE_SIZE + 1.
ARWorkflowLockStruct	lockKey	The type changed from ARAccessNameType to ARPasswordType.

## New server information options

The following server information options have been added. For more information, see “ARGetServerInfo” on page 322 and “ARSetServerInfo” on page 403.

- AR\_SERVER\_INFO\_ADMIN\_OP\_PROGRESS
- AR\_SERVER\_INFO\_ADMIN\_OP\_TRACKING
- AR\_SERVER\_INFO\_DISABLE\_NON\_UNICODE\_CLIENTS
- AR\_SERVER\_INFO\_DSO\_MAX\_QUERY\_SIZE
- AR\_SERVER\_INFO\_EA\_GROUP\_MAPPING
- AR\_SERVER\_INFO\_EA\_IGNORE\_EXCESS\_GROUPS
- AR\_SERVER\_INFO\_ERROR\_EXCEPTION\_LIST
- AR\_SERVER\_INFO\_EXCEPTION\_OPTIONS
- AR\_SERVER\_INFO\_FT\_CASE\_SENSITIVITY

- AR\_SERVER\_I NFO\_FT\_COLLECTI ON\_DI R
- AR\_SERVER\_I NFO\_FT\_CONFI GURATI ON\_DI R
- AR\_SERVER\_I NFO\_FT\_DI SABLE\_SEARCH
- AR\_SERVER\_I NFO\_FT\_OPTI MI ZE\_THRESHOLD
- AR\_SERVER\_I NFO\_FT\_RECOVERY\_I NTERVAL
- AR\_SERVER\_I NFO\_FT\_RELINDEX
- AR\_SERVER\_I NFO\_FT\_SEARCH\_MATCH\_OP
- AR\_SERVER\_I NFO\_FT\_STOP\_WORDS
- AR\_SERVER\_I NFO\_FT\_TEMP\_DI R
- AR\_SERVER\_I NFO\_FT\_THRESHOLD\_HI GH
- AR\_SERVER\_I NFO\_FT\_THRESHOLD\_LOW
- AR\_SERVER\_I NFO\_FTINDEXER\_LOG\_FI LE
- AR\_SERVER\_I NFO\_GUESTS\_RESTRI CT\_READ
- AR\_SERVER\_I NFO\_MAX\_PASSWORD\_ATTEMPTS
- AR\_SERVER\_I NFO\_NEXT\_I D\_BLOCK\_SI ZE
- AR\_SERVER\_I NFO\_NOTI FY\_WEB\_PATH
- AR\_SERVER\_I NFO\_ORACLE\_CLOB\_STORE\_I NROW
- AR\_SERVER\_I NFO\_PLUGI N\_DEFAULT\_TI MOUT
- AR\_SERVER\_I NFO\_PLUGI N\_LOG\_LEVEL
- AR\_SERVER\_I NFO\_PREF\_SERVER\_OPTI ON



# Index

## A

access control functions, listed 125  
Action Request System External Authentication (AREA)  
    API function calls 505  
    data structure 502  
    functions  
        FreeCallback 505  
        NeedToSyncCallback 506  
        VerifyLoginCallback 506  
    plug-in  
        authentication services 483  
        described 25  
        running 490  
    response data structure 504  
Action Request System Filter (ARF) plug-in  
    API function calls 526  
    API structure 526  
    described 25, 484  
    example files 484  
    running 494  
active links  
    functions for XML API calls 531  
    functions, listed 122  
    object properties 77  
    structures for 77  
ActiveLinkFromXML function 536  
ActiveLinkToXML function 565  
alert functions, listed 125  
allocated memory, freeing 112

application object  
    GetActiveForm 461  
    GetFormList 459  
    GetServerList 458  
    HasDefaultSession 462  
    LoadForm 460  
    Login 457  
    Logout 458  
    OpenForm 459  
    OpenGuide 462  
    RunMacro 463  
application object, ICOMAppObj 456  
AR System  
    objects, transforming 530  
    plug-ins 481  
ar.cfg and ar.conf files 485  
ARActiveLinkActionList structure 83  
ARActiveLinkActionStruct structure 85  
ARArithOpAssignStruct structure 90  
ARArithOpStruct structure 59  
ARAssignFieldStruct structure 88  
ARAssignFilterApiStruct structure 94  
ARAssignSQLStruct structure 93  
ARAssignStruct structure 87  
ARAutomationStruct structure 96  
ARBEGINBulkEntryTransaction function 127  
ARCharLimitsStruct structure 64  
ARCharMenuItemStruct structure 101  
ARCharMenuStruct structure 101

