# ACME Developer's Guide

**June 2010**

This guide describes how to write an Authentication and Credential Management Extension (ACME) agent to provide customized authentication on the OpenVMS operating system. It also discusses writing an OpenVMS Alpha persona extension to support your own credentials to accompany OpenVMS credentials.

| | |
|---|---|
| **Revision/Update Information:** | This manual supersedes the *ACME Developer's Guide*, Version 7.3-1. |
| **Software Version:** | OpenVMS Version 8.4 for Integrity servers<br>OpenVMS Alpha Version 8.4 |

**Hewlett-Packard Company**
**Palo Alto, California**

ZK6199

The HP OpenVMS documentation set is available on CD-ROM.

This document was prepared using DECdocument, Version 3.3-1b.

# Contents

## 3 Testing and Debugging Your ACME Agent



## 4 ACME Agent Data Structure

## 5 ACME Agent Control Callout Routines



## 6 ACME Authentication and Password Callout Routines

## 10 Persona Extensions Overview



## 11 Persona Extensions Entry Points

## 12 Connecting Your Persona Extension Image to the OpenVMS Executive



## A SYS$ACM[W] Data Structures



## B ACME Agent Interface Data Structures



## C Persona Extension Interface Data Structures

## Figures



## Tables

# Preface

## Intended Audience

This guide is intended for individuals who want to provide authentication services in their software.

## Document Structure

This guide is organized as follows:

- Chapter 1 provides an overview of the ACME client and ACME server processes and how they interact to provide authentication services.
- Chapter 2 discusses programming guidelines you should be familiar with before you begin writing your own ACME agent shareable images.
- Chapter 3 provides tips for testing your ACME agent shareable images.
- Chapter 4 describes the agent data structure.
- Chapter 5 lists ACME callout routines for agent control.
- Chapter 6 lists ACME callout routines for processing Authentication and Password Change requests.
- Chapter 7 lists ACME callout routines for event and query request processing.
- Chapter 8 lists ACME status codes.
- Chapter 9 lists ACME callback routines.
- Chapter 10 gives an overview of persona extensions.
- Chapter 11 describes persona extension entry points.
- Chapter 12 describes connecting your persona extension image to the OpenVMS executive.
- Appendix A describes SYS$ACM[W] data structures.
- Appendix B describes ACME agent interface data structures.
- Appendix C describes persona extension interface data structures.
- Appendix D provides an ACME programming example.
- The Glossary lists terms you need to know to write an ACME agent.

## Related Documents

If you are writing software to be a *consumer* of authentication services, you should read the description of $ACM in the *HP OpenVMS System Services Reference Manual*.

For additional information about HP OpenVMS products and services, see:

`http://www.hp.com/go/openvms`

## Reader's Comments

HP welcomes your comments on this manual. Please send your comments or suggestions to:

`openvmsdoc@hp.com`

## How To Order Additional Documentation

For information about how to order additional documentation, see:

`http://www.hp.com/go/openvms/doc/order`

## Conventions

The following conventions are used in this manual:

| | |
|---|---|
| Ctrl/*x* | A sequence such as Ctrl/*x* indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button. |
| PF1 *x* | A sequence such as PF1 *x* indicates that you must first press and release the key labeled PF1 and then press and release another key or a pointing device button. |
| Return | In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)<br>In the HTML version of this document, this convention appears as brackets, rather than a box. |
| . . . | A horizontal ellipsis in examples indicates one of the following possibilities:<br>• Additional optional arguments in a statement have been omitted.<br>• The preceding item or items can be repeated one or more times.<br>• Additional parameters, values, or other information can be entered. |
| .<br>.<br>. | A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed. |
| ( ) | In command format descriptions, parentheses indicate that you must enclose choices in parentheses if you specify more than one. |

| | |
|---|---|
| [ ] | In command format descriptions, brackets indicate optional choices. You can choose one or more items or no items. Do not type the brackets on the command line. However, you must include the brackets in the syntax for OpenVMS directory specifications and for a substring specification in an assignment statement. |
| \| | In command format descriptions, vertical bars separate choices within brackets or braces. Within brackets, the choices are optional; within braces, at least one choice is required. Do not type the vertical bars on the command line. |
| { } | In command format descriptions, braces indicate required choices; you must choose at least one of the items listed. Do not type the braces on the command line. |
| **bold type** | Bold type represents the introduction of a new term. It also represents the name of an argument, an attribute, or a reason. |
| *italic type* | Italic type indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where *dd* represents the predefined code for the device type). |
| UPPERCASE TYPE | Uppercase type indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege. |
| `Example` | This typeface indicates code examples, command examples, and interactive screen displays. In text, this type also identifies URLs, UNIX commands and pathnames, PC-based commands and folders, and certain elements of the C programming language. |
| - | A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line. |
| numbers | All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated. |

# 1
# Overview

The Authentication and Credential Management Extensions (ACME) subsystem was designed with two objectives:

- Provide application programs with a native system service for user authentication ($ACM[W]). This service supports passwords as the default authentication mechanism, but is able to support additional mechanisms through the use of customized policy providers (ACME agents).
- Provide a means to add site-specific or third-party authentication policy providers (ACME agents) accessed through the $ACM system service.

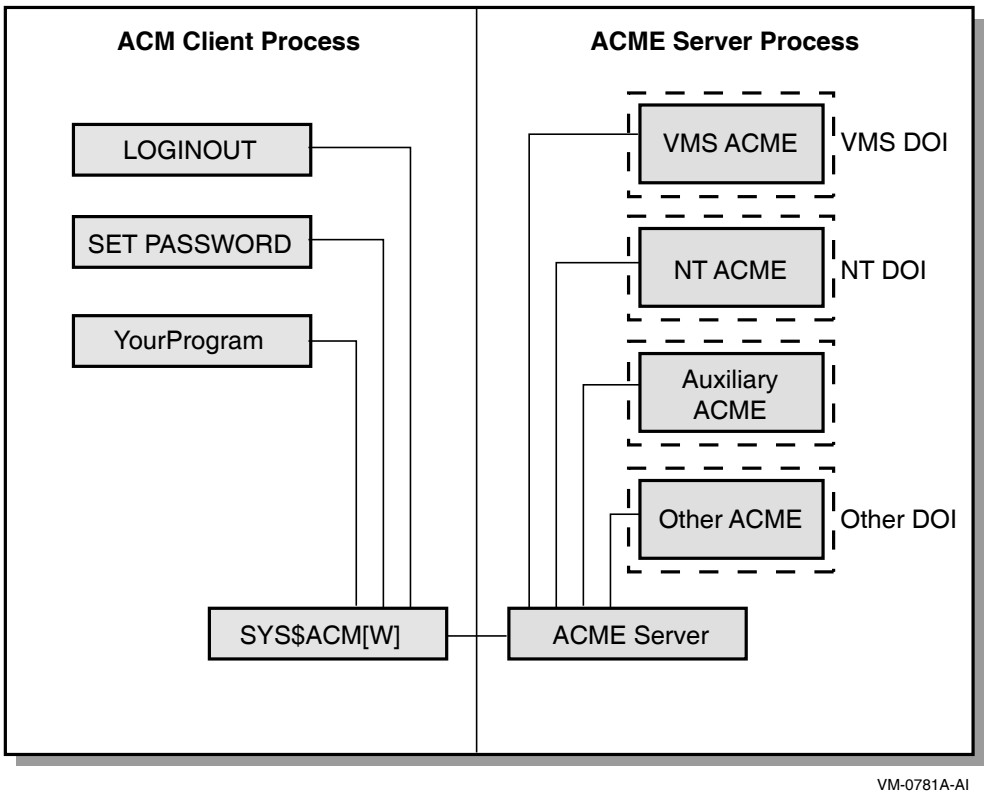
When an application calls $ACM, the ACME_SERVER process dispatches the requests to one or more agents that have been configured by the system manager. An agent drives the input/output (prompting) operations directed at the user, enforces authentication and authorization, and issues credentials (information containing the user's identity, privileges, and roles within a given security environment).

The ACME subsystem was designed to operate under one of two authentication models: *cooperative* and *independent*. In the cooperative model, all domain-of-interpretation (DOI) agents enforce authentication and issue credentials using a single principal-name and password scheme as seen from the perspective of the $ACM application. In the independent model, a DOI agent performs authentication and issues credentials only when it is operating as the designated DOI agent, otherwise it does not participate in the request. These concepts are explained further in Section 1.8.1.

These two models are implemented using a set of rules that are conventionally followed by agents. It is technically possible to design an agent that does not follow these rules. But such agents will not operate with other agents, which follow the conventional set of rules.

Figure 1–1 shows how the ACM client and ACME server processes interoperate to implement authentication.

**Figure 1–1 Overview of the ACME Subsystem**




This chapter provides a general description of the ACME agent interface and rules. Before proceeding, familiarize yourself with the ACME-related terms defined in Table 1–1.

**Table 1–1 ACME Terminology**

| Term | Definition |
|---|---|
| Agent-specific item codes | The set of extended item codes that are defined by an agent and known only to that agent and any customized $ACM applications. An agent never prompts a generic $ACM application for agent-specific item codes unless the item code represents a simple text-based data element that a generic $ACM application can process blindly. For example, an agent can prompt a generic $ACM application for an agent-specific item code representing a “token id” string which can be responded to by a human user, but the agent is not allowed to prompt a generic $ACM application for specialized, binary data such as might be used in a hardware token. |
| Auxiliary agent | An agent that implements a partial authentication policy or some function such as password filtering, but does not issue credentials. It cannot be the target of an $ACM call. |

(continued on next page)

**Table 1–1 (Cont.) ACME Terminology**

| Term | Definition |
|---|---|
| Common item codes | The set of basic item codes documented by the $ACM system service that exists for every OpenVMS system and is recognized by all agents and $ACM applications. All common item codes can be specified on the initial $ACM call, but only a subset of well-known common item codes may be processed in dialogue mode (see below). Examples of common item codes are:<br><br>ACME$_LOGON_TYPE<br>ACME$_AUTH_MECHANISM<br>ACME$_NEW_PASSWORD_FLAGS<br>ACME$_PRINCIPAL_NAME_IN<br>ACME$_PASSWORD_1 |
| Designated DOI agent | For untargeted $ACM calls, the DOI agent that locates the principal-name in its principal-name database. For targeted $ACM calls, the DOI agent specified in the call. This is the only DOI agent allowed to prompt for passwords. It always issues credentials and is responsible for the ultimate success or failure of the request. |
| Dialogue mode | The mode in which an agent issues a request to acquire information from the user (or to be displayed to the user). The calling application obtains the information from the user (or displays it to the user) and calls $ACM again to proceed until the service indicates that no further interaction is required. $ACM applications that specify the context argument operate in dialogue mode. |
| DOI | Domain-of-Interpretation. A DOI represents a security environment having a principal-namespace, authentication and authorization schemes, and information representing a user's identity (both VMS and DOI-specific) and privileges. A DOI agent implements a particular DOI. A DOI agent can be the target of an $ACM call. |
| Password | A string, known only to the user and the system, used to verify the user's identity. |
| Phase | A phase is a discrete stage of request processing. Each phase is associated with a callout routine within an agent that the ACME server invokes. |
| Principal-Name | A string representing a user (sometimes referred to as "username"). |
| Request | An $ACM request is represented internally as a *work queue entry* (WQE). The WQE is used to maintain the state of the request through multiple stages of processing. It is also used to control certain interactions among the agents. |
| Secondary DOI agent | A DOI agent is one that is not operating as the designated DOI agent. It performs authentication and issues credentials according to the model under which it was designed and configured. |
| Targeted call | A call to $ACM that specifies the ACME$_TARGET_DOI_ID or ACME$_TARGET_DOI_NAME item code. |
| Untargeted call | A call to $ACM that does not specify the ACME$_TARGET_DOI_ID or ACME$_TARGET_DOI_NAME item code. |
| Well-known item codes | The set of common item codes that an $ACM application can expect to process in dialogue mode (or to supply in a single non-dialogue $ACM call). Generic $ACM applications can respond to well-known item codes, even in restricted operating environments where there is no human user with which to interact or where application protocols accept only username and password data. Examples of well-known item codes are:<br><br>ACME$_PASSWORD_SYSTEM<br>ACME$_PRINCIPAL_NAME_IN<br>ACME$_PASSWORD_1 |

## 1.1 Concepts

Understanding of following concepts is required before implementing an ACME agent.

### 1.1.1 Agents

An ACME agent enforces a user authentication policy when a program calls the $ACM system service. Agents perform the following operations:

- Identification
- Authentication
- Authorization
- Credential issuance

There are two types of agents, *DOI* and *auxiliary*.

A DOI agent is responsible for password prompting, authentication, and issuing credentials. For any given request, one DOI agent declares itself to be the designated DOI agent. All other DOI agents operate as secondary DOI agents, participating in request processing according to the model under which they were designed and configured to operate. Auxiliary agents (see below) operate in concert with the designated DOI agent, providing stronger authentication or some other function.

An auxiliary agent logically works in conjunction with the designated DOI agent. Both designated DOI agent and auxiliary agent must pass authentication and authorization checks for the request to succeed. The designated DOI agent will perform all of its usual authentication operations.

### 1.1.2 $ACM Call Types

An application can target an $ACM call to a particular DOI agent, or it can call $ACM without directing the request to any particular DOI.

In the first instance, known as a *targeted call*, an $ACM call may target a particular DOI agent. When it does, the targeted DOI agent will be the designated DOI agent.

In the second instance, known as an *untargeted call*, an $ACM call does not target a DOI agent. Instead, the first DOI agent to locate the principal-name in its namespace will be the designated DOI agent. The remaining DOI agents function as secondary DOI agents.

When multiple DOI agents are involved in processing a request, each DOI agent authenticates and issue credentials according to the model under which it was designed and configured to operate.

### 1.1.3 Call Modes

$ACM calls operate in either *dialogue mode* or *non-dialogue mode*.

When called with a non-zero $ACM context argument, the $ACM service operates in dialogue mode which allows an agent to drive input and output operations for the user. This is the recommended method of calling $ACM when there is a human user (as opposed to a client program) interacting with the $ACM application.

### 1.1.4 Ordering of Agents

In most cases, the order in which agents are configured is not important. Requests are processed in stages according to the phases defined for each function. The sequence of phases impose a logical ordering of processing steps and user input/output operations.

There are two cases where ordering is important:

- When the designated DOI agent for untargeted calls is determined according to principal-name.

  For an untargeted $ACM call, the first DOI agent to find the principal-name valid in its namespace declares itself the designated DOI agent. If namespaces among DOI agents overlap, the DOI agent that is configured first (using the SET SERVER ACME/ENABLE command) will always become the designated DOI agent.

- When the sequence of prompts is generated by multiple agents in a single phase.

  In this case, any user input/output operations generated by more than one agent in any given phase will be prompted or displayed in the order the agents are configured.

## 1.2 $ACM Request Functions and Phases

$ACM requests specify the ACME function to be performed. Requests can specify one of the following functions:

| | |
|---|---|
| AUTHENTICATE_PRINCIPAL | Authenticate a user and obtain credentials. |
| CHANGE_PASSWORD | Change a user's password. |
| QUERY | Perform a DOI-defined "query" function. |
| EVENT | Perform a DOI-defined "event" function. |
| RELEASE_CREDENTIALS | Delete DOI-specific credentials. |

The ACME server processes requests in a series of phases. Each phase is associated with a callout routine within an agent. Most functions are processed as a single phase while two are processed in several phases.

EVENT, QUERY, and RELEASE_CREDENTIALS functions are processed as a single phase. They are targeted to a particular DOI agent, do not involve other DOI agents, nor do they interact with the $ACM caller. These are simple functions from an agent's point of view.

AUTHENTICATE_PRINCIPAL and CHANGE_PASSWORD functions are processed as a sequence of phases that may involve more than one DOI agent and may also involve dialogue with the $ACM caller. These are relatively complex functions.

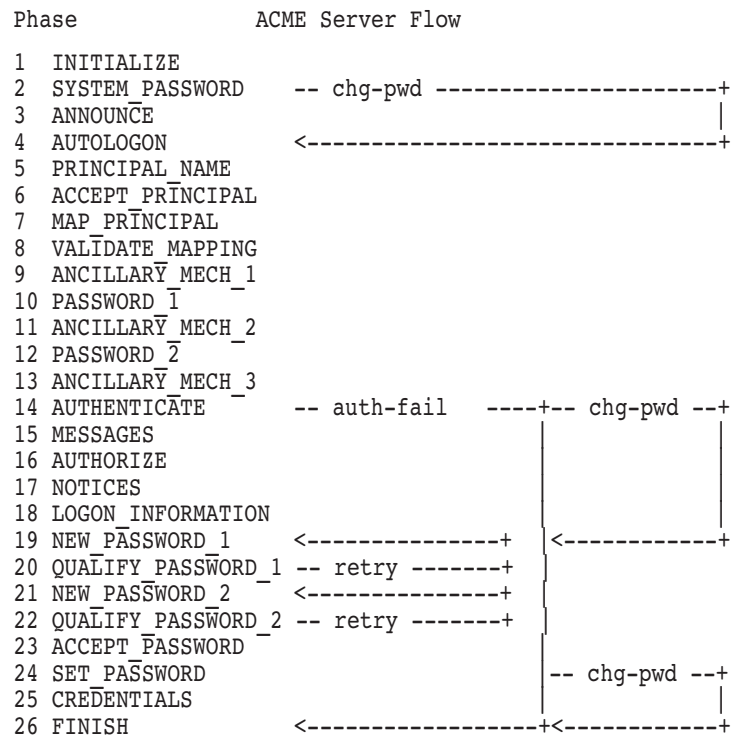
## 1.3 Authentication and Change-Password Phase Dispatching

AUTHENTICATE_PRINCIPAL and CHANGE_PASSWORD phases are defined to be compatible with the VMS user authentication policy while allowing flexibility to work with other policies. Phases are defined for standard operations such as prompting for principal-name, validating a password, etc. An agent is expected to utilize each phase as it is defined and ignore a phase that has no meaning to the agent.

The ACME server invokes each agent's callout routines in order, beginning with INITIALIZE (see below) and ending with FINISH. When multiple agents exist, the ACME server invokes a given callout routine for every agent (in the order that the agents were configured) before moving to the next phase. Agents follow a set of rules which, by convention, allow agents to cooperate in a meaningful and predictable manner for the $ACM application and user. These rules are described later in this document.

The rest of this section describes the general nature of phase operations and how an agent interacts with the ACME server.

In the following diagram, AUTHENTICATE_PRINCIPAL and CHANGE_PASSWORD functions are processed in a sequence of 26 phases. (The ACME server skips certain phases for the CHANGE_PASSWORD function.)




## 1.3.1 ACME Server Flow Control

ACME server flow control is managed by callout routines that return values that affect dispatching flow and determine the final status of the request, as shown in the following table.

**Table 1–2 Values Returned by Callout Routines**

| Value | Action |
|---|---|
| 0 | Converted to ACME$_FAILURE by ACME server, set request status, advance to FINISH phase |
| SS$_NORMAL | Converted to ACME$_CONTINUE by ACME server |
| ACME$_CONTINUE | Advance to next agent's callout routine |

(continued on next page)

**Table 1–2 (Cont.) Values Returned by Callout Routines**

| Value | Action |
|---|---|
| ACME$_WAITAST | Stall until AST is delivered and return to this agent's callout routine |
| ACME$_WAITRESOURCE | Stall until resource is available and return to this agent's callout routine |
| ACME$_PERFORMDIALOGUE | Perform dialogue and return results to this agent's callout routine |
| ACME$_RETRYPWD | Backup to the start of the previous NEW_PASSWORD_* phase |
| ACME$_AUTHFAILURE prior to AUTHENTICATE phase | Set request status, convert to ACME$_CONTINUE, continue processing until AUTHENTICATE phase, then return ACME$_AUTHFAILURE |
| Any other status | Set request status and advance to FINISH phase |

## 1.3.2 Special Dispatching Cases

The following table shows how the ACME server responds to values returned by agents.

| If | An agent returns | The ACME server |
|---|---|---|
| Prior to the AUTHENTICATE phase | ACME$_AUTHFAILURE | 1. Sets the status in the WQE<br>2. Flags the request as failed<br>3. Continues processing through the AUTHENTICATE phase<br><br>Once the AUTHENTICATE phase is complete, the ACME server checks the request failure flag and if set, stores the ACME$_AUTHFAILURE value in the request's status field and moves directly to the FINISH phase.<br><br>This permits agents to keep their prompting sequence intact in the event of an early failure (e.g. "no such user") without giving away this information to the user. |
| In the NEW_ PASSWORD_* phase | ACME$_RETRYPWD | Returns control to the NEW_PASSWORD_* phase for retry |
| In the QUALIFY_ PASSWORD_* phase | ACME$_RETRYPWD | Returns control to the NEW_PASSWORD_* phase for retry |
| At anytime | ACME$_WAITAST<br>ACME$_WAITRESOURCE<br>ACME$_ PERFORMDIALOGUE | Stalls the request until the required action is performed and then returns control to the same callout routine to continue processing |

Note the following:

- If no agent services the request, the ACME server will fail the request with ACME$_AUTHFAILURE.
- The ACME server will call every agent's FINISH phase regardless of the status returned by the previous agent.
- The first agent to return a failure status (other than one of the dispatching control statuses) to the ACME server will cause the request (WQE) status field to be set to that value.

### 1.3.3 Dialogue Mode

Agents interact with the $ACM caller using dialogue operations. A dialogue operation can be a request for data from the user or information to be displayed to the user. The $ACM caller processes the dialog request and prompts the user for the data or displays the information to the user, as specified. The agent must supply the item code that is associated with the data to be returned by the $ACM caller and, optionally, supplies the prompt and default strings for user input operations.

To initiate a dialogue request, an agent calls the ACME$CB_QUEUE_DIALOGUE callback routine to prepare the dialog request and then returns ACME$_PERFORMDIALOGUE when returning from the agent's callout routine.

In this ACME version, only text-based dialogue interactions are supported. The INPUT dialogue-support flag supplied by the $ACM caller indicates the client is capable of text-based user input/output operations. In the future, other forms of dialogue may be supported.

### 1.3.4 Waiting for ASTs

Agents can stall request processing while waiting for an asynchronous operation to complete with an AST.

An agent waits for AST delivery from a asynchronous service by calling the ACME$CB_ACQUIRE_WQE_AST callback to obtain an AST routine address and AST parameter address when calling the asynchronous service. The agent returns ACME$_WAITAST upon returning from the callout routine. When the asynchronous service completes, the ACME server will receive delivery of the AST and will dispatch it to the agent at non-AST level when it invokes the agent's callout routine again.

### 1.3.5 Waiting for Resources

Agents can stall request processing while waiting for an agent-specific resource to become available. The ACME server provides support for an agent to acquire (ACME$CB_ACQUIRE_RESOURCE) and release (ACME$CB_RELEASE_RESOURCE) agent-specific resources. There is no special callback routine to create a resource. To create resources, an agent calls ACME$CB_RELEASE_RESOURCE one or more times in the agent's startup phase, depending on the number of such resources an agent will manage.

During request processing, the agent calls ACME$CB_ACQUIRE_RESOURCE to attempt to allocate an instance of the resource. If the call fails, the agent returns ACME$_WAITRESOURCE upon returning from the callout routine. As the resource becomes available (by another request calling ACME$_RELEASE_RESOURCE), the ACME server will again invoke the agent's callout routine.

## 1.4 ACME Server Callback Routines

Callbacks are routines in the ACME server image for agents to call for various operations. The routine addresses for callbacks are obtained from the ACME server's kernel callback (KCB) vector that is provided as an argument to each callout routine.

Callback routines are categorized by how they provide support for agents, as follows:

**Coordinate activities with other agents**

ACME$CB_SET_DESIGNATED_DOI
ACME$CB_SET_PHASE_EVENT
ACME$CB_SET_WQE_FLAG
ACME$CB_SET_WQE_PARAMETER

**Manage AST contexts (global and per-request)**

ACME$CB_ACQUIRE_ACME_AST
ACME$CB_ACQUIRE_ACME_RMSAST
ACME$CB_ACQUIRE_WQE_AST
ACME$CB_ACQUIRE_WQE_RMSAST
ACME$CB_RELEASE_ACME_AST
ACME$CB_RELEASE_ACME_RMSAST
ACME$CB_RELEASE_WQE_AST
ACME$CB_RELEASE_WQE_RMSAST

**Manage Resources**

ACME$CB_ACQUIRE_RESOURCE
ACME$CB_RELEASE_RESOURCE

**Allocate memory (global and per-request)**

ACME$CB_ALLOCATE_ACME_VM
ACME$CB_ALLOCATE_WQE_VM
ACME$CB_DEALLOCATE_ACME_VM
ACME$CB_DEALLOCATE_WQE_VM

**Communicate with the $ACM client process (dialogue calls, set request status, and load output items requested by the client)**

ACME$CB_CANCEL_DIALOGUE
ACME$CB_FORMAT_DATE_TIME
ACME$CB_ISSUE_CREDENTIALS
ACME$CB_SET_2ND_STATUS
ACME$CB_ACME_STATUS
ACME$CB_SET_LOGON_FLAG
ACME$CB_SET_LOGON_STATS_DOI
ACME$CB_SET_LOGON_STATS_VMS
ACME$CB_QUEUE_DIALOGUE_ITEM
ACME$CB_SET_OUTPUT_ITEM

**Write to the ACME server log file or operator**

ACME$CB_SEND_LOGFILE
ACME$CB_SEND_OPERATOR

**Report agent status to the ACME server**

ACME$CB_REPORT_ACTIVITY
ACME$CB_REPORT_ATTRIBUTES

## 1.5 Agent "Request" Callout Routines

Request phases for a given $ACM function invoke an agent's callout routines. EVENT and QUERY functions are single-phased functions operating under a single DOI agent. AUTHENTICATE_PRINCIPAL and CHANGE_PASSWORD functions are multi-phased functions that operate under multiple DOI agents and auxiliary agents.

The ACME server calls the following callout routines for the EVENT and QUERY functions.

| Phase | Routine |
|---|---|
| EVENT | ACME$CO_EVENT |

Agent-specific. No defined operations.

| Phase | Routine |
|---|---|
| QUERY | ACME$CO_QUERY |

Agent-specific. No defined operations.

The ACME server calls the following callout routines for the AUTHENTICATE_ PRINCIPAL and CHANGE_PASSWORD functions:

| Phase | Routine |
|---|---|
| INITIALIZE | ACME$CO_INITIALIZE |

Each new request is processed starting with this phase. Agents should allocate a request context structure in this phase that the ACME server will then return in subsequent callout routines via the request-context argument.

Agents may also check the item lists supplied as arguments to the callout routine to determine whether any well-known items were specified on the initial call. This may be more convenient than searching through the WQE's item lists which contain the entire history of items supplied by the caller when operating in dialogue mode.

| Phase | Routine |
|---|---|
| SYSTEM_PASSWORD | ACME$CO_SYSTEM_PASSWORD |

The VMS agent processes the VMS system password in this phase, if one is defined for the system. This phase is associated with the ACME$_PASSWORD_ SYSTEM item code.

| Phase | Routine |
|---|---|
| ANNOUNCE | ACME$CO_ANNOUNCE |

This phase is used to display information to the user prior to the username prompt.

The VMS agent displays the contents of SYS$ANNOUNCE.

| Phase | Routine |
|---|---|
| AUTOLOGON | ACME$CO_AUTOLOGON |

Agents automatically determine the principal-name based on some criteria and store it in the WQE, if possible.

The VMS agent processes the Automatic Login Facility (ALF) function in this phase.

| Phase | Routine |
|---|---|
| PRINCIPAL_NAME | ACME$CO_PRINCIPAL_NAME |

If no principal-name has been determined by any previous agent (i.e. no value is stored in the WQE), the current agent can prompt the user for it and store it in the WQE. Prompting is optional, because the VMS agent will prompt if no principal-name has been stored in the WQE.

Agents should use the following sequence to determine the principal-name:

1. Look in the WQE (already established)
2. Check if supplied in initial phase on the first dialogue call
3. Search the common item list supplied as an argument to the callout routine (which represents the items supplied on this dialogue request)
4. Prompt the user

This phase is associated with the ACME$_PRINCIPAL_NAME_IN item code.

The VMS agent prompts for "Username: " in this phase.

| Phase | Routine |
|---|---|
| ACCEPT_PRINCIPAL | ACME$CO_ACCEPT_PRINCIPAL |

The first agent that successfully locates the principal-name in its namespace declares itself the designated DOI agent. In later phases, the designated DOI agent maps the principal-name to a VMS username, prompts for passwords, performs authentication and authorization, and issues credentials.

If no agent has declared itself the designated DOI, the VMS agent looks up the principal-name in the SYSUAF.DAT file. If it finds a record with that username and it satisfies certain criteria (see EXTAUTH and IGNORE_EXTAUTH controls), the VMS agent declares itself the designated DOI agent.

| Phase | Routine |
|---|---|
| MAP_PRINCIPAL | ACME$CO_MAP_PRINCIPAL |

The designated DOI agent provides a VMS username-mapping for the principal-name in this phase.

The VMS agent simply uses the principal-name, if it is the designated DOI agent.

| Phase | Routine |
|---|---|
| VALIDATE_MAPPING | VALIDATE_MAPPING |

Agents validate the VMS username-mapping in this phase, if they will authenticate and issue credentials for this request. If the agent's VMS mapping is different than the one stored in the WQE, the agent must fail the request.

If the VMS agent is not the designated DOI agent, it uses this phase to ensure a record exists in the SYSUAF.DAT file for the mapped username and that it satisfies certain criteria (see the EXTAUTH and IGNORE_EXTAUTH controls).

| Phase | Routine |
|---|---|
| ANCILLARY_MECH_1 | ACME$CO_ANCILLARY_MECH_1 |

Agent-specific.

The VMS agent does not use this phase.

| Phase | Routine |
|---|---|
| PASSWORD_1 | ACME$CO_PASSWORD_1 |

The designated DOI agent uses this phase to prompt for a primary password, if required, and stores it in the WQE. No password validation is performed in this phase.

Agents should use the following sequence to determine the password:

1. Look in the WQE (already established)
2. Check if supplied in initial phase on the first dialogue call
3. Search the common item list supplied as an argument to the callout routine (which represents the items supplied on this dialogue request)
4. Prompt the user

This phase is associated with the ACME$_PASSWORD_1 item code.

The VMS agent prompts for "Password: " in this phase.

| Phase | Routine |
|---|---|
| ANCILLARY_MECH_2 | ACME$CO_ANCILLARY_MECH_2 |

Agent-specific. The VMS agent does not use this phase.

| Phase | Routine |
|---|---|
| PASSWORD_2 | ACME$CO_PASSWORD_2 |

The designated DOI agent uses this phase to prompt for a secondary password, if required, and stores it in the WQE. No password validation is performed in this phase.

Agents should use the following sequence to determine the password:

1. Look in the WQE (already established)
2. Check if supplied in initial phase on the first dialogue call
3. Search the common item list supplied as an argument to the callout routine (which represents the items supplied on this dialogue request)
4. Prompt the user

This phase is associated with the ACME$_PASSWORD_2 item code.

The VMS agent prompts for “Secondary Password: ” in this phase.

| Phase | Routine |
|---|---|
| ANCILLARY_MECH_3 | ACME$CO_ANCILLARY_MECH_3 |

Agent-specific.

The VMS agent does not use this phase.

| Phase | Routine |
|---|---|
| AUTHENTICATE | ACME$CO_AUTHENTICATE |

The designated DOI agent and any auxiliary agents validate the passwords in this phase and perform other validation as defined for those agents. Secondary DOI agents also perform validation, if designed and configured to operate under the cooperative model.

The VMS agent performs standard SYSUAF.DAT password validation in this phase, including intrusion detection.

| Phase | Routine |
|---|---|
| MESSAGES | ACME$CO_MESSAGES |

Agents use this phase to display information to the user after authentication and before authorization.

The VMS agent does not use this phase.

| Phase | Routine |
|---|---|
| AUTHORIZE | ACME$CO_AUTHORIZE |

The designated DOI agent and any auxiliary agents perform authorization checks. Secondary DOI agents may also perform authorization, if so designed and configured to operate under the cooperative model.

The VMS agent uses this phase to perform modal restrictions checks and account disabled check.

| Phase | Routine |
|---|---|
| NOTICES | ACME$CO_NOTICES |

This phase is used to display lengthy notices following successful authentication and authorization.

The VMS agent uses this phase to display the contents of SYS$WELCOME.

| Phase | Routine |
|---|---|
| LOGON_INFORMATION | ACME$CO_LOGON_INFORMATION |

This phase is used to display short notices following successful authentication and authorization.

The VMS agent uses this phase to display last-login time, number of failed logins, etc.

| Phase | Routine |
|---|---|
| NEW_PASSWORD_1 | ACME$CO_NEW_PASSWORD_1 |

In this phase, the designated DOI agent prompts for a new primary password, if the existing password has expired.

Agents should use the following sequence to determine the new password:

1. Look in the WQE (already established)
2. Check if supplied in initial phase on the first dialogue call
3. Search the common item list supplied as an argument to the callout routine (which represents the items supplied on this dialogue request)
4. Prompt the user

This phase is associated with the ACME$_NEW_PASSWORD_1 item code.

If operating as the designated DOI agent, the VMS agent uses this phase to prompt as follows:

```
New Password:
Verification:
```

| Phase | Routine |
|---|---|
| QUALIFY_PASSWORD_1 | ACME$CO_QUALIFY_PASSWORD_1 |

This phase is used by secondary DOI agents operating under the cooperative model when auxiliary agents check the validity of the proposed password.

The VMS agent uses this phase to check the password history and password dictionary databases.

| Phase | Routine |
|---|---|
| NEW_PASSWORD_2 | ACME$CO_NEW_PASSWORD_2 |

In this phase, the designated DOI agent prompts for a new secondary password, if the existing secondary password has expired.

Agents should use the following sequence to determine the new password:

1. Look in the WQE (already established)

2. Check if supplied in initial phase on the first dialogue call
3. Search the common item list supplied as an argument to the callout routine (which represents the items supplied on this dialogue request)
4. Prompt the user

This phase is associated with the ACME$_NEW_PASSWORD_2 item code. If operating as the designated DOI agent, the VMS agent uses this phase to prompt as follows:

```
New Secondary Password:
Verification:
```

| Phase | Routine |
|---|---|
| QUALIFY_PASSWORD_2 | ACME$CO_QUALIFY_PASSWORD_2 |

This phase is used by secondary DOI agents operating under the cooperative model when auxiliary agents check the validity of the proposed secondary password.

The VMS agent uses this phase to check the password history and password dictionary databases.

| Phase | Routine |
|---|---|
| ACCEPT_PASSWORDS | ACME$CO_ACCEPT_PASSWORDS |

This phase is used to prepare for setting the passwords in the next phase.

| Phase | Routine |
|---|---|
| SET_PASSWORDS | ACME$CO_SET_PASSWORDS |

This phase is used to write the passwords to the agent’s authentication database.

The VMS agent uses this phase to store the password in the SYSUAF.DAT file.

| Phase | Routine |
|---|---|
| CREDENTIALS | ACME$CO_CREDENTIALS |

This phase is used by the designated DOI agent and secondary DOI agents operating under the cooperative model to issue credentials that will be converted to a persona extension by the ACME server and returned to the $ACM application.

The VMS agent uses this phase to issue the VMS security profile for the user that will be returned in the base persona structure.

| Phase | Routine |
|---|---|
| FINISH | ACME$CO_FINISH |

This is the final phase of request processing. Agents should audit the success or failure of the request and perform any clean-up that is necessary. If a request context structure was allocated, it should be deallocated in this phase.

The VMS agent uses this phase to generate login and logfail audits.

## 1.6 Agent “Control” Callout Routines

Control operations directed to the ACME server by the system manager involve activities that must be performed by agents. There are callout routines associated with these activities, as follows.

| Command | Routine |
|---|---|
| SET SERVER ACME/CONFIGURE | ACME$CO_AGENT_INITIALIZE |

Agents are dynamically-activated into the ACME server process (brought into the ACME server’s virtual memory space), but not yet active. Request dispatching is disabled.

| Command | Routine |
|---|---|
| SET SERVER ACME/ENABLE | ACME$CO_AGENT_STARTUP |

Requests dispatching is enabled. The agent’s request callout routines will be invoked following execution of this routine.

This is the preferred stage to allocate global memory and create agent resources.

| Command | Routine |
|---|---|
| SET SERVER ACME/SUSPEND | ACME$CO_AGENT_STANDBY |

Request dispatching is temporarily disabled. Agent should become idle while system management tasks, such as system backups, are performed. That is, the agent should close any open files or shutdown any other activity that might prevent system managements tasks from completing.

Request dispatching will resume when the SET SERVER ACME /RESUME command is issued. There is no callout routine for this command, so agents should be prepared to open files as needed during request processing.

| Commands | Routine |
|---|---|
| SET SERVER ACME/EXIT<br>SET SERVER ACME/DISABLE | ACME$CO_AGENT_SHUTDOWN |

Request dispatching is disabled. Agent must shut down for possible reconfiguration. There is no guarantee that the agent will be reenabled in the future.

This is the preferred stage to deallocate global memory and release agent resources.

## 1.7 WQE Fields

Table 1–3 shows the standard fields in the WQE.

**Table 1–3 WQE Standard Fields**

| Field | Definition |
|---|---|
| FLAGS | Set by an agent to control certain processing features. Set using ACME$CB_SET_WQE_FLAG. |
| TARGET_ACME_ID | Set by $ACM service when the caller specifies either the ACME$_TARGET_DOI_ID or ACME$_TARGET_DOI_NAME item code. |
| DESIGNATED_ACME_ID | Set by the first DOI agent that finds the principal- name valid in its namespace. Set using ACME$CB_SET_DESIGNATED_DOI. |
| STATUS | Set by ACME server. If the request is rejected by an agent, the first failure status reported by an agent returning from a callout routine is used. |
| SECONDARY_STATUS | Set by an agent to provide more details for the status of a request. Set using ACME$CB_SET_2ND_STATUS. |
| ACME_STATUS | Set by an agent to provide agent-specific error codes. Set using ACME$CB_SET_ACME_STATUS. |

Table 1–4 shows the WQE extensions for the AUTHENTICATE_PRINCIPAL and CHANGE_PASSWORD functions.

**Table 1–4 WQE Extensions**

| Field | Definition |
|---|---|
| NEW_PASSWORD_FLAGS | Set by $ACM service when the caller specifies the ACME$_NEW_PASSWORD_FLAGS item code. |
| LOGON_FLAGS | Set by an agent. Used to load the logon-flags portion of the ACME$_LOGON_INFORMATION item code. Set using ACME$CB_SET_LOGON_FLAG. |
| LOGON_STATS_DOI | Set by an agent. Used to load the DOI-specific portion of the ACME$_LOGON_INFORMATION item code. Set using ACME$CB_SET_LOGON_STATS_DOI. |
| LOGON_STATS_VMS | Set by the VMS agent. Used to load the VMS DOI portion of the ACME$_LOGON_INFORMATION item code. Set using ACME$CB_SET_LOGON_STATS_VMS. |
| SYSTEM_PASSWORD | Set by an agent when the caller specifies the ACME$_PASSWORD_SYSTEM item code. Set using ACME$CB_SET_WQE_PARAMETER. |
| PRINCIPAL_NAME | Set by an agent when the caller specifies the ACME$_PRINCIPAL_NAME_IN item code. Set using ACME$CB_SET_WQE_PARAMETER.<br>This value must be loaded, unmodified, from the ACME$_PRINCIPAL_NAME_IN item code. The value represents the original principal-name string as specified by the user and used to determine the appropriate DOI agent. |
| PRINCIPAL_NAME_OUT | Set by the designated DOI agent to convert the principal-name to a standard format. Used to load the ACME$_PRINCIPAL_NAME_OUT item code. Set using ACME$CB_SET_WQE_PARAMETER.<br><br>This string is used to load the principal-name field of the persona extension and will be used to identify the default user in future re-authenticate and change password operations. |
| VMS_USERNAME | Set by the designated DOI agent. Used to load the ACME$_MAPPED_VMS_USERNAME item code. Set using ACME$CB_SET_WQE_PARAMETER. |

(continued on next page)

**Table 1–4 (Cont.) WQE Extensions**

| Field | Definition |
|---|---|
| PASSWORD_1 | Set by the designated DOI agent (or auxiliary agent) when the caller specifies the ACME$_PASSWORD_1 item code. Set using ACME$CB_SET_WQE_PARAMETER.<br>This password string may be modified before being stored in the WQE. This allows an auxiliary agent to use the password supplied by the user to unlock the password needed by the designated DOI agent. |
| PASSWORD_2 | Set by the designated DOI agent (or auxiliary agent) when the caller specifies the ACME$_PASSWORD_2 item code. Set using ACME$CB_SET_WQE_PARAMETER.<br>This password string may be modified before being stored in the WQE. This allows an auxiliary agent to use the password supplied by the user to unlock the password needed by the designated DOI agent. |
| NEW_PASSWORD_1 | Set by the designated DOI agent (or auxiliary agent) when the caller specifies the ACME$_NEW_PASSWORD_1 item code. Set using ACME$CB_SET_WQE_PARAMETER. |
| NEW_PASSWORD_2 | Set by the designated DOI agent (or auxiliary agent) when the caller specifies the ACME$_NEW_PASSWORD_1 item code. Set using ACME$CB_SET_WQE_PARAMETER. |

## 1.8 Agent Rules

Agents that are designed to work together must comply with the rules described in this section. These rules are designed with the following two goals in mind:

- To present the user with a logical sequence of prompts.
- To allow the system manager to define configurations that use a common (synchronized across DOI's) principal-name and password and issue credentials as a group.

### 1.8.1 DOI Agents

DOI agents operate as either the designated DOI agent or a secondary DOI agent for a given request. The functions that a DOI performs depends on the model, cooperative or independent, under which is was designed and configured to operate.

**Cooperative Model: User obtains a credential from each DOI**

```
Username: joe@acme.com
Password:
```

All DOI agents authenticate and issue credentials, using a single principal-name and password. The designated DOI agent drives the prompting sequence.

While the user enters only one principal-name and password, the agents use this information as-is (synchronized principal-names and passwords).

**Independent Model: User obtains a credential from one DOI**

```
Username: joe@acme.com
Password:
```

Only the designated DOI agent authenticates the user and issues credentials. Secondary DOI agents do not participate in the request.

### 1.8.2 Auxiliary Agents

An auxiliary agent works in conjunction with the designated DOI agent to enforce stronger authentication or other administrative functions such as displaying special information. For example:

```
Username: abc
Password:
Token-Challenge: 3VN0-QVV5-TVQ5-524T
Token Response: 53BV-2GC5-36V5-V21Y
```

The designated DOI agent authenticates using the password while the auxiliary agent authenticates using the token challenge-response. The principal-name is associated with the DOI represented by the designated DOI agent.

Auxiliary agents always operate alongside a designated DOI agent and are restricted to the following operations:

- Prompt/load principal-name field
- Validate VMS username mapping
- Prompt/load password field
- Utilize enhanced authentication mechanisms
- Perform additional authentication checks
- Perform additional authorization checks
- Perform additional password filtering
- Obtain or display miscellaneous information

It is important to note that since auxiliary agents do not provide credentials, the principal-name must be the one that the designated DOI agent recognizes and loads into its persona extension. This is the principal-name that will be used as the default for any future authentication or change-password operations.

### 1.8.3 Choosing Between a DOI Agent or an Auxiliary Agent

When designing an agent, the first decision to make is whether the agent will function as an auxiliary agent or DOI agent. An auxiliary agent is simpler to design and implement, but is functionally less powerful than a DOI agent.

Use an auxiliary agent when you wish to add an authentication mechanism on top of a DOI's password mechanism or to add extra filtering of a user's new password during a password change. In either case, the auxiliary uses the designated DOI agent's principal-name to identify the user. An auxiliary agent does not replace the designated DOI agent's password mechanism.

Use a DOI agent when you wish to issue credentials that represent the user within a particular DOI, including principal-name and other information. You must use a DOI agent when you wish to replace the VMS password mechanism, because the password mechanism enforced by the designated DOI agent replaces the VMS agent's password mechanism during authentication.

### 1.8.4 Controls for Secondary DOI Agents

All DOI agents are responsible for principal-name validation, VMS username mapping, password validation, authorization, and issuing credentials.

Secondary DOI agents can operate under the cooperative or independent model, depending on how the system manager has configured them to operate and how the agents were designed.

Developers should provide controls to allow the system manager to configure their DOI agent for either model when it is operating as a secondary DOI agent. Table 1–5 summarizes the control semantics for secondary DOI agents.

**Table 1–5 Control Semantics**

| In this role | The agent |
|---|---|
| Cooperative | Will service the request<br>May prompt for principal-name<br>Must not prompt for password<br>Authenticates, authorizes, and issues credentials |
| Independent | Will not service the request<br>(no prompting, no authentication, no authorization, no credentials issued) |

These controls are implemented in a DOI-specific manner using logical names or some other mechanism designed by the agent developer.

## 1.9 Phase Rules

Table 1–6 and Table 1–7 show the ACME phase rules for the AUTHENTICATE_PRINCIPAL and CHANGE_PASSWORD functions in the cooperative and independent models.