architecture  
     C API 25  
     Java API 25  
     plug-in 27

ARColumnLimitsStruct structure 65

ARCOMMMethodParamStruct structure 98

ARCOMMMethodStruct structure 96

ARCompoundSchema structure 61

ARCOMValueStruct structure 98

ARContainerInfoList structure 102

ARControlStruct structure 46

ARCoordList data type 67

ARCreate functions  
     ActiveLink 128  
     AlertEventLists 132  
     CharMenu 134  
     Container 136  
     diary values and 55  
     Entry 139  
     Escalation 140  
     Field 143  
     Filter 160  
     License 163  
     Schema 164  
     SupportFile 168  
     VUI 169

ARDateLimitsStruct structure 65

ARDateToJulianDate function 175

ARDBC  
     API  
         described 508  
         function calls 509  
     functions  
         CommitTransaction 509  
         CreateEntry 510  
         DeleteEntry 511  
         GetEntry 513  
         GetEntryBLOB 515  
         GetEntryStatistics 516  
         GetListEntryWithFields 519  
         GetListSchemas 521  
         GetMultipleFields 522  
         RollbackTransaction 523  
         SetEntry 524

ARDBC (continued)  
     plug-in  
         described 25, 32, 36  
         running 492

ARDDEStruct structure 93

ARDecimalLimitsStruct structure 65

ARDecode functions  
     AlertMessage 176  
     ARAssignStruct 179  
     ARQualifierStruct 180  
     Diary 55, 181  
     StatusHistory 182

ARDelete functions  
     ActiveLink 183  
     CharMenu 184  
     Container 185  
     Entry 186  
     Escalation 187  
     Field 189  
     Filter 190  
     License 191  
     MultipleFields 192  
     Schema 193  
     SupportFile 195  
     VUI 196

ARDeregisterForAlerts function 197

ARDiaryLimitStruct structure 64

ARDisplayInstanceList structure 65

ARDisplayInstanceStruct structure 65, 66

AREA  
     API function calls 505  
     data structure 502  
     functions  
         FreeCallback 505  
         NeedToSyncCallback 506  
         VerifyLoginCallback 506  
     plug-in  
         authentication services 483  
         described 25  
         running 490  
         response data structure 504

AREAreaResponseStruct structure 502

AREEncode functions  
     ARAssignStruct 198  
     ARQualifierStruct 199

AREncode functions (continued)  
    Diary 200  
    StatusHistory 201  
AREndBulkEntryTransaction function 202  
AREEntryIdListList structure, and passing entry  
    IDs 76  
AREEntryListFieldList structure 73  
AREEntryListFieldStruct structure 73  
AREEntryListFieldValueList structure 71  
AREEntryListList structure 71  
AREEntryListStruct structure 71  
ARExecuteProcess function 203  
AREExpandCharMenu function 204  
ARExport function 206  
ARExportLicense function 210  
ARExtReferenceStruct structure 109  
ARF plug-in  
    API  
        function calls 526  
        structure 526  
    described 25  
    example files 484  
    running 494  
ARFieldAssignList structure 86  
ARFieldAssignStruct structure 86  
ARFieldLimitStruct structure 63, 64  
ARFieldMappingStruct structure 68, 69  
ARFieldValueList structure 75  
ARFieldValueListList structure 76  
ARFieldValueOrArithStruct structure 59  
ARFieldValueStruct structure 72  
ARFilterActionList structure 43, 81  
ARFilterApiCall function 526  
ARFunctionAssignStruct structure 90  
ARGet functions  
    ActiveLink 211  
    ActiveLinkFromXML 536  
    AlertCount 214  
    ApplicationState 215  
    CharMenu 216  
    Container 219  
    ContainerFromXML 539  
    CurrencyRatio 222  
    DSOMappingFromXML 542  
    DSOPoolFromXML 545

ARGet functions (continued)  
    Entry 223  
    EntryBLOB 225  
    EntryBlock 227  
    EntryStatistics 228  
    Escalation 230  
    EscalationFromXML 547  
    Field 233  
    FieldFromXML 550  
    Filter 237  
    FilterFromXML 553  
    ListActiveLink 240  
    ListAlertUser 241  
    ListApplicationState 242  
    ListCharMenu 243  
    ListContainer 244  
    ListEntry 246  
    ListEntryBlock 249  
    ListEntryWithFields 252  
    ListEscalation 254  
    ListExtSchemaCandidates 256  
    ListField 257  
    ListFilter 258  
    ListGroup 260  
    ListLicense 261  
    ListRole 262  
    ListSchema 263  
    ListSchemaWithAlias 266  
    ListServer 269  
    ListSQL 270  
    ListSupportFile 271  
    ListUser 273  
    ListVUI 274  
    ListXMLObjects 556  
    LocalizedValue 275  
    MenuFromXML 557  
    MultipleActiveLinks 276  
    MultipleCharMenus 281  
    MultipleContainers 284  
    MultipleCurrencyRatioSets 288  
    MultipleEntries 293  
    MultipleEntryPoints 289  
    MultipleEscalations 295  
    MultipleExtFieldCandidates 298  
    MultipleFields 299

- ARGet functions (continued)
  - MultipleFilters 304
  - MultipleLocalizedValues 308
  - MultipleSchemas 309
  - MultipleVUIs 314
  - Schema 316
  - SchemaFromXML 559
  - ServerInfo 218, 320, 322
  - ServerStatistics 351
  - SessionConfiguration 349
  - SupportFile 358
  - TextForErrorMessage 360
  - VUI 360
  - VUIFromXML 563
- ARGetFromXML function 534
- ARGetListXMLObjects function 535
- ARIImport function 363
- ARIImportLicense function 366
- ARInitialization function 367
- ARIIntegerLimitsStruct structure 64
- arithmetic result values, assigning 90
- ARJoinMappingStruct structure 69, 70
- ARJoinSchema structure 62
- ARJulianDateToDate function 368
- ARLoadARQualifierStruct function 369
- ARMergeEntry function
  - described 370
  - diary values and 55
- ARParseXMLDocument function 533, 564
- ARPermissionList structure 51
- ARPlugin
  - accessing 490
  - functions
    - CreateInstance 498
    - DeleteInstance 498
    - Event 499
    - Identify 499
    - Initialization 500
    - SetProperties 501
    - Termination 502
  - log file 488
  - program 487
  - running 489
- ARPluginEvent function 499
- ARPropList structure 65, 66
- ARPropStruct structure 66, 77
- ARPushFieldsList structure 94
- ARQualifierStruct structure 55, 57
- ARRealLimitsStruct structure 64
- ARReferenceStruct structure 105
- ARRegisterForAlerts function 372
- ARRelOpStruct structure 58
- ARRPC environment variable 405
- ARSet functions
  - ActiveLink 373
  - ActiveLinkToXML 565
  - ApplicationState 378
  - CharMenu 379
  - Container 381
  - ContainerToXML 569
  - diary values and 55
  - DSOMappingToXML 572
  - DSOPoolToXML 575
  - Entry 384
  - Escalation 386
  - EscalationToXML 577
  - Field 389
  - FieldToXML 580
  - Filter 393
  - FilterToXML 583
  - ImpersonatedUser 397
  - Logging 397
  - MenuToXML 586
  - Schema 399
  - SchemaToXML 588
  - ServerInfo 403
  - ServerPort 405
  - SessionConfiguration 406
  - SupportFile 408
  - VUI 410
  - VUIToXML 592
  - XMDocFooterToXML 593
  - XMDocHeaderToXML 594
- ARSetFooterToXML function 535
- ARSetHeaderToXML function 534
- ARSetToXML function 534
- ARSignal function 412
- ARSortList structure 74
- ARSortStruct structure 74
- ARStatHistoryValue structure 60

ARStatusList structure 48, 49  
 ARStatusStruct structure 49, 51  
 arstruct.h file 75, 170  
 ARStructItemList structure 109  
 ARStructItemStruct structure 111  
 ARTableLimitsStruct structure 65  
 ARTCPPORT environment variable 405  
 ARTermination function 413  
 aruser.log file 502  
 ARValidateFormCache function 413  
 ARValidateLicense function 416  
 ARValidateMultipleLicenses function 417  
 ARValueStruct structure 53  
 ARVerifyUser function 418  
 ARXMLEOutputDoc structure 534  
 authentication services, using with AREA plug-in 25, 483  
 automation  
     actions 95  
     building an automation client 451  
     handling errors 452  
     interfaces  
         ICOMField 469  
         ICOMQueryResult 470  
         ICOMQueryResultSet 471  
         Remedy User object 455  
     invoking member methods 451  
     object model 455  
     sample program 452  
     type library 450