**Table 1–6 Cooperative Model**

| Phase | | Silver[1] | Gold[1] | VMS[1] | Zinc[2] |
|---|---|---|---|---|---|
| 1 | Initialize | {+} | {+} | {+} | {+} |
| 2 | System-Password (prompt) | | | {+} | |
| 3 | Announce | <> | <> | <> | <> |
| 4 | Auto-Logon | [] | [] | [] | [] |
| 5 | Principal-Name (prompt) | [+] | [] | [] | [] |
| 6 | Accept-Principal (establish DOI) | [] | [+] | [] | |
| 7 | Map-Principal | | {+} | | |
| 8 | Validate-Mapping | {+} | {+} | {+} | <> |
| 9 | Ancillary-Mech-1 | | <> | | <> |
| 10 | Password-1 (prompt) | | {[+]} | | [] |
| 11 | Ancillary-Mech-2 | | <> | | <> |
| 12 | Password-2 (prompt) | | {[]} | | [] |
| 13 | Ancillary-Mech-3 | | <> | | <+> |
| 14 | Authenticate | {+} | {+} | | <+> |
| 15 | Messages | <> | <> | <> | <> |
| 16 | Authorize | {+} | {+} | {+} | <+> |
| 17 | Notices | <> | <> | <> | <> |
| 18 | Logon-Information | | {+} | {+} | |
| 19 | New-Password-1[3] (prompt) | | {[+]} | | [] |
| 20 | Qualify-Password-1[3] | {+} | {+} | | <> |
| 21 | New-Password-2[3] (prompt) | | {[]} | | [] |
| 22 | Qualify-Password-2[3] (prompt) | {} | {} | | <> |
| 23 | Accept-Passwords[3] | {+} | {+} | | <> |
| 24 | Set-Password[3] | {+} | {+} | | <> |
| 25 | Credentials | {+} | {+} | {+} | |
| 26 | Finish | {+} | {+} | {+} | {+} |

[1]DOI Agent

[2]Auxiliary Agent

[3]Used only when a password expires (or is requested) and needs to be set to a new value.

**Key to Phase Rules**

[]—Operation is completed by the first agent to perform it (optional, unless indicated otherwise)
<>—Optional operation
{}—Mandatory operation, if condition applies
+—Operation performed by the agent in this example

**Table 1–7 Independent Model**

| Phase | | Silver[1] | Gold[1] | VMS[1] | Zinc[2] |
|---|---|---|---|---|---|
| 1 | Initialize | {+} | {+} | {+} | {+} |
| 2 | System-Password (prompt) | | | {+} | |
| 3 | Announce | <> | <> | <> | <> |
| 4 | Auto-Logon | [] | [] | [] | [] |
| 5 | Principal-Name (prompt) | [+] | [] | [] | [] |
| 6 | Accept-Principal (establish DOI) | [] | [+] | [] | |
| 7 | Map-Principal | | {+} | | |
| 8 | Validate-Mapping | | {+} | {+} | <> |
| 9 | Ancillary-Mech-1 | | <> | | <> |
| 10 | Password-1 (prompt) | | {[+]} | | [] |
| 11 | Ancillary-Mech-2 | | <> | | <> |
| 12 | Password-2 (prompt) | | {[]} | | [] |
| 13 | Ancillary-Mech-3 | | <> | | <+> |
| 14 | Authenticate | | {+} | | <+> |
| 15 | Messages | | <> | <> | <> |
| 16 | Authorize | | {+} | {+} | <+> |
| 17 | Notices | | <> | <> | <> |
| 18 | Logon-Information | | {+} | {+} | |
| 19 | New-Password-1[3] (prompt) | | {[+]} | | [] |
| 20 | Qualify-Password-1[3] | | {+} | | <> |
| 21 | New-Password-2[3] (prompt) | | {[]} | | [] |
| 22 | Qualify-Password-2[3] (prompt) | | {} | | <> |
| 23 | Accept-Passwords[3] | | {+} | | <> |
| 24 | Set-Password[3] | | {+} | | <> |
| 25 | Credentials | | {+} | {+} | |
| 26 | Finish | {+} | {+} | {+} | {+} |

[1]DOI Agent

[2]Auxiliary Agent

[3]Used only when a password expires (or is requested) and needs to be set to a new value.

**Key to Phase Rules**

[]—Operation is completed by the first agent to perform it (optional, unless indicated otherwise)
<>—Optional operation
{}—Mandatory operation, if condition applies
+—Operation performed by the agent in this example

| Prompt/Display | Phase | Agent |
|---|---|---|
| (system password) | 2 | VMS |
| Username: | 5 | Silver |
| Password: | 10 | Gold |

| Prompt/Display | Phase | Agent |
|---|---|---|
| Token-ID: | 13 | Zinc |
| New Password: | 19 | Gold |
| Verification: | 19 | Gold |

**Table 1–8 Example, WQE Standard Fields**

| WQE Field | Phase | Agent |
|---|---|---|
| TARGET_ACME_ID | n/a | server |
| FLAGS | any | any |
| DESIGNATED_ACME_ID | 6 | Gold |
| STATUS | any | server |
| SECONDARY_STATUS | any | any |
| ACME_STATUS | any | any |

**Table 1–9 Example, WQE Extensions**

| WQE Field | Phase | Agent |
|---|---|---|
| NEW_PASSWORD_FLAGS | n/a | server |
| LOGON_FLAGS | 18 | Gold |
| LOGON_STAT_VMS | 18 | VMS |
| LOGON_STATS_DOI | 18 | Gold |
| SYSTEM_PASSWORD | 2 | VMS |
| PRINCIPAL_NAME | 5 | Gold |
| PRINCIPAL_NAME_OUT | 6 | Gold |
| VMS_USERNAME | 7 | Gold |
| PASSWORD_1 | 10 | Gold |
| PASSWORD_2 | | |
| NEW_PASSWORD_1 | 19 | Gold |
| NEW_PASSWORD_2 | | |

## 1.10 VMS Agent Operation and Controls

Whether operating as a designated DOI agent or a secondary DOI agent, the VMS agent always operates as a cooperative agent. It never operates as an independent DOI agent. Because of its special role as the native DOI on OpenVMS systems, the VMS agent follows a slightly different set of rules than other DOI agents.

The system manager determines the conditions under which the VMS agent operates (designated or secondary DOI agent). The system manager also selects which SYSUAF accounts are allowed to have the VMS agent operate as a secondary DOI.

#### Secondary DOI Agent

If a record exists in the SYSUAF file for the VMS username mapped by the designated DOI agent, the VMS agent may operate as a secondary DOI agent under any of the following conditions:

- The EXTAUTH flag is set in the user's SYSUAF record
- The IGNORE_EXTAUTH bit is set in the SECURITY_POLICY system parameter bitmask

When it is operating as a secondary DOI agent, the VMS agent enforces authorization and issues credentials, but does not perform authentication (the designated DOI effectively replaces the VMS password policy in this case).

The VMS agent will keep the user's password synchronized between the SYSUAF.DAT file and the designated DOI's password database by generating a VMS hash value for the password in the WQE and storing the hash value in the SYSUAF record (exceptions: see DISPWDSYNCH and GUARD_PASSWORDS). This is done for the benefit of older software which might reference the password hash in the SYSUAF.DAT file.

#### Designated DOI Agent

If a record exists in the SYSUAF file for the principal-name, the VMS agent may operate as the designated DOI agent under any of the following conditions:

- The EXTAUTH flag is clear in the user's SYSUAF record
- The VMSAUTH flag is set in the user's SYSUAF record and the VMS DOI is targeted
- The IGNORE_EXTAUTH bit is set in the SECURITY_POLICY system parameter bitmask

VMS agent controls consist of a set of UAF flags and the bits for the SECURITY_POLICY system parameter, as described in Table 1–10 and Table 1–11.

**Table 1–10 UAF Flags**

| | |
|---|---|
| EXTAUTH | The VMS agent operates as a secondary DOI agent when this flag is set in the user's SYSUAF record (exceptions: see VMSAUTH and IGNORE_EXTAUTH). EXTAUTH is used to flag accounts for which another DOI agent enforces authentication and the VMS agent acts as a secondary DOI agent. VMS authorization (account-disable and modal restrictions) is still enforced. |
| VMSAUTH | The VMS agent operates as the designated DOI agent when this flag is set in the user's SYSUAF record and the call is targeted to the VMS DOI. |
| DISPWDSYNCH | If this flag is set in the user's SYSUAF record, the VMS agent will not synchronize the user's password with the value stored in the WQE. |

**Table 1–11 System Parameter SECURITY_POLICY Bits**

| | |
|---|---|
| IGNORE_EXTAUTH | The VMS agent may operate as the designated DOI agent or a secondary DOI agent, regardless of EXTAUTH flag setting. |

(continued on next page)

**Table 1–11 (Cont.) System Parameter SECURITY_POLICY Bits**

| | |
|---|---|
| GUARD_PASSWORDS | When operating as the designated DOI agent, the VMS agent does not store the password in the WQE. When operating as the secondary DOI agent, the VMS agent does not synchronize the password from the WQE. |
| ALLOW_NOAUTHORIZATION | The VMS agent does not enforce modal restriction checks when /NOAUTHORIZATION is specified in $ACM call. |

## 1.11 NT Agent Operation and Controls

The NT agent operates as either the designated DOI agent or a secondary DOI agent. The system manager can configure the NT agent to operate in either a cooperative or independent role.

To configure an NT agent in a cooperative role, define the PWRK$ACME_GRANT_SECONDARY_CREDS logical name as ALWAYS. If authentication fails, the NT agent does not grant credentials but allows the request to proceed when operating as a secondary DOI agent.

To configure an NT agent in an independent role (the default), define the PWRK$ACME_GRANT_SECONDARY_CREDS logical name as NEVER (or undefined). The NT agent does not perform authentication nor does it issue credentials when operating as the secondary DOI agent.

## 1.12 Operating Environment Restrictions

The ACME server is a multi-threaded process: user threads with kernel threads and upcalls enabled. Requests are serviced on multiple threads, so agents need to be aware of concurrency implications. It is also important that agents do not interfere with the user thread manager.

Here are some important things to remember:

- Use asynchronous system services for best performance
- Use AST callback routines for handling AST delivery
- Do not use event flag 0 in asynchronous system services calls
- Always specify an IOSB parameter in asynchronous system service calls
- Do not assume a given work request will always execute in the same thread following a stall (do not use thread-local storage)
- Do not assume a particular thread implementation

For further details and recommendations, refer to the *Guide to the POSIX Threads Library*.

# 2

# ACME Agent Programming Guidelines

ACME agents work with the ACME server main image to provide comprehensive authentication and credential management (for ACM client processes) through the SYS$ACM[W] system service. To ensure a coherent interface, all ACME agents should follow the programming guidelines presented in this chapter.

Unless otherwise indicated, all pass-by-reference arguments and address pointers within data structures are 32-bit addresses.

Use the CC/VAXC compiler switch to have the ACME agent header files generate the convenient field references to ACME data structures.

The VMS ACME agent is required for a complete operational environment. If you start the ACME_SERVER process manually using SET SERVER ACME commands, you must configure the VMS ACME in order to grant persona-based credentials. Use the following commands to start the ACME_SERVER and configure ACME agents:

```
$ SET SERVER ACME/START/LOG
$ SET SERVER ACME/CONFIG=(NAME=VMS,CRED=VMS)
$ SET SERVER ACME/CONFIG=(NAME=<your-agent>[,CRED=<your-cred>])
$ SET SERVER ACME/ENABLE
```

## 2.1 Operating Environment

Your ACME agent shareable image runs in a multithreaded environment established by the ACME server main image, so your code must not do anything to interfere with scheduling within the ACME server process.

### 2.1.1 Wait Form System Service Calls

Except during ACME agent control callout routines, your ACME agent should not call a wait form system service, including at least:

PTD$READW
SYS$ABORT_TRANSW
SYS$ACMW
SYS$ADD_BRANCHW
SYS$AUDIT_EVENTW
SYS$BRKTHRUW
SYS$CHECK_PRIVILEGEW
SYS$CPU_TRANSITIONW
SYS$CREATE_BRANCHW
SYS$DECLARE_RMW
SYS$DNSW
SYS$END_BRANCHW
SYS$END_TRANSW
SYS$ENQW
SYS$FINISH_RMOPW

SYS$FORGET_RMW
SYS$GETDTIW
SYS$GETDVIW
SYS$GETJPIW
SYS$GETLKIW
SYS$GETQUIW
SYS$GETSYIW
SYS$GETUAI
SYS$ICC_CONNECTW
SYS$ICC_DISCONNECTW
SYS$ICC_RECEIVEW
SYS$ICC_REPLYW
SYS$ICC_TRANSCEIVEW
SYS$ICC_TRANSMITW
SYS$IO_FASTPATHW
SYS$IPCW
SYS$JOIN_RMW
SYS$QIOSERVERW
SYS$QIOW
SYS$RECOVERW
SYS$REGISTRYW
SYS$SETDTIW
SYS$SETEVTASTW
SYS$SETUAI
SYS$SET_DEFAULT_TRANSW
SYS$SNDJBCW
SYS$START_BRANCHW
SYS$START_TRANSW
SYS$SYNCH
SYS$TRANS_EVENTW
SYS$UPDSECW
SYS$UPDSEC_64W
SYS$WAIT

Avoid synchronous I/O. Instead use asynchronous I/O with the ACM dispatcher support for AST (Asynchronous System Trap) contexts.

### 2.1.2 Event Flags and IOSBs

Your ACME agent should always specify EFN$C_ENF when calling a system service that uses an event flag. Avoid the default value of zero since this will result in false “wakes” for other threads of execution. Always supply the IOSB argument to system services that accept one.

### 2.1.3 AST Contexts

Your ACME agent should obtain an ACME **AST context** for any system service call it makes that requires an AST routine address and AST parameter. This allows the ACM dispatcher to intercept the AST and deliver it to your agent at non-AST level. Obtain an AST context by calling one of the following ACME callback routines and specifying the AST routine address and AST parameters in your ACME agent:

- ACME$CB_ACQUIRE_ACME_AST
- ACME$CB_ACQUIRE_ACME_RMSAST
- ACME$CB_ACQUIRE_WQE_AST

- ACME$CB_ACQUIRE_WQE_RMSAST

Pass the AST_HANDLER value and the AST_CONTEXT value you receive from the callback to the system service as the ASTADR and ASTPRM arguments, respectively.

The AST context provided by ACM is a 64-bit quantity that ignores the high order 32 bits. The programmer should pass either the 64-bit value or the low order 32 bits, depending on what system service is being called.

When the system service completes, the ACM dispatcher calls your routine at non-AST level using the AST routine address and AST parameter you specified, to avoid interfering with cooperative multitasking in the ACME server process.

## 2.2 Process Context

Do not directly use the POSIX Threads Library (formerly DECthreads) or Ada Tasking. You should invoke tasking services only by interaction with the ACME server main image, because the mechanism for tasking might change between OpenVMS releases.

### 2.2.1 Privilege Manipulation

You must never set or clear privileges since other ACME agents will need certain privileges. You should specify any privileges needed by your ACME agent in the required call to ACME callback routine ACME$CB_REPORT_ATTRIBUTES. The ACME server main image ensures that the specified privileges are enabled.

Your ACME agent must never depend upon some privilege being absent, since at some time it might run alongside some other ACME agent that requires that privilege.

### 2.2.2 Thread Safety

All ACME agent code must be thread-safe except for the code executed via the following ACME agent control callout routines:

- ACME$CO_AGENT_INITIALIZE
- ACME$CO_AGENT_STARTUP
- ACME$CO_AGENT_STANDBY
- ACME$CO_AGENT_SHUTDOWN

### 2.2.3 Memory Allocation

Use the following ACME callback routines to manage blocks of memory:

- ACME$CB_ALLOCATE_ACME_VM
- ACME$CB_ALLOCATE_WQE_VM
- ACME$CB_DEALLOCATE_ACME_VM
- ACME$CB_DEALLOCATE_WQE_VM

Although other mechanisms might work, using these entry points brings the memory allocated by your ACME agent under the umbrella of the ACME server process debugging tools. This makes it easier to deal with problems involving ACME agents from multiple creators.

Using these memory allocation ACME callback routines also defends against memory leaks within your ACME agent in certain circumstances. The ACM dispatcher deallocates memory allocated by ACME$CB_ALLOCATE_WQE_VM after processing a particular request completes.

The ACM dispatcher never deallocates ACME-allocated memory except in response to a specific deallocation request from the ACME agent.

### 2.2.4 ACME-Specific Resources

An **ACME-specific resource** is an entity that your ACME agent defines and then caches with the ACME server main image by a call to ACME callback routine ACME$CB_RELEASE_RESOURCE.

Your ACME agent can use a subsequent call to ACME callback routine ACME$CB_ACQUIRE_RESOURCE to retrieve the ACME-specific resource. If that callback indicates there are no such ACME-specific resources available, your ACME agent can wait until one is available by returning ACME$_ WAITRESOURCE.

The ACME server main image has no information regarding the nature of the ACME-specific resource, it merely caches it and releases it in a thread-safe manner at the direction of the ACME agent that created it. The reason for using this capability is that if there is no ACME-specific resource left, the ACM dispatcher can stall activity on a given request until an ACME-specific resource becomes available (due to being released by a thread working on a different request).

### 2.2.5 ACME Process Control

Your ACME agent should not invoke any process control system service affecting its own process, such as the following:

SYS$EXIT
SYS$FORCEX
SYS$SETAST
SYS$SETDDIR
SYS$SETDFPROT
SYS$SETEXV
SYS$SETPRN
SYS$SETPRT
SYS$SETPRT_64
SYS$SETPRV
SYS$SETRWM
SYS$SETSTK
SYS$SETSWM
SYS$SETUP_AVOID_PREEMPT
SYS$SET_IMPLICIT_AFFINITY
SYS$SUSPND
SYS$RESUME
SYS$CANWAK
SYS$HIBER
SYS$SCHDWK
SYS$WAKE

## 2.3 ACME Callout Routine Dispatching

When the ACM dispatcher calls an ACME callout routine in your ACME agent shareable image, your code can call other (non-blocking) routines, but must eventually return to the ACM dispatcher with a status code. In returning, there are special meanings conveyed to the ACM dispatcher, if your ACME callout routine returns one of the following status codes:

- ACME$_CONTINUE

  The ACME callout routine has completed processing for this request. Proceed to subsequent ACME agents' callout routines.

- ACME$_WAITAST

  Call this ACME callout routine back (at non-AST level) after the completion of the next AST (for this ACME agent).

- ACME$_WAITRESOURCE

  Call this ACME callout routine back when the next ACME-specific resource (for this ACME agent) becomes available of the type code for which the most recent ACME$RESOURCENOTAVAIL was returned.

- ACME$_PERFORMDIALOGUE

  Send all dialogue data that has been queued by a call to ACME callback routine ACME$CB_QUEUE_DIALOGUE back to the ACM client process and call this ACME callout routine back when there is either a response (for input) or an acknowledgement (for output) available from the client.

- Other code

  Complete the request with this status.

When the ACME server main image calls the ACME callout routines provided by various ACME agents, there are three different dispatch patterns, as described in the next three sections.

### 2.3.1 To ACME Agent Control Callout Routines

When a privileged user invokes the SET SERVER ACME command, depending on command qualifiers, the ACM dispatcher calls one of the following ACME agent control callout routines for one ACME agent repeatedly until it returns ACME$_CONTINUE or a failure:

- ACME$CO_AGENT_INITIALIZE
- ACME$CO_AGENT_STARTUP
- ACME$CO_AGENT_STANDBY
- ACME$CO_AGENT_SHUTDOWN

Then the ACM dispatcher moves on to the next ACME agent. The ACM dispatcher finishes when all ACME agents have completed. It returns results to the process that issued the SET SERVER ACME command.

### 2.3.2 To ACME Event and Query Callout Routines

When an ACM client process calls the SYS$ACM[W] system service with a function code of ACME$_FC_EVENT or ACME$_FC_QUERY the ACM dispatcher calls the corresponding ACME event and query callout routines:

- ACME$CO_EVENT
- ACME$CO_QUERY

The ACM dispatcher calls only that ACME callout routine provided by the ACME agent specified by the client's ACME$_TARGET_DOI_ID or ACME$_TARGET_DOI_NAME item. It calls this repeatedly until done, and then returns results to the ACM client process.

### 2.3.3 To ACME Authentication and Password Callout Routines

Except as noted in this section, the ACME server main image processes each Authenticate Principal or Change Password request by calling each ACME callout routine in the order specified in Table 6–1. Where multiple ACME agents provide the same ACME callout routine (as is often the case), the ACME server main image calls the ACME callout routine from each ACME agent until it returns ACME$_CONTINUE or a failure.

#### 2.3.3.1 Failure of an Authentication/Password Request

Before processing reaches the ACME callout routine ACME$CO_AUTHENTICATE, any failure code that is returned by any ACME agent is simply noted without disturbing the dispatching activity. This supports the goals outlined in Section 2.5.

Once processing reaches the ACME callout routine ACME$CO_AUTHENTICATE, all authentication-related user interaction has been completed. Any failure code that has been returned by any ACME agent (including one noted from an earlier ACME callout routine) causes processing to skip to the end of dispatching, where each ACME agent gets called at the required ACME callout routine ACME$CO_FINISH.

#### 2.3.3.2 Free Context

The following two situations reseult in early termination of the dispatch cycle:

- The client program aborts the Authenticate Principal or Change Password request while it is handling a Dialogue request by sending function code ACME$_FC_FREE_CONTEXT (or the client program exits, with the same effect).
- The system manager uses the command SET SERVER ACME/ABORT to force processing to stop.

As in the case specified in Section 2.3.3.1, each ACME agent is called at ACME callout routine ACME$CO_FINISH. The first of the two cases previously listed can only happen when a dialogue request is pending. It is particularly important for the ACME agent that had an outstanding dialogue request to clean up the ACME-specific resource, other ACME-specific resource, general resource, memory and AST context usage, since it was not between ACME callout routines, like other ACME agents, and might have temporary allocations.

If a client aborts, your ACME agent might not get called for several ACME callout routines. If you have work to do in these situations (such as auditing), your ACME agent can depend on always getting called for ACME callout routine ACME$CO_FINISH.

#### 2.3.3.3 New Password Retry

An ACME agent solicits or obtains a new password in either of the following ACME callout routines:

- ACME$CO_NEW_PASSWORD_1
- ACME$CO_NEW_PASSWORD_2

The ACME agent that has received a password it finds acceptable, stores it in the work queue entry for access by other ACME agents. Then, the ACM dispatcher calls all other ACME agents to ensure it is acceptable to them.

All ACME agents have an opportunity to veto the new password as unacceptable in the appropriate ACME callout routine:

- ACME$CO_QUALIFY_PASSWORD_1
- ACME$CO_QUALIFY_PASSWORD_2

Each ACME agent is expected to test the user-supplied password against its own criteria and, if there is a problem, do the following:

1. Report the reason the password is unacceptable via a dialogue message to the ACM client process.
2. Return with the status code ACME$_RETRYPWD.

When the ACM dispatcher receives the status code ACME$_RETRYPWD, it loops back to continue execution at the corresponding prior ACME authentication and password callout routine:

- ACME$CO_NEW_PASSWORD_1
- ACME$CO_NEW_PASSWORD_2

Only after a full cycle of no ACME agent reporting a problem is the pair of steps declared done.

Once there is agreement among all concerned ACME agents, each concerned ACME agent should set the password into its own records using the ACME authentication and password callout routines ACME$CO_ACCEPT_PASSWORDS and ACME$CO_SET_PASSWORDS.

## 2.4 Beyond Dispatching

The information in this section pertains to ACME authentication and password callout routines. There are certain cases where your ACME agent must omit the normal processing associated with the purpose of an ACME callout routine even though it is called at that routine. In those cases it is reasonable and expected that your ACME agent might continue to perform any other processing it normally carries out during that ACME callout routine where such processing is unrelated to the nominal purpose of the ACME callout routine.

### 2.4.1 Preauthentication

Your ACME agent must make its own tests for the ACMEWQEFLG$V_ PREAUTHENTICATED flag described in Section 4.2.2.1 and, if it is set, your ACME agent must eschew all authentication-related activity in the following ACME callout routines:

- ACME$CO_ANCILLARY_MECH_1
- ACME$CO_PASSWORD_1

- ACME$CO_ANCILLARY_MECH_2
- ACME$CO_PASSWORD_2
- ACME$CO_ANCILLARY_MECH_3
- ACME$CO_AUTHENTICATE

It may not be possible for your ACME agent to issue credentials in this situation. The ACMEWQEFLG$V_PREAUTHENTICATED flag described in Section 4.2.2.1 is used in the creation of certain detached processes where authentication is not feasible, such as DECnet proxy logins and initialization of batch jobs.

### 2.4.2 Phase Done

Setting the ACMEWQEFLG$V_PHASE_DONE flag described in Section 4.2.2.1 indicates completion described in Section 4.2.2.1 of an ACME callout routine with respect to **well-known items** for that ACME callout routine. Other ACME agents should honor that flag and not perform the business of that ACME callout routine. But ACME agents might have additional activity overloaded onto an ACME callout routine, and they are free to pursue those actions.

## 2.5 Concealing Authentication Details

From the original OpenVMS release at the end of the 1970's, security policy has been to keep the reasons why login failed a secret from the person attempting to log in.

As part of the TCB, your ACME agent must follow the standards for keeping authentication failure details secret from those who have not yet been authenticated. The measures to take fall into the following two basic groups:

- Avoid explicit explanations via status codes.
- Avoid implicit explanations via interaction patterns.

These are explained in detail below.

### 2.5.1 Return Status Codes

In the event of an authentication failure, your ACME agent should do the following:

1. Indicate the real cause of failure by a call to ACME callback routine ACME$CB_SET_2ND_STATUS [1] providing a true error code for return to the caller ACMSB field ACMESB$L_SECONDARY_STATUS.
2. Return the sanitized error code ACME$_AUTHFAILURE to the ACME server process so it can return it to the caller ACMSB field ACMESB$L_STATUS.

The SYS$ACM[W] system service ensures that only the non-revealing ACME$_ AUTHFAILURE code is delivered to unprivileged callers.

In other failure cases, such as where authentication checks succeed but authorization checks fail, your ACME agent should return some status other than ACME$_AUTHFAILURE , and use the **secondary status** for a subsidiary message, if appropriate. The VMS ACME, for instance, under the appropriate circumstances can return one of the following errors as the primary status, visible even to unprivileged callers:

[1] Calls to either ACME callback routine ACME$CB_SET_2ND_STATUS or ACME callback routine ACME$CB_SET_ACME_STATUS have the side effect of returning the ACME ID of your ACME agent to the caller ACMSB field ACMESB$L_ACME_ID.

```
%LGI-F-LOGDISABL, logins are currently disabled; try again later
%LGI-F-BADHOUR, you are not authorized to login at this time
%LGI-F-BADDAY, you are not authorized to login today
%LGI-F-RESTRICT, you are not authorized to login from this source
%LGI-F-ACNTEXPIR, your account has expired; contact your system manager
%ACME-F-PWDEXPIRED, your password has expired; contact your system manager
%LGI-F-FRCPWDERR, error changing expired password
```

### 2.5.2 Interaction Patterns

As noted in Section 2.3.3.2, there are two mechanisms that terminate an Authenticate Principal or Change Password request at any stage of request processing:

- An ACME$_FC_FREE_CONTEXT function code from the ACM client process
- Management intervention through the ACME server main image

Aside from those cases, the ACME server main image calls each ACME callout routine you provide between the following ACME callout routines in the order specified in Chapter 6, regardless of any failure returned by an ACME agent:

- ACME$CO_INITIALIZE
- ACME$CO_AUTHENTICATE

The client interaction that may take place in the following ACME callout routines is still possible when some ACME authentication and password callout routine has returned a failure code, and your ACME agent should pursue that interaction:

- ACME$CO_SYSTEM_PASSWORD
- ACME$CO_PRINCIPAL_NAME
- ACME$CO_VALIDATE_MAPPING
- ACME$CO_ANCILLARY_MECH_1
- ACME$CO_ANCILLARY_MECH_2
- ACME$CO_PASSWORD_2
- ACME$CO_ANCILLARY_MECH_3

Your ACME agent should behave in a “typical” fashion in the above circumstance. (For example, the VMS ACME behaves as though the specified principal name existed and used a single password.)

## 2.6 Dialogue Support for ACME Authentication and Password Callout Routines

Your ACME agent can make its own tests of the ACMEWQE$L_DIALOGUE_FLAGS field described in Section 4.2.3. If necessary capabilities are missing, your ACME agent can return the failure code ACME$_INSFDIALSUPPORT to the ACM dispatcher for ultimate return as the primary status to the SYS$ACM[W] system service caller.

Alternatively, your ACME agent can simply call ACME callback routine ACME$CB_QUEUE_DIALOGUE and use the fact that it returns ACME$_INSFDIALSUPPORT as a basis for taking an alternate course of action.

Except in the event of failures after the ACME callout routine ACME$CO_AUTHENTICATE, the ACME server main image calls each ACME callout routine in order.

The ACM dispatcher ensures that the only item codes that a client adds to a request after the first call are those for which the client was prompted. For common item codes, an ACME agent can rely on the fact that the only ones added along the way are responses to prompts for well-known items.

Whenever your ACME agent prompts for a well-known item, it should preserve secrecy regarding which ACME agent has asked for the input by using the same prompt any other ACME agent would use. To do this, determine the prompt text from the appropriate OpenVMS message value, stripped of facility severity and code indications, according to the following table:

| Well-Known Item | Item Code | Message Code for Prompt Text | |
|---|---|---|---|
| | | Authenticate Principal | Change Password |
| System password | ACME$_SYSTEM_PASSWORD | None | LGI$_OLDPASS |
| Principal Name In | ACME$_PRINCIPAL_NAME_IN | LGI$_USERNAME | LGI$_USERNAME |
| Password 1 | ACME$_PASSWORD_1 | LGI$_PASSWORD | LGI$_OLDPASS |
| Password 2 | ACME$_PASSWORD_2 | LGI$_PASSWORD | LGI$_OLDPASS |
| New password 1 | ACME$_NEW_PASSWORD_1 | LGI$_NEWPASS | LGI$_NEWPASS |
| Confirm password 1 | ACME$_NEW_PASSWORD_1 | LGI$_CHKPASS | LGI$_CHKPASS |
| New password 2 | ACME$_NEW_PASSWORD_2 | LGI$_NEWPASS | LGI$_NEWPASS |
| Confirm password 2 | ACME$_NEW_PASSWORD_2 | LGI$_CHKPASS | LGI$_CHKPASS |

When your ACME agent requests input of a principal name, the maximum length of ACME$K_MAXCHAR_PRINCIPAL_NAME applies, and your ACME agent can use a buffer of limited size (perhaps allowing a few extra characters for leading and trailing spaces).

When your ACME agent requests input of a password, the maximum size is the length that can be described in a 16-bit word. Your ACME agent must be prepared to handle a very long string for sharing with other ACME agents.

### 2.6.1 Text Output

Where there is any chance of user confusion, your ACME agent should ensure that any output text clearly indicates the context. For instance, the VMS ACME already provides output text for Last interactive login and Last non-interactive login, corresponding to the values it passes to ACME authentication and password callout routine ACME$CB_SET_LOGON_STATS_VMS. If your ACME agent is going to provide an additional "Last Login" date, it should indicate the context, such as:

```
Last Login with the Vulcan Mind-Meld technique was 25-Feb-2184
```

When your ACME agent interacts with the ACME server main image, text strings must use the **UCS encoding**. To send text data, queue the output using an ACME-specific message category that has bit 14 set, and thus is susceptible to UCS-Latin1 conversion.

**Warning**

Because UCS supports a broader range of characters than Latin-1, translations of arbitrary UCS characters may be impossible. In this case, a substitute symbol is used. This will probably not result in the desired effect. ACME agents that generate such unmappable characters should

refrain from sending them to ACM client programs that have not specified the ACME$M_UCS2_4 function modifier.

---

### 2.6.2 Binary Output

Your ACME agent can send irregular **itemset entries** (such as those specifying binary data) to clients, but your ACME should only do this to particular clients that have special knowledge regarding the message category for those itemset entries.

To send binary data, queue the output using an ACME-specific message category that does not have bit ACMEMC$V_UCS set, and thus is not susceptible to UCS-Latin1 conversion.

### 2.6.3 Binary Input

Your ACME agent can request irregular input (such as that containing binary data) from clients, but your ACME agent should only do this to particular clients that have special knowledge regarding the item codes for those requests.

To request binary data input, use an ACME-specific item code that does not have bit ACMEIV$V_UCS set, and thus is not susceptible to UCS-Latin1 conversion.

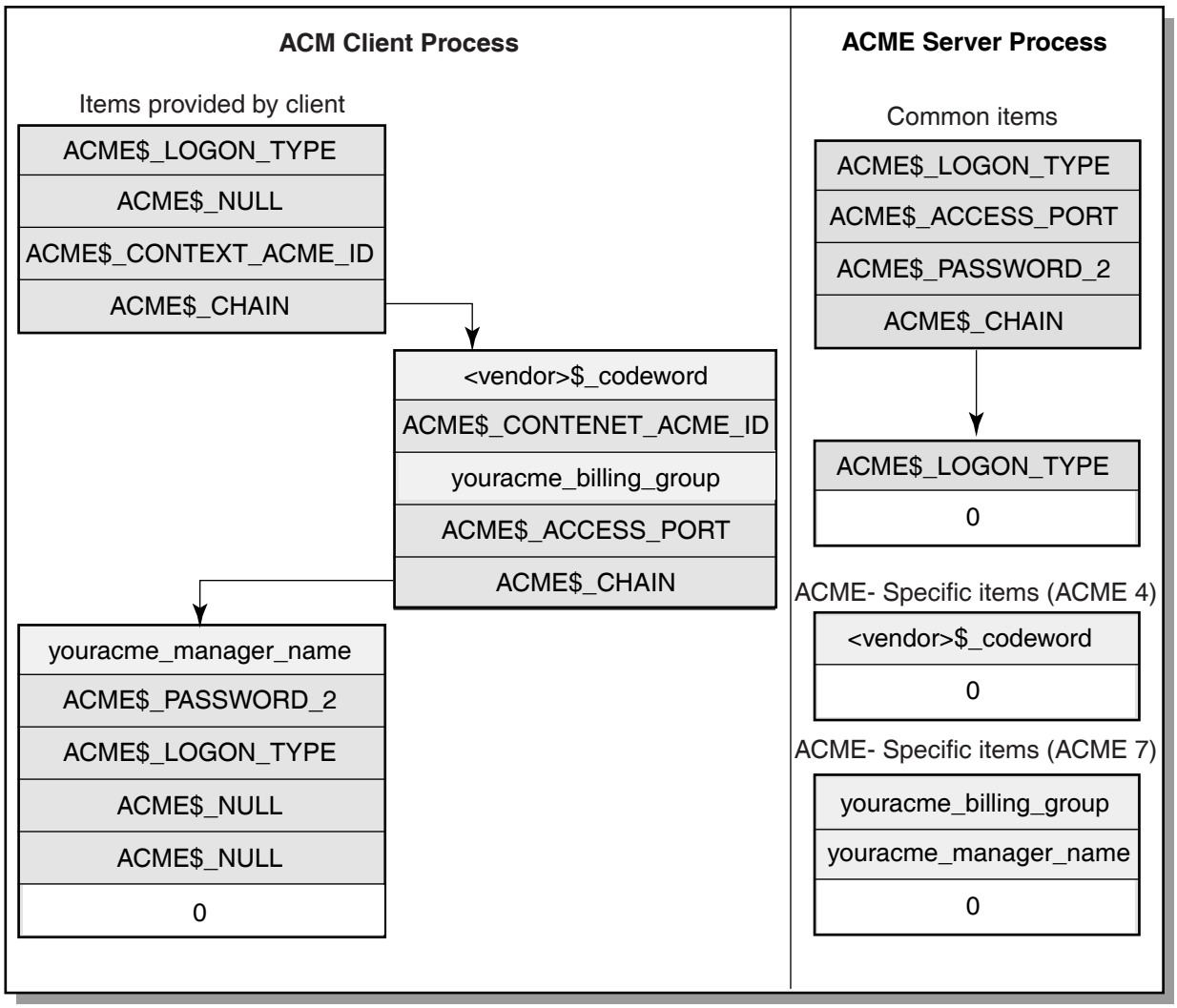
## 2.7 Item Code Support

The caller of the SYS$ACM[W] system service provides a single **item list** with common items and ACME-specific items intermixed, but that is split up by the ACME server main image and presented to each ACME agent as the following two separate chains of **item list segments**:

- Common items
- ACME-specific items

The ACME-specific items presented to each ACME agent are only those specifically intended for that ACME agent. Those intended for other ACME agents are filtered out, as shown in Figure 2–1.

**Figure 2–1 Item List Processing**




VM-0782A-AI

## 2.7.1 Using Input Item Code Requests

Within a chain of item list segments, use the last rendition of an input-item as the defining occurence. This is required (except for any ACME-specific input items for which multiple values are actually honored in a single call) in order to provide callers of SYS$ACM[W] with the same item list semantics for all ACME agents. Note that the following parameters, provided by the ACM dispatcher to your ACME agent's callout routines, give you a pointer to the results of the current interactions with the client:

- ITEM_LIST
- ACME_ITEM_LIST

Because those interactions are guaranteed to include items that your ACME agent requested during this ACME authentication and password callout routine, using those parameters provides a handy way to speed processing compared to using the following WQE entries:

- ACMEWQE$PS_ITEMLIST
- ACMEWQE$PS_ACME_ITEMLIST

#### 2.7.1.1 Input for Well-Known Items

Your ACME agent should engage in dialogue to obtain well-known items only in the ACME authentication and password callout routine designated for each item according to the following table:

| Well-Known Item | Item Code | ACME Callout Routine |
|---|---|---|
| System password | ACME$_SYSTEM_PASSWORD | ACME$CO_SYSTEM_PASSWORD |
| Principal Name In | ACME$_PRINCIPAL_NAME_IN | ACME$CO_PRINCIPAL_NAME |
| Password 1 | ACME$_PASSWORD_1 | ACME$CO_PASSWORD_1 |
| Password 2 | ACME$_PASSWORD_2 | ACME$CO_PASSWORD_2 |
| New Password 1 | ACME$_NEW_PASSWORD_1 | ACME$CO_NEW_PASSWORD_1 |
| New Password 2 | ACME$_NEW_PASSWORD_2 | ACME$CO_NEW_PASSWORD_2 |

The ACME authentication and password callout routine should engage in this dialogue only if all of the following are true:

1. This ACME agent needs the item.
2. The initial call to the SYS$ACM[W] system service did not include the item in the item list.
3. No other ACME agent has prompted for the item yet (and so indicated by setting flag ACMEWQEFLG$V_PHASE_DONE.[2]
4. If the well-known item is a password of any sort (anything other than ACME$CO_PRINCIPAL_NAME), this ACME agent is the designated DOI.

Your ACME agent should engage in dialogue to obtain ACME-specific items only when one of the following is true:

- The ACME authentication and password callout routine is not designated for any well-known item.
- This ACME is the one that has engaged in the dialogue to obtain the well-known item for the ACME authentication and password callout routine and the ACME- specific item is associated with the well-known item (such as the confirmation of the selection of a new password).

[2] In the case of principal name, the relevant flag to check is subfield ACMEWQEITM$L_ACME_ID of field ACMEWQEAX$R_PRINCIPAL_NAME since the principal name might be set in an earlier ACME callout routine than ACME$CO_PRINCIPAL_NAME.

#### 2.7.1.2 Reporting Input Item Code Errors

When an item list entry provided by the client is inadequate, your ACME agent should return the following status values:

| Status | Meaning |
|---|---|
| SS$_BADITMCOD | An ACME-specific item code is undefined or is inappropriate in the circumstance (for example, incompatible with the function code or another item). Alternatively, a required item code is not provided.<br>Common item codes undefined at the time your ACME agent is built should be ignored, since they might be added later on by even a patch to an existing version of OpenVMS. |
| SS$_BADBUFLEN | An item length is wrong for the item code used. |
| SS$_BADPARAM | The contents of an item are incorrect for the circumstance. |
| SS$_NO<priv> | The client does not have a single required privilege needed for an operation. |
| SS$_NOPRIV | The client does not have any of a set of privileges which would suffice for an operation. |

When your ACME agent returns any of the first three status values in connection with a specific item, it should return the item code in the ACME-specific status by calling ACME callback routine ACME$CB_SET_ACME_STATUS. That not only indicates which item was faulty to the client, but also indicates which ACME agent found fault with the item.

### 2.7.2 Fulfilling Output Item Code Requests

The ACME server main image silently prevents the return of output item code results in the event of a failure. Your ACME agent can store output item code results using the ACME callback routine ACME$CB_SET_OUTPUT_ITEM during any ACME callout routine. They are not returned to the client program until after the final ACME callout routine processing, and then only if the final status indicates success.

#### 2.7.2.1 Special Output Items for ACME Authentication and Password Callout Routines

It is actually the ACME server main image that will satisfy client requests for the following output items:

- ACME$_LOGON_INFORMATION

  During processing, the ACME agent that supports the designated DOI calls ACME$CB_SET_LOGON_STATS_DOI to store data in the ACMEWQEAX$R_LOGON_STATS_DOI field of the WQE extension for authentication.

- ACME$_MAPPED_VMS_USERNAME

  During processing, the ACME agent that maps the principal name calls ACME$CB_SET_WQE_PARAMETER to store an OpenVMS user name into the ACMEWQEAX$R_VMS_USERNAME field of the WQE extension for authentication.

- ACME$_PERSONA_HANDLE_OUT

  During processing, the ACME agents that support DOIs call ACME callback routine ACME$CB_ISSUE_CREDENTIALS, providing it with credentials for the SYS$ACM[W] system service to amalgamate into a persona for the client.

- ACME$_PRINCIPAL_NAME_OUT

During processing, the ACME agent that determines the principal name calls ACME$CB_SET_WQE_PARAMETER to store a normalized (according to the rules of that ACME agent) rendition of the principal name into the ACMEWQEAX$R_PRINCIPAL_NAME_OUT field of the WQE extension for authentication;

#### 2.7.2.2 Normal Output Items

Specify the results as they become known to your ACME agent. The ACM dispatcher returns them to the SYS$ACM client at the end of successful processing.

Your ACME agent may return different results for the same item at successive stages of the processing. Only the last value for each item is returned. This is *different* from saying the same thing for a given item *code*.

If you need to communicate to a cooperating client program before processing of the request is complete, use the ACME callback routine ACME$CB_QUEUE_DIALOGUE with an ACME-specific output data type known to the client. You can reliably determine the identity of an installed ACM client program from the ACMEWQE$R_SERVICE_NAME field.

## 2.8 Auditing Within an ACME Callout Routine

If you want to audit via the use of SYS$AUDIT_EVENT, SYS$CHECK_PRIVILEGE or a similar system service that provides an AUDIT_FLAGS parameter, you should specify NSA$M_SERVER in that parameter to cause auditing to be done even though the context is that of a server in the TCB.

## 2.9 Writing to the ACME$SERVER Log File

Unexpected operational problems (except those that could be caused by faulty input from an end user) should be logged to the ACME$SERVER log (whose default location is SYS$MANAGER).

If the problem prevents completion of the request, do the following:

1. Indicate the real cause of failure by a call to ACME callback routine ACME$CB_SET_2ND_STATUS which provides one of the following:
   - A failure code from Section 8.4
   - A failure code specific to your ACME agent
2. Return the code ACME$_CONTACTSYSMGR (not ACME$_AUTHFAILURE) to cause the system manager to investigate the details your ACME agent stored in the ACME$SERVER log.

## 2.10 ACME Callout Internationalization

Core OpenVMS services allow for internationalization on a per-process basis through the redefinition of SYS$MESSAGE. Within the ACME server process, only one natural language setting of this nature is possible, just as with all processes.

## 2.11 Password Policy ACME Agents and Older Password Policy Models

If your only goal in writing an ACME agent is to enforce additional local restrictions on the selection of new passwords, then you can get by with only the following three ACME authentication and password callout routines:

- ACME$CO_QUALIFY_PASSWORD_1
- ACME$CO_QUALIFY_PASSWORD_2
- ACME$CO_FINISH

For more information on writing ACME authentication and password callout routines, see Chapter 6.

## 2.12 ACME Agent Design Alternatives

In many cases those who want to write an ACME agent need to interface to an external mechanism that already has particular rules of operation. The rules described in this chapter should allow you sufficient flexibility to bridge the two environments.

### 2.12.1 Separate Qualification of Proposed New Passwords

You will find you can write a much more interoperable ACME agent if you separate the following two notions:

- Checking the acceptability of a proposed new password
- Setting the new password

Some ACME agents are constrained from providing that separation by protocols on an external network over which they communicate. Where this is not an issue, keep the two actions separate.

### 2.12.2 Using a Separate Process

If some of the rules described in this chapter are not compatible with a technique you want to use (such as calling an existing library that expects some other environment), you might choose to do the following:

1. Create a detached process with that environment.
2. Communicate with that process from your ACME agent using the Intra-Cluster Communication (ICC) system services discussed in the *OpenVMS Programming Concepts Manual*.

## 2.13 Naming Your ACME Agent

Suitable system manager commands can load a properly built ACME agent, regardless of its name. For maximum ease-of-use your ACME agent should have a name in the following form:

```
<facility-prefix><acme-name>_ACMESHR.EXE
```

where:

- <facility-prefix> is the facility prefix name for your ACME agent, ending with a dollar sign for registered facilities and with an underscore for local facilities
- <acme-name> is the simple name of your ACME agent

- _ACMESHR.EXE is an underscore followed by the standard termination and extension for ACME agent shareable images

For example, if you register the facility MYCORP and write a DNA ACME agent, the name might be:

```
MYCORP$DNA_ACMESHR.EXE
```

Writing the same code for local use, the name might be:

```
OURSTUFF_DNA_ACMESHR.EXE
```

For maximum ease of configuration, store your ACME agent in SYS$COMMON:[SYSLIB].

# 3
# Testing and Debugging Your ACME Agent

Under actual operation, your ACME agent might encounter circumstances that you did not anticipate. This chapter outlines the tools and techniques available to test and debug ACME agents.