**B**

bounding boxes 67  
 bulk entry functions, listed 125

**C**

C API  
     architecture 25  
     C data types 41  
     calls, components 28  
     clients, multithreaded 445  
     data relationships 40  
     overview 22

C API (continued)  
     package contents 32  
     program design tips 445  
     program structure 434  
     programming, when to use 28  
     programs, executing in workflow 443  
     terms vs. Remedy User terms 29  
 C API sample source code 36  
 changed parameters 599  
 character menus  
     defined 100  
     functions for XML API calls 532  
     functions, listed 124  
     search style, populating 484  
 Close form object 465  
 CommitTransaction function 509  
 common tasks, performing 436  
 compiler information 35  
 configuration files, plug-ins 485  
 ContainerFromXML function 539  
 containers  
     data structures 102  
     functions  
         for XML API calls 532  
         listed 122  
     retrieving lists of 102  
     retrieving or modifying 103  
 ContainerToXML function 569  
 control parameter, and ARControlStruct 45  
 coordinate list 67  
 Create Filter form, and ARF plug-in 494  
 CreateEntry function 510  
 CreateInstance function 498  
 currency functions, listed 126

**D**

data  
     fields, and value limits 64  
     relationships 40  
     types, C 41  
     types, changed 598  
 data structures. *See structures*  
 date sources, external 484  
 DDE result value, assigning 93

debugging  
    print.c routines 476  
    using driver program 475  
    using log files 474, 483

DeleteEntry function 511

DeleteInstance function 498

dInstanceList parameter, and display properties 66

Disable field object 470

DLL  
    plug-ins 485  
    required DLLs 33

driver program  
    scripts 480  
    using  
        described 475  
        from the command line 478

DSO, functions for XML API calls 532

DSOMappingFromXML function 542

DSOMappingToXML function 572

DSOPoolFromXML function 545

DSOPoolToXML function 575

**E**

encryption functions 93

entries  
    ARFieldValueStruct 75  
    data structures 70  
    entry functions, listed 123  
    modifying 75  
    retrieving  
        as concatenated strings 71  
        as field/value pairs 71  
        entry lists 70  
        individual entries 75  
        multiple entries 76

entryList parameter  
    and ARGetListEntry 71  
    and ARGetListEntryWithFields 71

environment, plug-ins 485

error  
    checking 441  
    handling, in automation 452

EscalationFromXML function 547

escalations  
functions  
    for XML API calls 532  
    listed 123  
    object properties 77  
    structures for 77

EscalationToXML function 577

exporting objects 109

external  
    data sources 484  
    references, defining 109

external data manipulation functions, listed 126

**F**

field object  
    described 469  
    Disable 470  
    MakeReadWrite 470  
    MakeVisible 469

FieldFromXML function 550

fieldMap parameter, in ARFieldMappingStruct 68

fields  
    and structure types 63  
    ARFieldValueList 75  
    ARJoinMappingStruct 69  
    defining field limits 63  
    field functions, listed 123  
    field retrieval, specifying 73  
    functions for XML API calls 532  
    specifying 440

FieldToXML function 580

files  
    header 32  
    include 32  
    library 33

Filter API  
    example files 484  
    result value, assigning 94

FilterFromXML function 553

filters  
    functions for XML API calls 532  
    functions, listed 124  
    object properties 77  
    structures for 77

FilterToXML function 583

form object

- Close 465

- described 463

- GetField 466

- GetFieldById 466

- GetFormName 468

- GetServerName 468

- GiveFieldFocus 467

- MakeVisible 466

- Modify 465

- Query 467

- Submit 465

formats, XML 114

forms

- described 61

- functions, listed 124

FreeAR

- function 419

- header files 32

FreeARXMLParsedStream function 534, 535

FreeCallback function 505

freeing allocated memory 112

freeStruct parameter 431

function return value, assigning 90

functions, listed

- access control 125

- active link 122

- alert 125

- bulk entry 125

- character menus 124

- containers 122

- currency 126

- entries 123

- escalation 123

- external data manipulation 126

- fields 123

- filters 124

- forms (schemas) 124

- housekeeping 126

- licensing 126

- object definition 126

- object manipulation 121, 531

- schemas (forms) 124

- server process 127

functions, listed (continued)