## 3.1 Using ACME Tracing

In addition to the logging described in Section 2.9, a series of trace flags exists (expressed here with the corresponding bit-masks):

- 0 - agent—Enable agent tracing
- 1 - general—General (non-specific) tracing
- 2 - vm—Virtual memory operations
- 3 - ast—AST processing
- 4 - wqe—WQE parameter values
- 5 - report—Agent status/attrbutes operations
- 6 - message—Messaging operations
- 7 - dialogue—Dialogue operations
- 8 - resource—ACME-specific resource operations
- 9 - callout—Invocations of any ACME callout routines
- 10 - callout_status—Status returned from any ACME callout routines

To set some combination of these trace flags, use a command similar to the following:

```
SET SERVER ACME/TRACE=%x146
```

The WQE flag ACMEWQEFLG$V_TRACE_ENABLED mirrors the state of flag value 1. Your ACME agent can test that flag and, if it is set, use the ACME$CB_SEND_LOGFILE callback routine to send extra tracing information to the ACME$SERVER log file. Particularly if your ACME agent will be used at another site, it is useful to log additional information specific to your ACME agent.

You can also use the other trace flags to send additional information to the ACME$SERVER log file to provide a context for unexpected behavior you might observe.

## 3.2 Using the Debugger

You can debug your ACME agent using the OpenVMS Debugger. Compile and link your agent code with debugger switches. From a privileged process, run the ACME server image, as follows:

```
$  RUN SYS$SYSTEM:ACME.SERVER
```

Once the ACME server is running, issue the following commands from a second terminal session:

```
$  SET SERVER ACME/CONF=(NAME=VMS,CRED=VMS)
$  SET SERVER ACME/CONF=(NAME=your-acme,CRED=your-cred)
$  SET SERVER ACME/ENABLE
```

Enter the debugger in the first terminal session by typing:

```
<Ctrl/Y>
$  DEBUG
DBG>  SET IMAGE your-image
DBG>  SET MODULE your-module
DBG>  SET BREAK your-bpt
DBG>  GO
```

# 4
# ACME Agent Data Structure

Cooperation between software from multiple providers to achieve a single coherent authentication environment requires careful use of common data structures. In general, your ACME agent can read these structures directly, but should modify them only by calling specific ACME callback routines.

## 4.1 ACME Server Process Data Types

This section describes some of the data structures used within the ACME server process. As with client software calling the SYS$ACM[W] system service, it is important to properly handle the version indications within those data structures that contain revision level fields.

### 4.1.1 Revision Level Fields to Check

Possible major and minor revision level values provided by the ACME server main image for use in ACME agents include the following:

- ACMEWQE$W_REVISION_LEVEL

  The major and minor subfields can contain the following values:

  - ACMEWQE$K_MAJOR_ID_001 together with ACMEWQE$K_MINOR_ID_001

    All data elements, as described in Section 4.2

  - ACMEWQE$K_MAJOR_ID_001 together with ACMEWQE$K_MINOR_ID_000

    Data elements described in Section 4.2, but without the following cells:

    ACMEWQE$L_TIMEOUT
    ACMEWQE$L_FACTOR

- ACMEKCV$W_ACM_REVISION_LEVEL

  The major and minor subfields can contain the following values:

  - ACME$K_MAJOR_ID_001 together with ACME$K_MINOR_ID_000

    The interface to the ACM dispatcher, as described in this manual.

- ACMEKCV$W_REVISION_LEVEL

  The major and minor subfields can contain the following values:

  - ACMEKCV$K_MAJOR_ID_001 together with ACMEKCV$K_MINOR_ID_000

    The interaction rules described in this manual.

Your ACME agent should test against those values that represent the level with which your code is compatible.

It is sufficient to check those revision fields only during ACME$CO_AGENT_INITIALIZE. If your ACME agent encounters a mismatch, callbacks are inadvisable. Your ACME agent should return ACME$_UNSUPREVLVL.

A difference in major ID indicates an incompatible change in structure layout or calling protocol such that you must specifically program your ACME agent to respond to different major IDs.

A difference in minor ID indicates additional fields in a structure, or additional calling protocol options, that will not affect an ACME agent that behaves appropriately for earlier minor ID values.

### 4.1.2 Revision Level Fields to Set

Possible major and minor revision level values for use by ACME agents include the following:

- ACMELIDOI$W_REVISION_LEVEL
  - ACMELIDOI$K_MAJOR_ID_001 together with ACMELIDOI$K_MINOR_ID_000

    All data elements, as described in Section A.5.
- ACMELIVMS$W_REVISION_LEVEL (only used by the VMS ACME)
  - ACMELIVMS$K_MAJOR_ID_001 together with ACMELIVMS$K_MINOR_ID_000

    All data elements, as described in Section A.6.
- ACMERSRC$W_REVISION_LEVEL
  - ACMERSRC$K_MAJOR_ID_001 together with ACMERSRC$K_MINOR_ID_000

    All data elements, as described in Section B.5.

Your ACME agent should generate those values that represent the level with which your code is compatible.

### 4.1.3 ACMEID Data Type, Also Used by SYS$ACM[W] System Service Callers

These data types are defined in STARLET and are also used by programs that call the SYS$ACM[W] system service.

ACMEID, a longword data type, is the base type for several items in calls to the SYS$ACM[W] system service and for several fields within the work queue entry, as discussed in Section 4.2.

It is divided into the following three subfields:

- ACMEID$V_ACME_NUM

  The 15-bit index number corresponding to the name of a particular ACME agent. This number does not vary on a given machine until the next reboot, providing continuity for SYS$ACM[W] system service callers even though an ACME server process may be replaced.
- ACMEID$V_DOI_DESIGNATOR

  The 1-bit flag indicating that a particular ACME agent has been registered via the command SET SERVER ACME as able to provide credentials. This means the ACME agent supports a DOI.
- ACMEID$V_SERVER_NUM

The incarnation number of the particular ACME server process handling a request.

Client programs that call the SYS$ACM[W] system service and use the ACMEID data type often use only the ACMEID$V_ACME_NUM field to perform comparisons, while within an ACME callout routine it is more typical to compare the entire longword of the ACMEID type. In fact, the ACME agent writer usually only needs to be concerned about whether a given value is equal to the ACMEWQE$L_CURRENT_ACME_ID value (that indicating this ACME agent itself), equal to zero, or something else. The exact value of "something else" is rarely of interest.

### 4.1.4 ACMEWQEITM Data Type, Unique to the ACME Server Process

These data types are defined in LIB and are only useful to those writing an ACME agent. Although the fact that they are defined in LIB makes them susceptible to change on subsequent releases, it is an OpenVMS development goal to minimize incompatible changes since the software making use of them (ACME agent shareable images) is user mode code.

The ACMEWQEITM data type (see Section B.14) describes a UCS encoding of a string item. The length can be arbitrarily long (although the initial SYS$ACM[W] system service and ACME server main image implementations limit it to 65535 bytes), so the data type does not actually contain a string, but rather points to a string stored elsewhere. The ACMEWQEITM data type contains the following subfields:

- ACMEWQEITM$L_ACME_ID

  The ACME ID of the ACME agent that most recently set this item. Zero indicates the item has never been set.

- ACMEWQEITM$L_PHASE

  The number of the ACME authentication and password callout routine in which this item was most recently set. ACME authentication and password callout routine numbering is private to the ACME server main image and can change from one release to the next as new ones are added. The only characteristics of an ACME callout routine usable by an ACME agent are the following:

  – The number for a particular ACME callout routine remains the same for a given execution of the ACME server main image.

  – The numbers of ACME callout routines are monotonically increasing according to the first encounter of each ACME authentication and password callout routine.

  Thus, the usefulness of the ACME authentication and password callout routine numbering is mainly limited to testing and debugging.

- ACMEWQEITM$L_LENGTH

  The length of the item string, in bytes (four times the number of characters).

- ACMEWQEITM$PS_POINTER

  The address of the item string in memory allocated and deallocated by the ACME agent shareable image that set the item.

Your ACME agent can read data directly from the ACMEWQEITM type and the string to which it points. But, to avoid problems with abnormally long strings, it is best to perform comparisons against the existing string rather than to make a copy of it.

Your ACME agent can change data in those instances of the ACMEWQEITM type, described in Section 4.2.21.3, by calling ACME callout routine ACME$CB_SET_WQE_PARAMETER.

## 4.2 Work Queue Entry Data Fields

This section describes fields of the work queue entry, listed in order of decreasing interest to the writer of a typical ACME agent. An ACME agent can read these fields directly. An ACME agent can also change certain fields, but only through the use of specific ACME callback routines.

### 4.2.1 Function Field

ACMEWQE$L_FUNCTION, the function field, is of data type ACMEFC, and is divided into two subfields, described below.

#### 4.2.1.1 ACMEFC$V_FUNCTION

This subfield contains the function code specified on the call to the SYS$ACM[W] system service. As documented in the *OpenVMS System Services Reference Manual*, that code will be one of the following:

- ACME$_FC_AUTHENTICATE_PRINCIPAL
- ACME$_FC_CHANGE_PASSWORD
- ACME$_FC_RELEASE_CREDENTIALS
- ACME$_FC_QUERY
- ACME$_FC_EVENT
- ACME$_FC_FREE_CONTEXT

#### 4.2.1.2 ACMEFC$V_MODIFIERS

This subfield contains the set of function modifiers specified on the call to the SYS$ACM[W] system service. As documented in the *OpenVMS System Services Reference Manual*, the valid modifiers are as follows:

- ACME$M_NOAUDIT
- ACME$M_TIMEOUT
- ACME$M_UCS2_4
- ACME$M_ACQUIRE_CREDENTIALS
- ACME$M_MERGE_PERSONA
- ACME$M_COPY_PERSONA
- ACME$M_OVERRIDE_MAPPING
- ACME$M_NOAUTHORIZATION
- ACME$M_FOREIGN_POLICY_HINTS
- ACME$M_DEFAULT_PRINCIPAL

### 4.2.2 Flags Field

ACMEWQE$L_FLAGS, the flags field, is of data type ACMEWQEFLG (see Section B.13) and contains flags in two sets, as described below.

#### 4.2.2.1 ACME Flags Field

ACMEWQEFLG$V_ACME_FLAGS is the ACME flags field. ACME flags are set by ACME agents calling ACME$CB_SET_WQE_FLAG. The following flags are potentially applicable to all requests:

- ACMEWQEFLG$V_PHASE_DONE

  For ACME authentication and password callout routines, this flag means that some ACME agent has completed the mandated activity for the current ACME authentication and password callout routine. Other ACME agents should only pursue their private activities. See the description of each specific ACME authentication and password callout routine in Chapter 6 for an indication of whether ACMEWQEFLG$V_PHASE_DONE is used for that routine.

  The ACM dispatcher clears this flag before calling the ACME agents for the next ACME authentication and password callout routine.

- ACMEWQEFLG$V_NO_RETRY

  An ACME agent is unable to retry some operation.

  For ACME authentication and password callout routines, if the original proposal is unsatisfactory, this flag means that some ACME agent is not capable of handling a different proposed new password. Your ACME agent should check this flag, if a proposed password fails qualification during one of the following ACME authentication and password callout routines:

  - ACME$CO_QUALIFY_PASSWORD_1
  - ACME$CO_QUALIFY_PASSWORD_2

  If your ACME agent finds the flag set, it should fail the authenticate principal or change password request rather than returning ACME$_RETRYPWD. To fail the request, your ACME agent should take the following steps:

  1. Call ACME callback routine ACME$CB_SET_2ND_STATUS, specifying one of the following:
     - A failure code from Section 8.3
     - A failure code specific to your ACME agent
  2. Return the code ACME$_INVNEWPWD.

- ACMEWQEFLG$V_PREAUTHENTICATED

  This request is implicitly authenticated based on its source, such as DECnet proxy logins and the initialization of batch jobs. No ACME agent should engage in any authentication activity.

  **Caution**

  Attempting to second-guess the decision of another ACME agent to set this flag is fraught with peril. For example, if your ACME agent were to provide some additional authentication involving out-of-band cryptographic handshakes, it could fail miserably in a case where there

is no “other end” such as traditional methods of starting DECwindows, DECnet or TCP/IP detached processes.

---

- ACMEWQEFLG$V_NO_EXTERNAL_AUTH

  A LOGINOUT.EXE user has explicitly specified /LOCAL_PASSWORD, so no ACME agent other than the VMS ACME should call the ACME callback routine ACME$CB_SET_DESIGNATED_DOI.

- ACMEWQEFLG$V_SKIP_NEW_PASSWORD

  No ACME agent should engage in new password processing, since at least one of the following occurred:

  – Item ACME$_AUTH_MECHANISM specified some non-zero value.

  – ACME$_LOGON_TYPE specified a **logon type** of either ACME$K_NETWORK or ACME$K_BATCH, where, by definition, password change dialogue is not possible.

#### 4.2.2.2 Dispatcher Flags Field

ACMEWQEFLG$V_DISPATCHER_FLAGS, the dispatcher flags field, contains the following flags set by the ACM dispatcher in response to various conditions not directly controlled by ACME agents:

- ACMEWQEFLG$V_DIALOGUE_POSSIBLE

  The call to the SYS$ACM[W] system service provided item ACME$_DIALOGUE_SUPPORT, so at least your ACME agent can queue output dialogue itemset entries. You can determine what sorts of input dialogue itemset entries to queue by examining subfields ACMEDLOGFLG$V_INPUT and ACMEDLOGFLG$V_NOECHO of field ACMEWQE$L_DIALOGUE_FLAGS.

- ACMEWQEFLG$V_AST_RECEIVED

  Reserved to HP.

- ACMEWQEFLG$V_MASK_STATUS

  When it returns final status to the ACM client program, the SYS$ACM[W] system service will mask field ACMESB$L_SECONDARY_STATUS if ACMESB$L_STATUS, if ACMESB$L_STATUS contains ACME$_AUTHFAILURE. This flag reflects the privilege state when the SYS$ACM[W] system service was called.

- ACMEWQEFLG$V_TRACE_ENABLED

  The agent tracing bit described in Section 3.1 was set, so this ACME agent should engage any tracing activity it has.

- ACMEWQEFLG$V_ABORT_REQUEST

  The ACM client process has called the SYS$ACM[W] system service with function code ACME$_FC_FREE_CONTEXT (or has performed an image exit, which has the same effect).

- ACMEWQEFLG$V_FAILED_REQUEST

  An ACME agent returned an ordinary (not reserved) non-success status, meaning the request will fail.

### 4.2.3 Dialogue Flags Field

ACMEWQE$L_DIALOGUE_FLAGS, the dialogue flags field, is of data type ACMEDLOGFLG and contains the set of flags specified by the ACM client program in item ACME$_DIALOGUE_SUPPORT. The following flags are currently defined:

- ACMEDLOGFLG$V_INPUT
- ACMEDLOGFLG$V_NOECHO

### 4.2.4 Status Field

ACMEWQE$R_STATUS, the status field, is of data type unsigned longword and contains the **primary status** that is currently scheduled to be returned to the caller of the SYS$ACM[W] system service in field ACMESB$L_STATUS.

If this is a success status, as indicated by the low bit being set, there is still a chance it could be replaced by a failure status before the completion of processing.

If multiple ACME agents return failure status, the first one to do so "wins" and has its failure status returned to the caller of SYS$ACM[W] system service. This prevailing ACME agent also has values (if any) it provided for the following cells:

- Secondary status - via ACME$CB_SET_2ND_STATUS
- ACME status - via ACME$CB_SET_ACME_STATUS

Any values provided by other ACME agents will be eliminated.

Whichever (if any) ACME agent ends up having its value used for the ACME status will also have its ACME ID returned to the ACM client program as the **ACME status ID**.

### 4.2.5 Secondary Status Field

ACMEWQE$R_SECONDARY_STATUS, the secondary status field, is of data type unsigned longword and contains the status that is currently scheduled to be returned to the caller of the SYS$ACM[W] system service in field ACMESB$L_ SECONDARY_STATUS.

If no ACME agent has yet specified a secondary status, there is still a chance it could be set before processing is completed.

### 4.2.6 ACME Status Field

ACMEWQE$R_ACME_STATUS, the ACME status field, is of data type unsigned longword and contains the status that is currently scheduled to be returned to the caller of the SYS$ACM[W] system service in field ACMESB$L_ACME_STATUS.

Except for the special cases described in Section 2.7.1.2, there is no general guarantee that this field is in the form of an OpenVMS status code.

If no ACME agent has yet specified an ACME status, there is still a chance it could be set before processing is completed.

### 4.2.7 AST Context Field

The ACMEWQE$Q_AST_CONTEXT field and the ACMDWQEFLG$V_AST_ RECEIVED flag are reserved for Hewlett-Packard.

### 4.2.8 Locale Field

ACMEWQE$R_LOCALE, the locale field, is of data type ACMEWQEITM (introduced in Section 4.1.4) and describes a UCS encoding of the ACME$_ LOCALE item provided on the call to the SYS$ACM[W] system service.

Most ACME agents have no particular use for this field, since it is for possible future internationalization support in the ACME server main image.

### 4.2.9 Service Name Field

ACMEWQE$R_SERVICE_NAME, the service name field, is of data type ACMEWQEITM (see Section 4.1.4) and describes a UCS encoding of the name of the installed image that called the SYS$ACM[W] system service for this request.

This information is actually provided by the SYS$ACM[W] system service itself or by privileged clients, and can thus be trusted.

### 4.2.10 Requestor Profile Field

ACMEWQE$L_REQUESTOR_PROFILE, the requestor profile field, is the **persona ID** for a copy of the active persona when the ACM client process called the SYS$ACM[W] system service for this request. Although that active persona was in the context of the ACM client process, the ACME server main image has recreated this copy in the context of the ACME server process. Thus, your ACME agent can use this persona ID to call the persona query system service (SYS$PERSONA_QUERY) for details about the exact identity with which the ACM client process called the SYS$ACM[W] system service. It is important to use this method, rather than the SYS$GETJPI system service, to inquire about security attributes of the ACM client process because the current state of the ACM client process might be different from the state when it called the SYS$ACM[W] system service.

### 4.2.11 Requestor Mode Field

ACMEWQE$L_REQUESTOR_MODE, the requestor mode field, contains the access mode from which the ACM client process made the call to the SYS$ACM[W] system service. This may be different from the access mode of the active persona at the time of the call to the SYS$ACM[W] system service, as found in item ISS$_MODE of the persona identified by the persona ID stored in ACMEWQE$L_REQUESTOR_PROFILE.

### 4.2.12 Requestor Process ID Field

ACMEWQE$L_REQUESTOR_PID, the requestor process ID field, contains the process ID (PID) of the ACM client process. On Alpha, when kernel threads are in use, this still is the ID of the process (first kernel thread), not the individual kernel thread from which the request was queued. A subsequent kernel thread might no longer exist in the future while an image is still running in a process, so the PID provided in this field must be one that will persist.

### 4.2.13 Current ACME ID Field

ACMEWQE$L_CURRENT_ACME_ID, the current ACME ID field, is of data type ACMEID and contains the ACME ID for the ACME agent currently executing.

### 4.2.14 Target ACME ID Field

ACMEWQE$L_TARGET_ACME_ID, the target ACME ID field, is of data type ACMEID and for a **targeted request** it contains the ACME ID (if any) specified by the ACM client process to indicate the **target DOI** specified via one of the following items:

- ACME$_TARGET_DOI_ID
- ACME$_TARGET_DOI_NAME

### 4.2.15 Designated ACME ID Field

ACMEWQE$L_DESIGNATED_ACME_ID, the designated ACME ID field, is of data type ACMEID and contains the ACME ID of the ACME agent that called ACME callback routine ACME$CB_SET_DESIGNATED_DOI to indicate that it would take charge in processing this request.

### 4.2.16 Designated Credentials Field

ACMEWQE$L_DESIGNATED_CRED, the designated credentials field, is of data type unsigned longword and contains the persona extension ID for the ACME agent that has designated itself as DOI by calling ACME callback routine ACME$CB_SET_DESIGNATED_DOI.

### 4.2.17 Timeout Field

ACMEWQE$L_TIMEOUT, the timeout field, reflects the ACME$_TIMEOUT parameter provided by the client (or defaulted by the SYS$ACM[W] system service).

### 4.2.18 Factor Field

ACMEWQE$L_FACTOR, the factor field, contains the maximum number of simultaneous requests the ACM dispatcher will expect your ACME agent to have pending at one time.

### 4.2.19 Itemlist Field

ACMEWQE$PS_ITEMLIST, the item list field, contains a pointer to the first item list segment of ACME-independent items that the ACM client process has provided throughout the history of processing this request.

### 4.2.20 ACME Itemlist Field

ACMEWQE$PS_ACME_ITEMLIST, the ACME item list field, contains a pointer to the first item list segment of ACME-specific items that the ACM client process has provided for this ACME agent throughout the history of processing this request.

### 4.2.21 Function-Dependent Parameters Field

ACMEWQE$PS_FUNC_DEP_PARAMS, the function-dependent parameters field, contains a pointer. The data structure addressed by the pointer varies depending upon the nature of ACME callout routine, as shown in the following sections.

#### 4.2.21.1 The Agent Initialization WQE Extension

On a call to the ACME agent control callout routine ACME$CO_AGENT_STARTUP, the function-dependent parameters field contains a pointer to the ACMEWQEAIX data structure described in Section B.8.

ACMEWQEAIX$L_AGENT_NAME, the agent name field of that data structure, holds a pointer to a 32-bit string descriptor of the ACME agent name.

Currently it contains a null pointer.

#### 4.2.21.2 The Agent Startup WQE Extension

On a call to ACME agent control callout routine ACME$CO_AGENT_STARTUP, the function-dependent parameters field contains a pointer to the ACMEWQEAEX data structure described in Section B.7.

ACMEWQEAEX$L_CONCURRENT_REQUESTS, the concurrent requests field of that data structure, is of data type unsigned longword and contains the maximum number of simultaneous requests that the ACME agent might have to handle.

#### 4.2.21.3 The Authentication and Password Change WQE Extension

On a call to an ACME authentication and password callout routine, when field ACMEFC$V_FUNCTION of cell ACMEWQE$L_FUNCTION contains ACME$_FC_AUTHENTICATE_PRINCIPAL or ACME$_FC_CHANGE_PASSWORD, then the function-dependent parameters field contains a pointer to the ACMEWQEAX data structure described in Section B.10.

Each field is described in detail below.

| Field Name | Contains | Data Type |
|---|---|---|
| **ACMEWQEAX$L_NEW_PASSWORD_FLAGS** | New password flags | ACMEPWDFLG |

ACMEWQEAX$L_NEW_PASSWORD_FLAGS consists of the following flags to indicate the passwords the caller of the SYS$ACM[W] system service wishes to change on this Authenticate Principal or Change Ppassword request.