- support files 124

- text manipulation, listed 127

- VUI 125

## G

GetActiveForm application object 461

GetEntry function 513

GetEntryBLOB entry 515

GetEntryStatistics function 516

GetField form object 466

GetFieldById form object 466

GetFormList application object 459

GetFormName form object 468

GetListEntryWithFields function 519

GetListSchemas function 521

GetMultipleFields function 522

GetServerList application object 458

GetServerName form object 468

GiveFieldFocus form object 467

global information, protection 489

groups, privileges 137

## H

HasDefaultSession application object 462

header files

- arstruct.h 75

- include directory 32

- include files 32

housekeeping functions, listed 126

## I

ICOMAppObj application object

- described 456

- GetActiveForm 461

- GetFormList 459

- GetServerList 458

- HasDefaultSession 462

- LoadForm 460

- Login 457

- Logout 458

- OpenForm 459

- OpenGuide 462

- RunMacro 463

ICOMField field object  
described 469  
Disable 470  
MakeReadWrite 470  
MakeVisible 469

ICOMFormWnd form object  
Close 465  
described 463  
GetField 466  
GetFieldById 466  
GetFormName 468  
GetServerName 468  
GiveFieldFocus 467  
GiveFieldFocusById 467  
MakeVisible 466  
Modify 465  
Query 467  
Submit 465

ICOMQueryResult object 470

ICOMQueryResultSet object  
described 471  
Item 471

Identify function 499

impersonating users 397

importing objects 109

include files, header files 32

Initialization function 500

installing plug-ins 485

Item query result set object 471

**J**

Java API  
architecture 25  
comparison with C API 23  
overview 23  
reasons for choosing 23  
requirements 24

**K**

keywords, specifying 440

**L**

LDAP plug-ins 483

library  
automation type 450  
files 33  
licensing functions, listed 126  
link information 36  
lists  
data structures 42  
described 42

ListXMLObjects function 556

LoadForm application object 460

log files, aruser.log 502

logging  
AR System activity 474  
plug-in information 496  
plug-in logging levels 496

Login application object 457

login information 45

Logout application object 458

**M**

makefiles  
samples 486  
working with 36

MakeReadWrite field object 470

MakeVisible field object 469

MakeVisible form object 466

malloc function 112

mapping fields, schemas 68

MenuFromXML function 557

MenuToXML function 586

messages, user 502

Microsoft Developer Studio, sample projects 486

migrating to new AR System version 597

Modify form object 465

Multithreaded C API clients 445

**N**

naming conventions, plug-ins 488

NeedToSyncCallback function 506

network directory services 483

new  
parameters 599

**O**

## object

- actions, server 81
- API function calls 531
- importing objects 109
- manipulation functions, listed 121
- properties, server 77
- property tags, server 79
- relationships, high-level 43
- object API functions, manipulation 531
- object definition functions, listed 126
- OpenForm application object 459
- OpenGuide application object 462
- operands, defining for a relational operation 59
- overview
  - C API 22
  - Java API 23

**P**

## parameters

- changed 599
- new 599

## password

- number of retries 337

## permission information 51

## plug-ins

- Action Request System External Authentication (AREA) 502
- API calls 497
- AR System 481
- architecture 27
- ARF 25
- arplugin program 487
- configuration files 485
- conventions 488
- creating 484, 485
- described 24, 483
- DLL 485
- environment, setting up 485
- global information protection 489
- input and output value conventions 488
- installing 485
- LDAP 483
- log file messages 488

## plug-ins (continued)

- logging facility
  - described 496
  - Log function 496
  - logging levels 496
- makefile samples 486
- memory management 488
- naming conventions 488
- program structure 486
- resource protection 489
- shared object library 485
- using 484

## print.c routines, using 476

## privileges

- admngrpListList 286, 312
- user 139

## program

- changes, to existing API programs 598
- design tips 445

## Push Fields action 94, 443

**Q**

## qualifications

- conditional operators 57
- dynamic 57
- relational operators 58
- representing 55