- ACMEPWDFLG$V_SYSTEM
- ACMEPWDFLG$V_PASSWORD_1
- ACMEPWDFLG$V_PASSWORD_2
- ACMEPWDFLG$V_SPECIFIED

  The caller of the SYS$ACM[W] system service does not specify flag ACMEPWDFLG$V_SPECIFIED. Instead, it is set by the ACME server main image to indicate that the ACME$_NEW_PASSWORD_FLAGS item was specified at all.

Aside from that last flag, the flags in this field are as supplied in the item ACME$_NEW_PASSWORD_FLAGS of the SYS$ACM[W] system service call.

| Field Name | Contains | Data Type |
|---|---|---|
| **ACMEWQEAX$L_LOGON_FLAGS** | Log on flags | ACMELGIFLG |

ACMEWQEAX$L_LOGON_FLAGS is comprised of flags collected from ACME agents by the ACME server main image for transmission back to the ACM client process that specified item code ACME$_LOGON_FLAGS. The following are the possible flags:

| Symbol | Meaning |
|---|---|
| ACMELGIFLG$V_NEW_MAIL_AT_LOGIN | The user had one or more new mail messages. |
| ACMELGIFLG$V_PASSWORD_CHANGED | The user changed the primary password. |
| ACMELGIFLG$V_PASSWORD_EXPIRED | The primary password expired without the user changing it. |
| ACMELGIFLG$V_PASSWORD_WARNING | The primary password will expire shortly. |
| ACMELGIFLG$V_PASSWORD2_CHANGED | The user changed the secondary password. |
| ACMELGIFLG$V_PASSWORD2_EXPIRED | The secondary password expired without the user changing it. |
| ACMELGIFLG$V_PASSWORD2_WARNING | The secondary password will expire shortly. |

ACME agents that handle primary (password_1) and secondary (password_2) passwords, or a mail system, should set these flags when appropriate by calling the ACME callback routine ACME$CB_SET_LOGON_FLAG with each code that is appropriate from the following list:

- ACMELGIFLG$K_NEW_MAIL_AT_LOGIN
- ACMELGIFLG$K_PASSWORD_CHANGED
- ACMELGIFLG$K_PASSWORD_EXPIRED
- ACMELGIFLG$K_PASSWORD_WARNING
- ACMELGIFLG$K_PASSWORD2_CHANGED
- ACMELGIFLG$K_PASSWORD2_EXPIRED
- ACMELGIFLG$K_PASSWORD2_WARNING

The ACME server main image sends the resulting ACMELGIFLG flags mask back to the ACM client process that specified the ACME$_LOGON_FLAGS output item code. This provides an indication of the processing that went on during authentication.

When LOGINOUT receives this flags mask, it makes it available to the created process through the SYS$GETJPI system service item code JPI$_LOGIN_FLAGS. Other callers of the SYS$ACM[W] system service make their own decision about what to do with the mask of flags.

| Field Name | Contains | Data Type |
|---|---|---|
| **ACMEWQEAX$R_LOGON_STATS_VMS** | OpenVMS log on statistics | ACMELIVMS[1] |

[1]See Section A.6.

ACMEWQEAX$R_LOGON_STATS_VMS contains statistics regarding the authentication from the VMS ACME. This information is part of the information returned to an ACM client program that specifies the ACME$_LOGON_INFORMATION output item code. The VMS ACME provides it by calling ACME authentication and password callout routine ACME$CB_SET_LOGON_STATS_VMS.

| Field Name | Contains | Data Type |
|---|---|---|
| **ACMEWQEAX$R_LOGON_STATS_DOI** | DOI log on statistics | ACMELIDOI[1] |

[1]See Section A.5.

ACMEWQEAX$R_LOGON_STATS_DOI contains statistics regarding the authentication from the ACME agent that called the ACME callback routine ACME$CB_SET_DESIGNATED_DOI. This information is part of the information returned to an ACM client program that specifies output item code ACME$_LOGON_INFORMATION. The ACME agent provides it by calling ACME authentication and password callout routine ACME$CB_SET_LOGON_STATS_VMS.

| Field Name | Contains | Data Type |
|---|---|---|
| **ACMEWQEAX$R_SYSTEM_PASSWORD** | System password | ACMEWQEITM[1] |

[1]See Section 4.1.4.

ACMEWQEAX$R_SYSTEM_PASSWORD describes a UCS encoding of the system password value received from the ACM client process. Typically only the VMS ACME handles the system password.

To set this field, the VMS ACME calls ACME callback routine ACME$CB_SET_WQE_PARAMETER with an ID parameter of ACMEWQE$K_SYSTEM_PASSWORD.

| Field Name | Contains | Data Type |
|---|---|---|
| **ACMEWQEAX$R_PRINCIPAL_NAME** | Principal name | ACMEWQEITM |

ACMEWQEAX$R_PRINCIPAL_NAME describes a UCS encoding of the principal name initially received by an ACME agent. Although the item code ACME$_PRINCIPAL_NAME_IN is generally considered to be the source of the principal name, other sources are possible. For example, in the case of the VMS ACME, the principal name might also come from the automatic login facility, or from a DECnet proxy login.

For honoring such alternate sources, your ACME agent should not edit the data it stores into field ACME$QEAX_PRINCIPAL_NAME. For settings made based on the item code ACME$_PRINCIPAL_NAME_IN, the only edit your ACME agent should make is to remove leading and trailing spaces.

To set this field, your ACME agent should call the ACME callback routine ACME$CB_SET_WQE_PARAMETER with an ID parameter of ACMEWQE$K_PRINCIPAL_NAME.

| Field Name | Contains | Data Type |
|---|---|---|
| **ACMEWQEAX$R_PRINCIPAL_NAME_OUT** | Principal name out | ACMEWQEITM |

ACMEWQEAX$R_PRINCIPAL_NAME_OUT describes a UCS encoding of the result of the **principal name mapping** ultimately performed for the authentication principal or change password operations.

This final selection is made during ACME authentication and password callout routine ACME$CO_ACCEPT_PRINCIPAL. The ACME agent that calls ACME callback routine ACME$CB_SET_DESIGNATED_DOI is the one to set this field.

In contrast to ACME$QEAX_PRINCIPAL_NAME, the ACME agent setting field ACMEW$EAX_PRINCIPAL_NAME_OUT can freely manipulate the contents.

To set this field, your ACME agent should call the ACME callback routine ACME$CB_SET_WQE_PARAMETER with an ID parameter of ACMEWQE$K_PRINCIPAL_NAME_OUT.

| Field Name | Contains | Data Type |
|---|---|---|
| **ACMEWQEAX$R_VMS_USERNAME** | OpenVMS username | ACMEWQEITM |

ACMEWQEAX$R_VMS_USERNAME describes a UCS encoding of the OpenVMS username to which the principal name out has been mapped. The ACME agent that calls ACME$CB_SET_DESIGNATED_DOI should set the ACME$QEAX_VMS_USERNAME field.

To set this field, your ACME agent should call the ACME callback routine ACME$CB_SET_WQE_PARAMETER with an ID parameter of ACMEWQE$K_VMS_USERNAME.

| Field Name | Contains | Data Type |
|---|---|---|
| **ACMEWQEAX$R_PASSWORD_1** | Password 1 | ACMEWQEITM |

ACMEWQEAX$R_PASSWORD_1 describes a UCS encoding of a ACME$_PASSWORD_1 value received from the ACM client process. To set this field, your ACME agent should call the ACME callback routine ACME$CB_SET_WQE_PARAMETER with an ID parameter of ACMEWQE$K_PASSWORD_1.

| Field Name | Contains | Data Type |
|---|---|---|
| **ACMEWQEAX$R_PASSWORD_2** | Password 2 | ACMEWQEITM |

ACMEWQEAX$R_PASSWORD_2 describes a UCS encoding of a ACME$_PASSWORD_2 value received from the ACM client process. To set this field, your ACME agent should call the ACME callback routine ACME$CB_SET_WQE_PARAMETER with an ID parameter of ACMEWQE$K_PASSWORD_2.

| Field Name | Contains | Data Type |
|---|---|---|
| **ACMEWQEAX$R_NEW_PASSWORD_1** | New password 1 | ACMEWQEITM |

ACMEWQEAX$R_NEW_PASSWORD_1 describes a UCS encoding of the new password 1 agreed upon by the various ACME agents. To set this field, your ACME agent should call the ACME callback routine ACME$CB_SET_WQE_PARAMETER with an ID parameter of ACMEWQE$K_NEW_PASSWORD_1.

| Field Name | Contains | Data Type |
|---|---|---|
| **ACMEWQEAX$R_NEW_PASSWORD_2** | New password 2 | ACMEWQEITM |

ACMEWQEAX$R_NEW_PASSWORD_2 describes a UCS encoding of the new password 2 agreed upon by the various ACME agents. To set this field, your

ACME agent should call the ACME callback routine ACME$CB_SET_WQE_ PARAMETER with an ID parameter of ACMEWQE$K_NEW_PASSWORD_2.

### 4.2.22 Revision Level Field

ACMEWQE$W_REVISION_LEVEL, the revision level field, is of type ACMEREVLVL (see Section A.9) and contains the overall ACME revision level, as defined by the ACME server main image.

### 4.2.23 Size Field

ACMEWQE$W_SIZE, the size field, is of data type unsigned word and contains the size of the work queue entry data structure.

### 4.2.24 FLINK Field

ACMEWQE$PS_FLINK is reserved for the ACME server main image.

### 4.2.25 BLINK Field

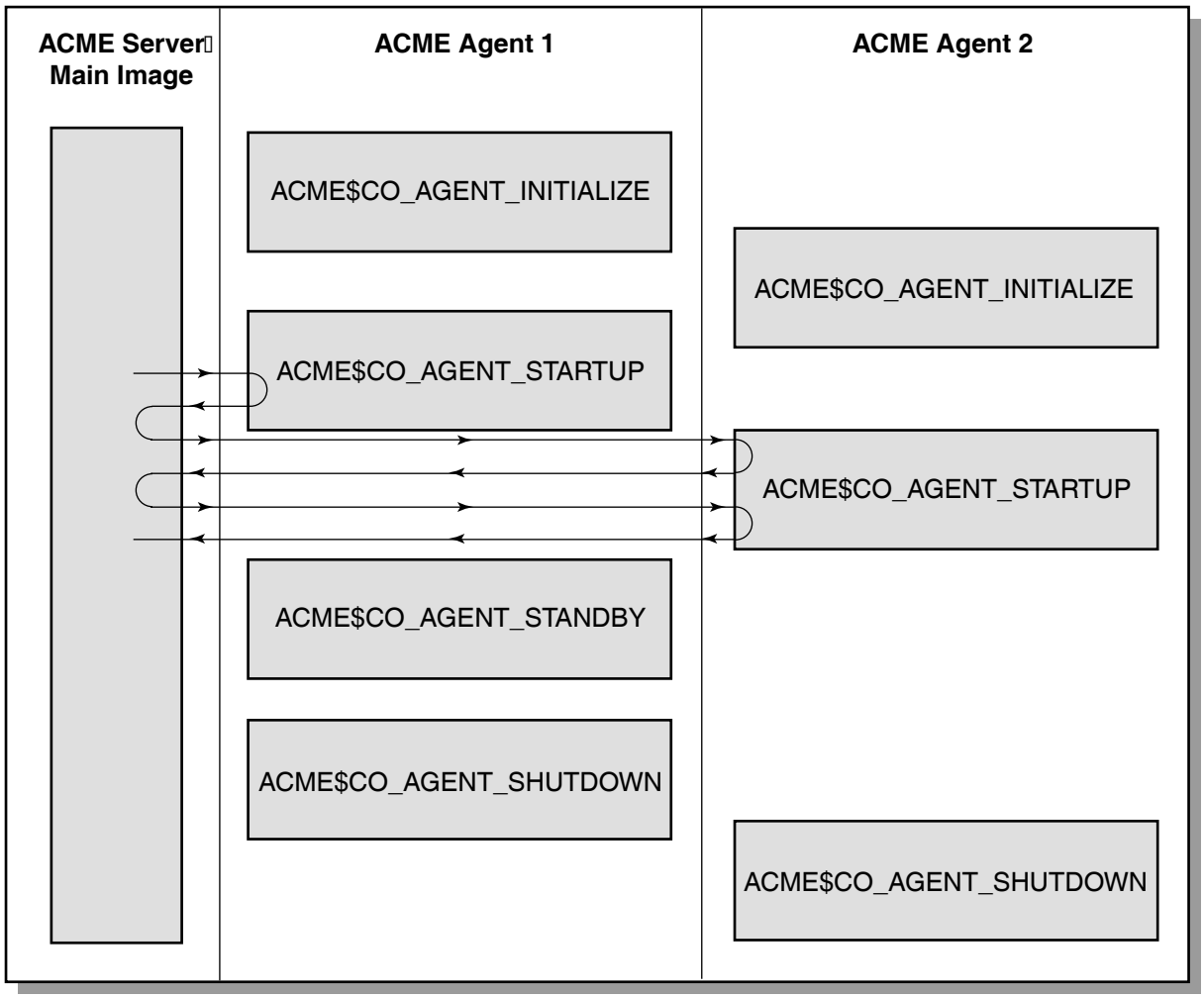
ACMEWQE$PS_BLINK is reserved for the ACME server main image.

# 5

# ACME Agent Control Callout Routines

This chapter describes the possible ACME agent control callout routines to support coordinated configuration and startup of ACME agents. Different ACME agents may each supply their own version of the same ACME agent control callout routine and each will be called in order, as shown in Figure 5–1.

**Figure 5–1 ACME Agent Control Callout Routine Control Flow**




VM-0783A-AI

In any ACME callout routine, an ACME agent may perform processing for its own purposes, so long as it does not violate any of the programming rules in Chapter 2. An ACME agent, however, must also perform certain mandatory

processing specific to the ACME callout routine it provides (see individual routine descriptions for more information).

No request context or item list arguments are provided for agent control callout routines since these are not request processing callouts (no call to the SYS$ACM [W] system service was made to trigger the invocation).

All ACME agents must support ACME callout routine ACME$CO_AGENT_ INITIALIZE. The other routines are optional.

The following section presents the ACME agent control callout routines in alphabetical order. The following table lists them in their normal processing order.

**Table 5–1 Processing Order for Agent Control Callout Routines**

| | |
|---|---|
| 1 | ACME$CO_AGENT_INITIALIZE |
| 2 | ACME$CO_AGENT_STARTUP |
| 3 | ACME$CO_AGENT_SHUTDOWN |
| 4 | ACME$CO_AGENT_STANDBY |

## 5.1 Arguments

The arguments are the same for each callout routine, with one exception. ACME$CO_AGENT_ STARTUP takes an additional argument: *factor*. All arguments for this set of callout routines are described in Table 5–2.

**Table 5–2 Arguments for Agent Control Callout Routines**

| Argument | Description |
|---|---|
| kcb_vector | Address of an array called the KCB (Kernel Callback) vector. To interact with the ACME server main image and other ACME agents, your ACME agent must invoke various ACME callback routines. The procedure values of those callback routines are stored in the KCB vector. |
| acme_context | Address of a common quadword provided on all invocations of your ACME agent, for whatever purpose you deem fit. |
| wqe | Address of the work queue entry for this request. Has the structure described in Section B.16. Your ACME agent can read information directly out of the work queue entry, but should only change the contents of the work queue entry through the designated ACME callback routines. See Section 4.2. |
| factor | ACME$CO_AGENT_STARTUP only. Address of a quadword containing a 32-bit value representing the number of concurrent threads in this process. |

## 5.2 Return Values for Agent Control Callout Routines

The return values are the same for each callout routine and are described in Table 5–3.

**Table 5–3 Return Values for Agent Control Callout Routines**

| Value | Description |
|---|---|
| ACME$_CONTINUE | Call next ACME agent |
| Other | Complete with success or failure |

# ACME$CO_AGENT_INITIALIZE

This routine must make exactly one call to the ACME callback routine ACME$CB_REPORT_ATTRIBUTES to indicate the required resources for the operation of this ACME agent, both on a static level and on a per-request level.

## Format

ACME$CO_AGENT_INITIALIZE kcb_vector, acme_context, wqe

## Description

This routine is the preferred place to check data structure versions, as described in Section 4.1.1.

# ACME$CO_AGENT_SHUTDOWN

This routine is called in response to the SET SERVER ACME/DISABLE command. Your ACME agent should use this opportunity to clean up any loose ends such as open channels, allocated memory, and others.

## Format

ACME$CO_AGENT_SHUTDOWN kcb_vector, acme_context, wqe

# ACME$CO_AGENT_STANDBY

The ACME server main image may call this routine when there has been no recent SYS$ACM[W] system service activity. Your ACME agent can take advantage of this hint to flush caches, close files and otherwise clean things up.

## Format

ACME$CO_AGENT_STANDBY kcb_vector, acme_context, wqe

## Description

No time-consuming activities should be undertaken, as this processing blocks the handling of new requests that ACM client processes may present.

# ACME$CO_AGENT_STARTUP

This routine is called in response to the SET SERVER ACME/ENABLE command. Your ACME agent can do the preparatory work you want, such as opening network connections.

## Format

ACME$CO_AGENT_STARTUP kcb_vector, acme_context, wqe, factor

## Description

Your ACME agent can also check ACMEWQE$L_FACTOR and use it to initialize or allocate data structures.

The acme_context argument can be used to store the address of memory allocated with ACME callback routine ACME$CB_ALLOCATE_ACME_VM.

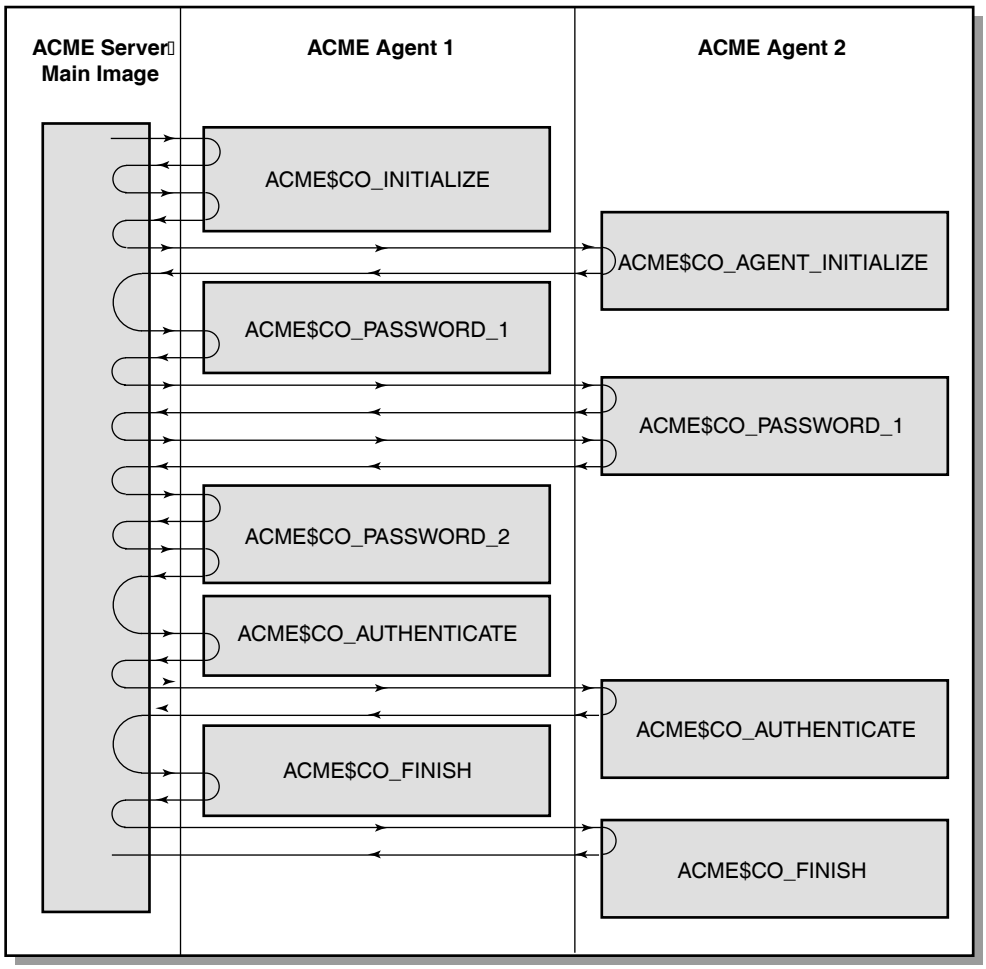
# 6

# ACME Authentication and Password Callout Routines

This chapter describes the possible ACME request processing callout routines to support processing Authenticate Principal or Change Password requests.

Different ACME agents may each supply their own version of the same ACME agent control callout routine and each will be called in order, as shown in Figure 6–1.

**Figure 6–1 ACME Authentication and Password Callout Routine Control Flow**



ACME authentication and password callout routines differ from the ACME callout routines described in Chapter 5 and Chapter 7 in the following ways:

- A series of different ACME callout routines are called in succession to process each SYS$ACM[W] system service request.
- For each ACME callout routine, each ACME agent that provides such an entry point will be called in turn, rather than relying on a single ACME agent.
- For dialog mode requests, any ACME agent may send output text or inquiries back to the client program.

Rather than acting alone, as they do for ACME agent control callout routines and ACME event and query callout routines, the ACME authentication and password callout routines act in concert to provide a unified authentication service.

In any ACME callout routine, an ACME agent may perform processing for its own purposes, as long as it does not violate any of the programming rules in Chapter 2. An ACME agent, however, must also perform certain mandatory processing specific to the ACME callout routine it provides, as outlined throughout the remainder of this chapter.

All ACME agents that support any ACME authentication and password callout routine must support ACME callout routine ACME$CO_FINISH. The other routines are optional.

The following sections present the ACME authentication and password callout routines in alphabetical order. The following table lists them in their normal processing order.

**Table 6–1 Processing Order for Authentication and Password Callout Routines**

| | |
|---|---|
| 1 | ACME$CO_INITIALIZE |
| 2 | ACME$CO_SYSTEM_PASSWORD |
| 3 | ACME$CO_ANNOUNCE |
| 4 | ACME$CO_AUTOLOGON |
| 5 | ACME$CO_PRINCIPAL_NAME |
| 6 | ACME$CO_ACCEPT_PRINCIPAL |
| 7 | ACME$CO_MAP_PRINCIPAL |
| 8 | ACME$CO_VALIDATE_MAPPING |
| 9 | ACME$CO_ANCILLARY_MECH_1 |
| 10 | ACME$CO_PASSWORD_1 |
| 11 | ACME$CO_ANCILLARY_MECH_2 |
| 12 | ACME$CO_PASSWORD_2 |
| 13 | ACME$CO_ANCILLARY_MECH_3 |
| 14 | ACME$CO_AUTHENTICATE |
| 15 | ACME$CO_MESSAGES |
| 16 | ACME$CO_AUTHORIZE |
| 17 | ACME$CO_NOTICES |

(continued on next page)

**Table 6–1 (Cont.) Processing Order for Authentication and Password Callout Routines**

| | |
|---|---|
| 18 | ACME$CO_LOGON_INFORMATION |
| 19 | ACME$CO_NEW_PASSWORD_1 |
| 20 | ACME$CO_QUALIFY_PASSWORD_1 |
| 21 | ACME$CO_NEW_PASSWORD_2 |
| 22 | ACME$CO_QUALIFY_PASSWORD_2 |
| 23 | ACME$CO_ACCEPT_PASSWORDS |
| 24 | ACME$CO_SET_PASSWORDS |
| 25 | ACME$CO_CREDENTIALS |
| 26 | ACME$CO_FINISH |

## 6.1 Arguments

The arguments are the same for each callout routine and are described in Table 6–2.

**Table 6–2 Arguments for ACME Authentication and Password Callout Routines**

| Argument | Description |
|---|---|
| kcb_vector | Address of an array called the KCB (Kernel Callback) vector. To interact with the ACME server main image and other ACME agents, your ACME agent must invoke various ACME callback routines. The procedure values of those callback routines are stored in the KCB vector. |
| acme_context | Address of a common quadword provided on all invocations of your ACME agent, for whatever purpose you deem fit. |
| wqe | Address of the work queue entry for this request. Has the structure described in Section B.16. Your ACME agent can read information directly out of the work queue entry, but should only change the contents of the work queue entry through the designated ACME callback routines. See Section 4.2. |
| request_context | Address of a single quadword provided on invocatins of your ACME agent in support of a particular request for whatever purpose you deem fit. |

(continued on next page)

**Table 6–2 (Cont.) Arguments for ACME Authentication and Password Callout Routines**

| Argument | Description |
|---|---|
| common_item_list | Consists of one or more item list segments chained together. All item list segments follow the 32-bit addressing item list format, to facilitate operation between ACME agents written in various programming languages (some of which might not support 64-bit addressing).<br>The chaining between item list segments does not reflect any initial chaining provided by the caller of the SYS$ACM[W] system service, but instead is the result of segmentation in the transfer of the item list data from the ACM client process context to the ACME server process context.<br>In the case of the ACME callout routiens ACME$CO_INITIALIZE, ACME$CO_ EVENT, and ACME$CO_QUERY the items provided include those from the initial call to the SYS$ACM[W] system service. For subsequent ACME authentication and password callout routines other than ACME$CO_INITIALIZE, the items include only those specifically requested in dialogue interaction from that ACME callout routine.<br>**Common item list** items are those provided during this ACME callout routine with common item codes (ACME$_codename). To inspect items provided to previous ACME authentication and password callout routines for this request, use the chain of item list segments pointed to by the work queue entry cell ACMEWQE$PS_ ITEMLIST. |
| acme_item_list | Consists of one or more item list segments chained together. All item list segments follow the 32-bit addressing item list format, to facilitate operation between ACME agents written in various programming languages (some of which might not support 64-bit addressing).<br>The chaining between item list segments does not reflect any initial chaining provided by the caller of the SYS$ACM[W] system service, but instead is the result of segmentation in the transfer of the item list data from the ACM client process context to the ACME server process context.<br>In the case of the ACME callout routiens ACME$CO_INITIALIZE, ACME$CO_ EVENT, and ACME$CO_QUERY the items provided include those from the initial call to the SYS$ACM[W] system service. For subsequent ACME authentication and password callout routines other than ACME$CO_INITIALIZE, the items include only those specifically requested in dialogue interaction from that ACME callout routine.<br>ACME-specific item list items are those provided during this ACME callout routine with ACME-specific item codes (not ACME$_codename) targeted at this ACME agent. Your ACME agent will not see any ACME-specific item codes targeted at other ACME agents. To inspect items provided to previous ACME authentication and password callout routines for this request, use the chain of item list segments pointed to by work queue entry cell ACMEWQE$PS_ACME_ITEMLIST. |

## 6.2 Return Values

The return values are the same for each callout routine and are described in Table 6–3. For information on dispatching, see Section 2.3.

**Table 6–3 Return Values for Authentication and Password Callout Routines**

| Value | Description |
|---|---|
| ACME$_CONTINUE | Call next ACME agent |
| ACME$_WAITAST | Return to this routine after completing the next AST for this agent |

(continued on next page)

**Table 6–3 (Cont.) Return Values for Authentication and Password Callout Routines**

| Value | Description |
|---|---|
| ACME$_WAITRESOURCE | Return to this routine when the next ACME-specific resource (for this ACME agent) becomes available of the type code for which the most recent ACME$RESOURCENOTAVAIL was returned |
| ACME$_ PERFORMDIALOGUE | Send all dialogue data that has been queued by a call to ACME callback routine ACME$CB_QUEUE_DIALOGUE back to the ACM client process and call this ACME callout routine back when there is either a response (for input) or an acknowledgement (for output) available from the client |
| Other | Complete with success or failure |

# ACME$CO_ACCEPT_PASSWORDS

In this routine your ACME agent should prepare to commit any agreed-upon new passwords.

**Note**

Because many ACME agents will engage in the actions of this routine, no ACME agent should call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE.

## Format

| | |
|---|---|
| ACME$CO_ACCEPT_PASSWORDS | kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list |

## Description

**Processing steps:**

1. If the ACMEWQEFLG$V_SKIP_NEW_PASSWORD flag is set, return ACME$_CONTINUE.
2. If operating under model 2, secondary DOI agents may prompt for new agent-specific passwords.
3. If operating under model 1 or 2, all DOI agents prepare to commit password changes. If unable to do so, return a failure status describing the nature of the failure.

# ACME$CO_ACCEPT_PRINCIPAL

In this routine the first ACME agent to find that the ACMEWQEAX$R_PRINCIPAL_NAME field of the WQE extension for authentication contains a principal name it can support must rewrite the ACMEWQEAX$R_PRINCIPAL_NAME_OUT field of the WQE extension for authentication with a canonical representation of the principal name. It must declare itself the designated DOI by calling ACME$CB_SET_DESIGNATED_DOI.

## Format

ACME$CO_ACCEPT_PRINCIPAL kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list

## Description

Various ACME agents have differing methods for initially considering whether they might recognize some particular principal-name In, such as particular characters being included or excluded from the name. For example, the NT ACME will try to recognize a principal-name In that contains a slash (/) or a commercial at sign (@), whereas the VMS ACME will not try to recognize a principal-name In that contains anything other than alphanumerics, the underscore and the dollar sign. After such an initial test, however, any ACME agent that has determined the principal-name In might be one that they recognize must make a thorough test to determine whether it in fact does recognize it, typically by consulting an ACME-specific table or a networked system that contains such a table.

If an ACME agent does not recognize a particular Principal Name In, it should return ACME$_CONTINUE from its ACME$CO_ACCEPT_PRINCIPAL ACME authentication and password callout routine to let other ACME agents have a chance to handle that principal name. If an ACME agent does recognize the Principal Name In stored in the PRINCIPAL_NAME field of the WQE authentication extension, it must call ACME$CB_SET_WQE_PARAMETER to specify the edited Principal Name Out that should be stored in the PRINCIPAL_NAME_OUT field of the WQE authentication extension. The editing to create the Principal Name Out could do some of the following:

- Remove leading or trailing spaces.
- Adjust cases of alphabetic characters.
- Make any other transformation deemed suitable.
- Make no changes at all.

**Processing Steps:**

The recognizing ACME agent declares itself the designated DOI by calling ACME$CB_SET_DESIGNATED_DOI.

1. Test flag ACMEWQEFLG$V_PHASE_DONE. If it has been set (presumably by the corresponding ACME authentication and password callout routine for some prior ACME agent) skip the following steps.

2. Using methods private to the ACME agent, test the string in ACMEWQEAX$R_PRINCIPAL_NAME to see if the syntax can be handled by this ACME agent, and if not, skip the following steps since this ACME agent cannot handle this principal name.
3. Using methods private to the ACME agent, look up the string in ACMEWQEAX$R_PRINCIPAL_NAME to see if it matches a username for the DOI supported by this ACME agent. If not, skip the following steps since this ACME agent cannot handle this principal name.
4. Call the ACME callback routine ACME$CB_SET_WQE_PARAMETER with an ID parameter of ACMEWQE$K_PRINCIPAL_NAME_OUT and a DATA parameter describing the edited principal name.
5. Call the ACME callback routine ACME$CB_SET_DESIGNATED_DOI to indicate for future name authentication and password callout routines that this ACME agent implemented the designated DOI.
6. Call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE to prevent other ACME agents from engaging in the purpose-based actions of this routine.

# ACME$CO_ANCILLARY_MECH_1

In this routine your ACME agent can gather client input (prompting if necessary and possible) for any ACME-specific authentication information to be gathered before PASSWORD_1.

**Note**

Because many ACME agents will engage in the actions of this routine, no ACME agent should call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE.

## Format

| | |
|---|---|
| ACME$CO_ANCILLARY_MECH_1 | kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list |

## Description

**Processing Steps:**

1. Test flag ACMEWQEFLG$V_PREAUTHENTICATED and if it has been set during some prior ACME authentication and password callout routine, skip the following steps.
2. Perform the authentication processing specific to this ACME agent, skipping the following steps if successful.
3. Indicate the real cause of failure for return to the caller ACMSB field ACMESB$L_SECONDARY_STATUS by a call to ACME callback routine ACME$CB_SET_2ND_STATUS providing one of the following:
   - A failure code from Section 8.4
   - A failure code specific to your ACME agent
4. Return the sanitized error code ACME$_AUTHFAILURE to the ACME server process so it can return it to the caller ACMSB field ACMESB$L_STATUS.

# ACME$CO_ANCILLARY_MECH_2

In this routine your ACME agent can gather client input (prompting if necessary and possible) for any ACME-specific authentication information to be gathered after PASSWORD_1 and before PASSWORD_2.

**Note**

Because many ACME agents will engage in the actions of this routine, no ACME agent should call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE.

## Format

| | |
|---|---|
| ACME$CO_ANCILLARY_MECH_2 | kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list |

## Description

**Processing Steps:**

1. Test flag ACMEWQEFLG$V_PREAUTHENTICATED and if it has been set during some prior ACME authentication and password callout routine, skip the following steps.
2. Perform the authentication processing specific to this ACME agent, skipping the following steps if successful.
3. Indicate the real cause of failure for return to the caller ACMSB field ACMESB$L_SECONDARY_STATUS by a call to ACME callback routine ACME$CB_SET_2ND_STATUS providing one of the following:
   - A failure code from Section 8.4
   - A failure code specific to your ACME agent
4. Return the sanitized error code ACME$_AUTHFAILURE to the ACME server process so it can return it to the caller ACMSB field ACMESB$L_STATUS.

# ACME$CO_ANCILLARY_MECH_3

In this routine your ACME agent can gather client input (prompting if necessary and possible) for any ACME-specific authentication information to be gathered after PASSWORD_2.

**Note**

Because many ACME agents will engage in the actions of this routine, no ACME agent should call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE.

## Format

| | |
|---|---|
| ACME$CO_ANCILLARY_MECH_3 | kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list |

## Description

**Processing Steps:**

1. Test flag ACMEWQEFLG$V_PREAUTHENTICATED and if it has been set during some prior ACME authentication and password callout routine, skip the following steps.
2. Perform the authentication processing specific to this ACME agent, skipping the following steps, if successful.
3. Indicate the real cause of failure for return to the caller ACMSB field ACMESB$L_SECONDARY_STATUS by a call to ACME callback routine ACME$CB_SET_2ND_STATUS providing one of the following:
   - A failure code from Section 8.4
   - A failure code specific to your ACME agent
4. Return the sanitized error code ACME$_AUTHFAILURE to the ACME server process so it can return it to the caller ACMSB field ACMESB$L_STATUS.

# ACME$CO_ANNOUNCE

In this routine each ACME agent has an opportunity to provide text to the ACM client process that might be shown to a user before authentication. The ACM dispatcher will not invoke this ACME callout routine for the ACME$_FC_CHANGE_PASSWORD function code.

**Note**

Because many ACME agents will engage in the actions of this routine, no ACME agent should call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE.

## Format

ACME$CO_ANNOUNCE kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list

## Description

**Processing Steps:**

If you decide to provide such text, your ACME agent should call the ACME callback routine ACME$CB_QUEUE_DIALOGUE successively to provide the output and then return code ACME$_PERFORMDIALOGUE to have it transmitted.

Since the local system manager controls the order in which ACME agents are configured, your ACME agent should make no assumption that the text it provides will be displayed in any particular order with respect to that provided by other ACME agents, such as the traditional SYS$ANNOUNCE display from the VMS ACME.

In many environments, system managers are sensitive regarding the volume of information presented to users logging in, so it is best if your ACME agent supports a mechanism for the system manager to control the amount of, or entirely disable, information that your ACME agent provides.

The ACM client process is not required to display the information you provide at all, much less with any particular prominence on its screen, printer, or tickertape.

# ACME$CO_AUTHENTICATE

In this routine, DOI agents evaluate the correctness of the saved authentication inputs in Password 1 and Password 2. Any other ACME agents evaluate the correctness of authentication inputs they collected.

**Note**

Because many ACME agents will engage in the actions of this routine, no ACME agent should call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE.

## Format

ACME$CO_AUTHENTICATE kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list

## Description

**Processing Steps:**

1. Test flag ACMEWQEFLG$V_PREAUTHENTICATED and if it has been set during some prior ACME authentication and password callout routine, skip the following steps.
2. Perform the authentication processing specific to this ACME agent. This can include evaluating unprocessed information specific to your ACME agent gathered during the following ACME authentication and password callout routines:
   - ACME$CO_ANCILLARY_MECH_1
   - ACME$CO_ANCILLARY_MECH_2
   - ACME$CO_ANCILLARY_MECH_3
3. Test any value supplied for item ACME$_AUTH_MECHANISM. If it is some non-zero value other than ACMEMECH$K_PASSWORD, skip the following steps since this authentication is not password based, so the actions of this ACME authentication and password callout routine are irrelevant.
4. Perform the authentication processing of primary and secondary passwords, skipping the following steps, if successful.
5. Indicate the real cause of failure for return to the caller ACMSB field ACMESB$L_SECONDARY_STATUS by a call to ACME callback routine ACME$CB_SET_2ND_STATUS providing one of the following:
   - A failure code from Section 8.4
   - A failure code specific to your ACME agent
6. Return the sanitized error code ACME$_AUTHFAILURE to the ACME server process so it can return it to the caller ACMSB field ACMESB$L_STATUS.

# ACME$CO_AUTHORIZE

In this routine each ACME agent evaluates the suitability of the authenticated client for access, such as time-of-day restrictions or break-in evasion. The ACM dispatcher will not invoke this ACME callout routine for the ACME$_FC_CHANGE_PASSWORD function code.

**Note**

Because many ACME agents will engage in the actions of this routine, no ACME agent should call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE.

## Format

ACME$CO_AUTHORIZE kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list

## Description

**Processing Steps:**

None.

# ACME$CO_AUTOLOGON

In this routine ACME agents perform any steps to specify the principal name other than through explicit client input. The VMS ACME, for example, uses this routine to check for automatic login facility (ALF) entries and to handle DECnet proxy login processing.

## Format

ACME$CO_AUTOLOGON kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list

## Description

**Processing Steps:**

1. Test flag ACMEWQEFLG$V_PHASE_DONE. If it has been set (presumably by the corresponding ACME authentication and password callout routine for some prior ACME agent) skip the following steps.
2. Using mechanisms specific to your ACME agent test to see whether you have a principal name to provide for this request. If not, return ACME$_CONTINUE.
3. Call the ACME callback routine ACME$CB_SET_WQE_PARAMETER with an ID parameter of ACMEWQE$K_PRINCIPAL_NAME and a DATA parameter describing the desired principal name.
4. Call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE to prevent other ACME agents from engaging in the purpose-based actions of this ACME authentication and password callout routine.

# ACME$CO_CREDENTIALS

In this routine your ACME agent should provide any DOI-specific credentials for the caller of SYS$ACM. The ACM dispatcher will not invoke this ACME callout routine for the ACME$_CHANGE_PASSWORD function code.

**Note**

Because many ACME agents will engage in the actions of this routine, no ACME agent should call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE.

## Format

ACME$CO_CREDENTIALS kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list

## Description

If the ACME$M_ACQUIRE_CREDENTIALS flag of the work queue entry is not set, the client has not requested credentials. This flag provides a hint to your ACME agent possibly avoiding delays due to lengthy procedures to create unneeded credentials.

# ACME$CO_FINISH

In this routine your ACME agent should clean up after successful or unsuccessful processing, including return of item codes and any ACME-specific logging.

**Note**

Because many ACME agents will engage in the actions of this routine, no ACME agent should call the ACME callback routine ACME$CB_SET_WQE_FLAGE with FLAG argument ACMEWQEFLG$K_PHASE_DONE.

## Format

ACME$CO_FINISH kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list

## Description

Your ACME agent should not perform any final recording of results until this ACME authentication and password callout routine since before now any ACME agent has the opportunity to fail a hitherto successful request, changing the results.

Since other ACME agents may have concluded their ACME$CO_FINISH processing before your ACME agent is called for ACME$CO_FINISH, it is undesirable to return a failure code from ACME$CO_FINISH for a request that has previously been successful. In particular, the VMS ACME will write an audit record to indicate the success or failure of a request, and that would be wrong if your ACME agent failed the request in the ACME$CO_FINISH authentication and password callout routine. There may be extraordinary cases where such a last minute failure might be required, but it should be avoided if possible.

**Processing Steps:**
ACMEWQEFLG$V_ABORT_REQUEST is set only when control has been transferred to ACME$CO_FINISH during a wait for client dialogue response (some ACME agent returned ACME$_PERFORMDIALOGUE).

Deallocation of memory that was in use and waiting for any outstanding ASTs that were queued for that request is the responsibility of the ACME$CO_FINISH ACME authentication and password callout routine for each ACME.

The following is a simple but correct method of handling this with minimal bookkeeping:

1. Never have I/O outstanding while waiting for dialogue response.
2. Never have I/O outstanding between ACME authentication and password callout routines (since other ACME agents running then might be waiting for the dialogue response that never comes, and instead sets ACMEWQEFLG$V_ABORT_REQUEST with a control transfer to ACME$CO_FINISH).
3. Never have other ASTs pending in those situations either.
4. Ensure that memory bookkeeping persists for all allocations that span dialogue responses or ACME authentication and password callout routines.

Not all ACME agents are able to handle things in such a dogmatic fashion. In those cases precise state information regarding I/Os, other ASTs and all memory allocations will be required.

# ACME$CO_INITIALIZE

In this routine all but the simplest ACME agents should allocate a request context data structure of their own design and store its address at the location pointed to by the request context parameter.

**Note**

Because many ACME agents will engage in the actions of this routine, no ACME agent should call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE.

## Format

ACME$CO_INITIALIZE kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list

## Description

Your ACME agent can also take this opportunity to read the initial item list data provided by the caller of the SYS$ACM[W] system service during this routine, while it is still provided in the item list argument. If your ACME agent waits until later ACME authentication and password callout routines, it would have to process the longer chains pointed to by the work queue entry fields ACMEWQE$PS_WQE_ITEMLIST and ACMEWQE$PS_WQE_ACME_ITEMLIST.

# ACME$CO_LOGON_INFORMATION

In this routine each ACME agent has an opportunity to provide short text to the ACM client process that might be shown to a user as part of the final output after a simple authentication. The intention is that this information be brief, so that it will certainly fit on the last screen of any video device being used, from a 24-inch graphics display to the smallest Personal Digital Assistant (PDA). Only crucial information, such as that included in data type ACMELIDOI, should be included.

The ACM dispatcher will not invoke this ACME callout routine for the ACME$_FC_CHANGE_PASSWORD function code.

**Note**

Because many ACME agents will engage in the actions of this routine, no ACME agent should call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE.

## Format

| | |
|---|---|
| ACME$CO_LOGON_INFORMATION | kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list |

## Description

If the status subfield of the ACMEWQE$R_STATUS field indicates a failure status, return ACME$_CONTINUE without sending any messages to the client process.

If you decide to provide such information, your ACME agent should call the ACME callback routine ACME$CB_QUEUE_DIALOGUE successively to provide the output and then return code ACME$_PERFORMDIALOGUE to have it transmitted.

Since the order in which ACME agents are configured is under the control of the local system manager, your ACME agent should make no assumption that the text it provides will be displayed in any particular order with respect to other ACME agents, such as the traditional security report information from the VMS ACME.

In many environments, system managers are sensitive regarding the volume of information presented to users logging in, so it is best if your ACME agent supports a mechanism for the system manager to control the amount of, or entirely disable, information that your ACME agent provides.

The ACM client process is not required to display the information you provide at all, much less with any particular prominence on its screen, printer, or tickertape.

# ACME$CO_MAP_PRINCIPAL

In this routine the designated DOI agent specifies the VMS username to use, if this authentication succeeds.

## Format

| | |
|---|---|
| ACME$CO_MAP_PRINCIPAL | kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list |

## Description

Call the ACME callback routine ACME$CB_SET_WQE_PARAMETER with an ID parameter of ACMEWQE$K_VMS_USERNAME and a DATA parameter describing the VMS username.

# ACME$CO_MESSAGES

In this routine each ACME agent can provide text to the ACM client process that might be shown to a user after authentication but before authorization tests. The ACM dispatcher will not invoke this ACME callout routine for the ACME$_FC_CHANGE_PASSWORD function code.

**Note**

Because many ACME agents will engage in the actions of this routine, no ACME agent should call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE.

## Format

ACME$CO_MESSAGES kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list

## Description

If you decide to provide such information, your ACME agent should call the ACME callback routine ACME$CB_QUEUE_DIALOGUE successively to provide the output and then return code ACME$_PERFORMDIALOGUE to have it transmitted.

Since the order in which ACME agents are configured is under the control of the local system manager, your ACME agent should make no assumption that the text it provides will be displayed in any particular order with respect to other ACME agents.

This output opportunity does not correspond to any traditional display from the VMS ACME.

In many environments, system managers are sensitive regarding the volume of information presented to users logging in, so it is best if your ACME agent supports a mechanism for the system manager to control the amount of, or entirely disable, information that your ACME agent provides.

The ACM client process is not required to display the information you provide at all, much less with any particular prominence on its screen, printer, or tickertape.

# ACME$CO_NEW_PASSWORD_1

In this routine the designated DOI agent can solicit and evaluate a new primary password from the client if explicitly requested (or in the case of authentication, if the previous password has expired).