## qualifier parameter, and entry retrieval 70

## query form object 467

## query result object, described 470

## query result set object

- described 471
- Item 471
- query 471

**R**

## Remedy User and C API terms 29

## representing values 53

## retries for passwords 337

## RollbackTransaction function 523

## Run Process action 444

## RunMacro application object 463

**S**

sample programs  
 automation 452  
 driver 28, 475, 478

SchemaFromXML function 559

schemas  
 ARJoinSchema 62  
 described 61  
 field values, assigning 87  
 functions for XML API calls 532  
 functions, listed 124  
 mapping fields 68

SchemaToXML function 588

search-style character menus, populating 25, 484

server managed object property tags 80

server object  
 actions 81  
 properties 77  
 property tags 79  
 types 77

server process functions, listed 127

session information 45

Set Fields action 86, 443

SetEntry function 524

SetProperties function 501

shared object library, plug-ins 485

sortList parameter, and sorting lists of entries 74

Source Code Control (SCC), system integration  
 support 78

source code, C API sample 36

SQL result values, assigning 93

status information, retrieving 48

structures  
 ARActiveLinkActionList 83  
 ARActiveLinkActionStruct 85  
 ARArithOpAssignStruct 90  
 ARArithOpStruct 59  
 ARAssignFieldStruct 88  
 ARAssignFilterApiStruct 94  
 ARAssignSQLStruct 93  
 ARAssignStruct 87  
 ARAttachLimitsStruct 65  
 ARAutomationStruct 96  
 ARCharLimitsStruct 64  
 ARCharMenuItemStruct structure 101

structures (continued)

ARCharMenuStruct 101  
 ARColumnLimitsStruct 65  
 ARCOMMMethodParmStruct 98  
 ARCOMMMethodStruct 96  
 ARCompoundSchema 61  
 ARCOMValueStruct 98  
 ARContainerInfoList 102  
 ARControlStruct 46  
 ARDBC API 508  
 ARDDEStruct 93  
 ARDecimalLimitsStruct 65  
 ARDiaryLimitStruct 64  
 ARDisplayInstanceList 65  
 ARDisplayInstanceStruct 65, 66  
 AREA ResponseStruct 502  
 AREAResponseStruct 502  
 AREEntryListFieldList 73  
 AREEntryListFieldStruct 73  
 AREEntryListFieldValueList 71  
 AREEntryListList 71  
 AREEntryListStruct 71  
 ARExternalReferenceStruct 109  
 ARFieldAssignList 86  
 ARFieldAssignStruct 86  
 ARFieldLimitStruct 63, 64  
 ARFieldMappingStruct 68, 69  
 ARFieldValueList 75  
 ARFieldValueOrArithStruct 59  
 ARFieldValueStruct 72, 75  
 ARFilterActionList 43, 81  
 ARFunctionAssignStruct 90  
 ARIntegerLimitsStruct 64  
 ARJoinMappingStruct 69, 70  
 ARJoinSchema 62  
 ARPermissionList 51  
 ARPropList 65, 66  
 ARPropStruct 66, 77  
 ARPushFieldsList 94  
 ARQualifierStruct 55, 57  
 ARReferenceStruct 105  
 ARSortList 74  
 ARSortStruct 74  
 ARStatHistoryValue 60  
 ARStatusList 48, 49

## structures (continued)

- ARStatusStruct 49, 51
- ARStructItemList 109
- ARStructItemStruct 111
- ARTableLimitsStruct 65
- ARValueStruct 53
- ARXMLOutputDoc 534
- changes 600
- help functions, listed 126
- listing AR System 535
- Lists 42
- new 599

Submit form object 465

support file functions, listed 124

**T**

Termination function 502

terms, C API vs. Remedy User 29

text manipulation functions, listed 127

tips, program design 445

type information, accessing 450

type library file, including 451

**U**

user messages 502

users, access privileges 139, 266

users, impersonating 397

**V**

value parameter 431

value scaling, coordinating 68

values
 

- data structures for 55
- representing 53

VerifyLoginCallback function 506

VUI functions
 

- for XML API calls 532
- listed 125

VUIFromXML function 563

VUIToXML function 592

**X**

XML
 

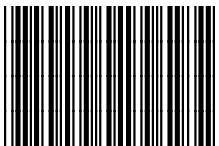
- documents, transforming 533
- formats 114

## XML (continued)

- schema 530
- schema definition files 531
- transforming AR System structures, described 534
- transforming objects, overview 530
- using object XML function calls 533
- XMDocFooterToXML function 593
- XMDocHeaderToXML function 594
- XSD files 530







\*58480\*