## Format

ACME$CO_NEW_PASSWORD_1 kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list

## Description

**Processing Steps:**

1. If the ACMEWQEFLG$V_SKIP_NEW_PASSWORD flag is set, return ACME$_CONTINUE.
2. If this is not the designated DOI agent for this request, skip the following steps.
3. If item code ACME$_NEW_PASSWORD_1 was included in the Common item list that the ACM client program supplied on the initial call to the SYS$ACM[W] system service, use that value (with leading and trailing spaces and tabs removed) for the desired new primary password.
4. Otherwise ask the ACM client process for the new primary password, as follows. Since security needs require that the password be typed without echoing the characters, the user cannot see the results of the typing, so your ACME agent must engage in either **deferred confirmation** or **immediate confirmation**. To ask the ACM client process for the new primary password, use the following steps:
   a. Call the ACME callback routine ACME$CB_QUEUE_DIALOGUE specifying the following:
      - An ITEM_CODE parameter of ACME$_NEW_PASSWORD_1
      - A DATA_1 parameter prompt string
      - A DATA_2 parameter confirmation string (if using immediate confirmation)
      - A FLAGS parameter with ACMEDLOGFLG$V_INPUT and ACMEDLOGFLG$V_NOECHO set

   b. Return code ACME$_PERFORMDIALOGUE.

   c. When called back, retrieve the new primary password from parameter ITEM_LIST, using the last item on the chain of item list segments that has the item code ACME$_NEW_PASSWORD_1.

   d. To support deferred confirmation:
      - Compare the proposed password against password quality rules specific to your ACME agent. If it fails, do the following:
        – Call the ACME callback routine ACME$CB_QUEUE_DIALOGUE to provide a one-line error message describing the password quality problem.

- Return code ACME$_PERFORMDIALOGUE to have your error message transmitted to the ACM client process.
- When your ACME authentication and password callout routine is called again (indicating your error message was transmitted to the ACM client process), return code ACME$_RETRYPWD to cause the ACM dispatcher to return control to ACME$CO_NEW_PASSWORD_1.

- Call the ACME callback routine ACME$CB_QUEUE_DIALOGUE and specify the following:
  - An ITEM_CODE parameter of ACME$_NEW_PASSWORD_1
  - A DATA_1 argument confirmation string
  - A FLAGS argument with ACMEDLOGFLG$V_INPUT and ACMEDLOGFLG$V_NOECHO set
- Return code ACME$_PERFORMDIALOGUE.
- When called back, retrieve the confirmation primary password from parameter ITEM_LIST, using the last item on the chain of item list segments that has the item code ACME$_NEW_PASSWORD_1.
- Compare the original proposed password to the confirmation proposed password and if they do not match, issue an error message and restart step 4 to ask for the proposed new password.

When your ACME agent has received a confirmed string from the ACM client program, use that value (with leading and trailing spaces and tabs removed) for the desired new primary password.

5. Call the ACME callback routine ACME$CB_SET_WQE_PARAMETER with an ID parameter of ACMEWQE$K_NEW_PASSWORD_1 and a DATA parameter describing the new primary password.

# ACME$CO_NEW_PASSWORD_2

In this routine the designated DOI agent can solicit and evaluate a new secondary password from the client if explicitly requested (or in the case of authentication, if the previous password has expired).

## Format

ACME$CO_NEW_PASSWORD_2 kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list

## Description

**Processing Steps:**

1. If the ACMEWQEFLG$V_SKIP_NEW_PASSWORD flag is set, return ACME$_CONTINUE.
2. If this is not the designated DOI agent for this request, skip the following steps.
3. If item code ACME$_NEW_PASSWORD_2 was included in the Common item list that the ACM client program supplied on the initial call to the SYS$ACM[W] system service, use that value (with leading and trailing spaces and tabs removed) for the desired new secondary password.
4. Otherwise ask the ACM client process for the new secondary password, as follows. Since security needs require that the password be typed without echoing the characters, the user cannot see the results of the typing, so your ACME agent must engage in either deferred confirmation or immediate confirmation.
   a. Call the ACME callback routine ACME$CB_QUEUE_DIALOGUE and specify the following:
      - An ITEM_CODE argument of ACME$_NEW_PASSWORD_2
      - A DATA_1 argument prompt string
      - A DATA_2 argument confirmation string (if using immediate confirmation)
      - A FLAGS argument with ACMEDLOGFLG$V_INPUT and ACMEDLOGFLG$V_NOECHO set

   b. Return code ACME$_PERFORMDIALOGUE.

   c. When called back, retrieve the new secondary password from parameter ITEM_LIST, using the last item on the chain of item list segments that has the item code ACME$_NEW_PASSWORD_2.

   d. To support deferred confirmation:
      - Compare the proposed password against password quality rules specific to your ACME agent. If it fails, do the following:
        – Call the ACME callback routine ACME$CB_QUEUE_DIALOGUE to provide a one-line error message describing the password quality problem.

- Return code ACME$_PERFORMDIALOGUE to have your error message transmitted to the ACM client process.
- When your ACME authentication and password callout routine is called again (indicating your error message was transmitted to the ACM client process), return code ACME$_RETRYPWD to cause the ACM dispatcher to return control to ACME$CO_NEW_PASSWORD_2.

- Call the ACME callback routine ACME$CB_QUEUE_DIALOGUE and specify the following:
  - An ITEM_CODE argument of ACME$_NEW_PASSWORD_2
  - A DATA_1 argument of that confirmation string
  - A FLAGS argument with ACMEDLOGFLG$V_INPUT and ACMEDLOGFLG$V_NOECHO set
- Return code ACME$_PERFORMDIALOGUE.
- When called back, retrieve the confirmation secondary password from parameter ITEM_LIST, using the last item on the chain of item list segments that has the item code ACME$_NEW_PASSWORD_2.
- Compare the original proposed password to the confirmation proposed password and if they do not match, issue an error message and restart step 4 to ask for the proposed new password.

When your ACME agent has received a confirmed string from the ACM client program, use that value (with leading and trailing spaces and tabs removed) for the desired new secondary password.

5. Call the ACME callback routine ACME$CB_SET_WQE_PARAMETER with an ID parameter of ACMEWQE$K_NEW_PASSWORD_2 and a DATA parameter describing the new secondary password.

# ACME$CO_NOTICES

In this routine each ACME agent has an opportunity to provide long text to the ACM client process that might be shown to a user after authorization. The ACM dispatcher will not invoke this ACME callout routine for the ACME$_FC_CHANGE_PASSWORD function code.

**Note**

Because many ACME agents will engage in the actions of this routine, no ACME agent should call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE.

## Format

ACME$CO_NOTICES kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list

## Description

**Processing Steps:**

If the status subfield of the ACMEWQE$R_STATUS field indicates a failure status, return ACME$_CONTINUE without sending any messages to the client process.

If there is no failure yet, there could be a subsequent failure during password change processing, but that is after the messages have been sent to the client process, and cannot be avoided.

If you decide to provide such information, your ACME agent should call the ACME callback routine ACME$CB_QUEUE_DIALOGUE successively to provide the output and then return code ACME$_PERFORMDIALOGUE to have it transmitted.

Since the order in which ACME agents are configured is under the control of the local system manager, your ACME agent should make no assumption that the text it provides will be displayed in any particular order with respect to other ACME agents, such as the traditional display from logical name SYS$WELCOME produced by the VMS ACME.

In many environments, system managers are sensitive regarding the volume of information presented to users logging in, so it is best if your ACME agent supports a mechanism for the system manager to control the amount of, or entirely disable, information that your ACME agent provides.

The ACM client process is not required to display the information you provide at all, much less with any particular prominence on its screen, printer, or tickertape.

# ACME$CO_PASSWORD_1

In this routine the designated DOI agent that needs a primary password obtains it from the ACM client process (prompting if necessary and possible).

## Format

ACME$CO_PASSWORD_1 kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list

## Description

**Processing Steps:**

1. Test flag ACMEWQEFLG$V_PREAUTHENTICATED and if it has been set during some prior ACME authentication and password callout routine, skip the following steps.
2. Test any value supplied for item ACME$_AUTH_MECHANISM. If it is some non-zero value other than ACMEMECH$K_PASSWORD, skip the following steps since this authentication is not password based, so the actions of this ACME authentication and password callout routine are irrelevant.
3. Test flag ACMEWQEFLG$V_PHASE_DONE. If it has been set (presumably by the corresponding ACME authentication and password callout routine for some prior ACME agent), skip the following steps.

   This test is required in addition to the following test for subfield ACMEWQEITM$L_ACME_ID of field ACMEWQEAX$R_PASSWORD_1, since that structure might not be filled in due either to some rule against sharing that is specific to a particular ACME agent or to an LGI callout routine returning the status LGI$_SKIPRELATED.
4. Test subfield ACMEWQEITM$L_ACME_ID of field ACMEWQEAX$R_ PASSWORD_1. If it has been set (presumably by the corresponding ACME authentication and password callout routine for some prior ACME agent), skip the following steps.
5. If item code ACME$_PASSWORD_1 was included in the Common item list that the ACM client program supplied on the initial call to the SYS$ACM[W] system service, use that value (with leading and trailing spaces and tabs removed) for the desired primary password.
6. If none of the preceding steps applied, do the following:
   - If no primary password is required by your ACME agent for this principal name, return ACME$_CONTINUE.
   - If this request is for function code ACME$_FC_CHANGE_ PASSWORD, test flag ACMEPWDFLG$V_PASSWORD_1 within field ACMEWQEAX$L_NEW_PASSWORD_FLAGS to see if the primary password is to be changed, and if not, return ACME$_CONTINUE.
   - Call the ACME callback routine ACME$CB_QUEUE_DIALOGUE specifying the following:
     - An ITEM_CODE parameter of ACME$_PASSWORD_1
     - A DATA_1 parameter of the prompt string

    - A FLAGS parameter with ACMEDLOGFLG$V_INPUT and ACMEDLOGFLG$V_NOECHO set
  - Return code ACME$_PERFORMDIALOGUE
  - When called back, retrieve the primary password from parameter ITEM_LIST, using the last item on the list that has the proper item code, using that value (with leading and trailing spaces and tabs removed) for the desired primary password.

7. Call the ACME callback routine ACME$CB_SET_WQE_PARAMETER with an ID parameter of ACMEWQE$K_PASSWORD_1 and a DATA parameter describing the primary password. Take this action regardless of whether your ACME agent views the password as "correct" because other ACME agents may need to see it. This is not a problem for passwords received as an item on an initial call, since those are available to all ACME agents.
8. Call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE to prevent other ACME agents from engaging in the purpose-based actions of this routine.

# ACME$CO_PASSWORD_2

In this routine the designated DOI agent that needs a secondary password obtains it from the ACM client process (prompting if necessary and possible).

## Format

ACME$CO_PASSWORD_2 kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list

## Description

**Processing Steps:**

1. Test flag ACMEWQEFLG$V_PREAUTHENTICATED and if it has been set during some prior ACME authentication and password callout routine, skip the following steps.
2. Test any value supplied for item ACME$_AUTH_MECHANISM. If it is some non-zero value other than ACMEMECH$K_PASSWORD, skip the following steps since this authentication is not password based, so the actions of this ACME authentication and password callout routine are irrelevant.
3. Test flag ACMEWQEFLG$V_PHASE_DONE. If it has been set (presumably by the corresponding ACME authentication and password callout routine for some prior ACME agent), skip the following steps.

   This test is required in addition to the following test for subfield ACMEWQEITM$L_ACME_ID of field ACMEWQEAX$R_PASSWORD_2 since that structure might not be filled in due either to some rule against sharing that is specific to a particular ACME agent or to an LGI callout routine returning the status LGI$_SKIPRELATED.
4. Test subfield ACMEWQEITM$L_ACME_ID of field ACMEWQEAX$R_ PASSWORD_2. If it has been set (presumably by the corresponding ACME authentication and password callout routine for some prior ACME agent), skip the following steps.
5. If item code ACME$_PASSWORD_2 was included in the Common item list that the ACM client program supplied on the initial call to the SYS$ACM[W] system service, use that value (with leading and trailing spaces and tabs removed) for the desired secondary password.
6. If none of the preceding steps applied, do the following:
   - If no secondary password is required by your ACME agent for this principal name, return ACME$_CONTINUE.
   - If this request is for function code ACME$_FC_CHANGE_ PASSWORD, test flag ACMEPWDFLG$V_PASSWORD_2 within field ACMEWQEAX$L_NEW_PASSWORD_FLAGS to see if the secondary password is to be changed, and if not, return ACME$_CONTINUE.
   - Call the ACME callback routine ACME$CB_QUEUE_DIALOGUE and specify the following:
     - An ITEM_CODE argument of ACME$_PASSWORD_2
     - A DATA_1 argument of the prompt string

    - A FLAGS argument with ACMEDLOGFLG$V_INPUT and ACMEDLOGFLG$V_NOECHO set
  - Return code ACME$_PERFORMDIALOGUE
  - When called back, retrieve the secondary password from parameter ITEM_LIST, using the last item on the list that has the proper item code, using that value (with leading and trailing spaces and tabs removed) for the desired secondary password.

7. Call the ACME callback routine ACME$CB_SET_WQE_PARAMETER with an ID parameter of ACMEWQE$K_PASSWORD_2 and a DATA parameter describing the secondary password. Take this action regardless of whether your ACME agent views the password as "correct" because other ACME agents may need to see it. This is not a problem for passwords received as an item on an initial call, since those are available to all ACME agents.
8. Call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE to prevent other ACME agents from engaging in the purpose-based actions of this routine.

# ACME$CO_PRINCIPAL_NAME

In this routine an ACME agent may choose to determine the principal name to be used for this request (prompting if necessary and possible).

## Format

ACME$CO_PRINCIPAL_NAME kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list

## Description

**Processing Steps:**

1. Test flag ACMEWQEFLG$V_PHASE_DONE. If it has been set (presumably by the corresponding ACME authentication and password callout routine for some prior ACME agent) skip the following steps.
2. Test subfield ACMEWQEITM$L_ACME_ID of field ACMEWQEAX$R_ PRINCIPAL_NAME. If it has been set (presumably by the corresponding ACME authentication and password callout routine for some prior ACME agent), skip the following steps.
3. If item code ACME$_PRINCIPAL_NAME_IN was included in the Common item list that the ACM client program supplied on the initial call to the SYS$ACM[W] system service, use that value (with leading and trailing spaces and tabs removed) for the desired principal name.
4. If none of the preceding steps applied, do the following:
   - Call the ACME callback routine ACME$CB_QUEUE_DIALOGUE specifying an ITEM_CODE parameter of ACME$_PRINCIPAL_NAME_ IN, along with that prompt string (parameter DATA_1) and a FLAGS parameter with ACMEDLOGFLG$V_INPUT set.
   - Return code ACME$_PERFORMDIALOGUE.
   - When called back, retrieve the principal name from parameter ITEM_ LIST, using the last item on the list that has the proper item code, using that value (with leading and trailing spaces and tabs removed) for the desired principal name.
   - Call the ACME callback routine ACME$CB_SET_WQE_PARAMETER with an ID parameter of ACMEWQE$K_PRINCIPAL_NAME and a DATA parameter describing the desired principal name.
   - Call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE to prevent other ACME agents from engaging in the purpose-based actions of this ACME authentication and password callout routine.

If your ACME agent is going to prompt for principal name, it must do so using the sequence shown in the list above. Since the VMS ACME is always enabled, it will perform those steps if no prior ACME agent does it. Thus, it is typically not necessary for your ACME agent to do so. An example of a case where you might wish to perform principal name in your own ACME agent is if you wanted to replace the string "Username:" with some other prompt, such as "Principal Name:" or "login:.

# ACME$CO_QUALIFY_PASSWORD_1

In this routine your ACME agent can evaluate a new primary password from the client. If your ACME agent returns ACME$_RETRYPWD from this ACME callout routine, the ACM dispatcher will cycle back to the ACME callout routine ACME$CO_NEW_PASSWORD_1. The reason for this call is to support input of a new primary password coordinated between ACME agents.

**Note**

Because many ACME agents will engage in the actions of this routine, no ACME agent should call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE.

## Format

| | |
|---|---|
| ACME$CO_QUALIFY_PASSWORD_1 | kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list |

## Description

**Processing Steps:**

1. If the ACMEWQEFLG$V_SKIP_NEW_PASSWORD flag is set, return ACME$_CONTINUE.
2. If no new password was supplied in field ACMEWQEAX$R_NEW_PASSWORD_1 of the ACMEWQEAX data structure, return ACME$_CONTINUE.
3. Compare the password stored in the field ACMEWQEAX$R_NEW_PASSWORD_1 of the ACMEWQEAX data structure pointed to by the authentication and password change WQE extension of the work queue entry against password quality rules specific to your ACME agent. If it fails, do the following:
   - Call the ACME callback routine ACME$CB_QUEUE_DIALOGUE to provide a one-line error message describing the password quality problem.
   - Return code ACME$_PERFORMDIALOGUE to have your error message transmitted to the ACM client process.
   - When your ACME authentication and password callout routine is called again (indicating your error message was transmitted to the ACM client process), return code ACME$_RETRYPWD to cause the ACM dispatcher to return control to ACME$CO_NEW_PASSWORD_1.

# ACME$CO_QUALIFY_PASSWORD_2

In this routine your ACME agent can evaluate a new secondary password from the client. If your ACME agent returns ACME$_RETRYPWD from this ACME callout routine, the ACM dispatcher cycles back to the ACME callout routine ACME$CO_NEW_PASSWORD_2. The reason for this call is to support input of a new secondary password coordinated between ACME agents.

**Note**

Because many ACME agents will engage in the actions of this routine, no ACME agent should call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE.

## Format

| | |
|---|---|
| ACME$CO_QUALIFY_PASSWORD_2 | kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list |

## Description

**Processing Steps:**

1. If the ACMEWQEFLG$V_SKIP_NEW_PASSWORD flag is set, return ACME$_CONTINUE.
2. If no new password was supplied in field ACME$QEAX_NEW_PASSWORD_2 of the ACMEWQEAX data structure, return ACME$_CONTINUE.
3. Compare the password stored in the field ACMEWQEAX$R_NEW_PASSWORD_2 of the ACMEWQEAX data structure pointed to by the authentication and password change WQE extension of the work queue entry against password quality rules specific to your ACME agent. If it fails, do the following:
   - Call the ACME callback routine ACME$CB_QUEUE_DIALOGUE to provide a one-line error message describing the password quality problem.
   - Return code ACME$_PERFORMDIALOGUE to have your error message transmitted to the ACM client process.
   - When your ACME authentication and password callout routine is called again (indicating your error message was transmitted to the ACM client process), return code ACME$_RETRYPWD to cause the ACM dispatcher to return control to ACME$CO_NEW_PASSWORD_2.

# ACME$CO_SET_PASSWORDS

In this routine your ACME agent should commit any agreed-upon new passwords.

**Note**

Because many ACME agents will engage in the actions of this routine, no ACME agent should call the ACME callback routine ACME$CB_SET_WQE_FLAG with FLAG argument ACMEWQEFLG$K_PHASE_DONE.

## Format

ACME$CO_SET_PASSWORDS kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list

## Description

**Processing Steps:**

1. If the ACMEWQEFLG$V_SKIP_NEW_PASSWORD flag is set, return ACME$_CONTINUE.
2. Perform all *Commit* operations required to change the primary or secondary password using only the contents of the following fields of the ACMEWQEAX data structure pointed to by the authentication and password change WQE extension of the work queue entry:
   - ACMEWQEAX$R_PRINCIPAL_NAME_OUT
   - ACMEWQEAX$R_VMS_USERNAME
   - ACMEWQEAX$R_PASSWORD_1
   - ACMEWQEAX$R_PASSWORD_2
   - ACMEWQEAX$R_NEW_PASSWORD_1
   - ACMEWQEAX$R_NEW_PASSWORD_2
   - ACMEWQEAX$L_NEW_PASSWORD_FLAGS
3. Exit with ACME$_CONTINUE.

# ACME$CO_SYSTEM_PASSWORD

In this routine the VMS ACME will see if system password processing is required, and if so, will validate the system password and keep prompting (if dialogue support allows) until it is correct.

## Format

| | |
|---|---|
| ACME$CO_SYSTEM_PASSWORD | kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list |

## Description

To set the ACMEWQEAX$R_SYSTEM_PASSWORD field in the authentication extension to the work queue entry, the VMS ACME calls ACME callback routine ACME$CB_SET_WQE_PARAMETER with an ID parameter of ACMEWQE$K_SYSTEM_PASSWORD. Typically the system password is handled only by the VMS ACME, and since the information need not be shared between ACME agents, the VMS ACME does not guarantee to store it in that cell.

# ACME$CO_VALIDATE_MAPPING

In this routine the VMS ACME determines whether the mapped VMS username is valid, and, in the process, reads user authorization information pertaining to that VMS username. Other DOIs use this phase to check for VMS username mapping mismatches.

## Format

ACME$CO_VALIDATE_MAPPING kcb_vector, acme_context, wqe, request_context, common_item_list, acme_item_list

## Description

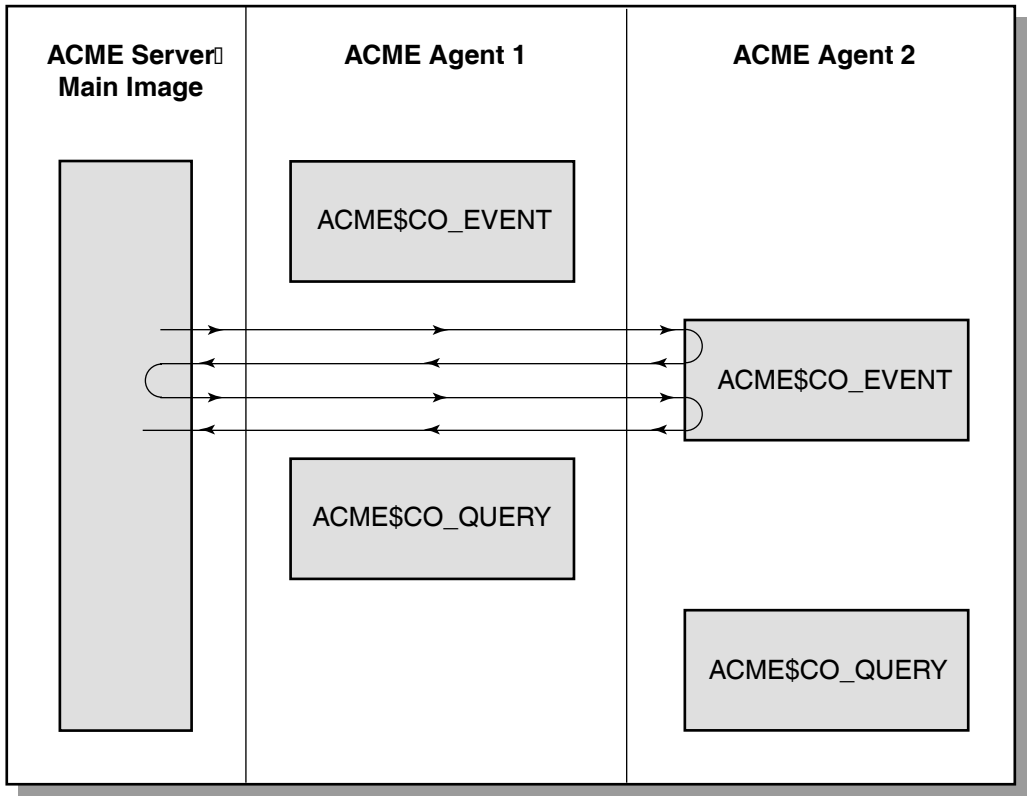
If your ACME agent is operating as a secondary DOI under model 1 (common passwords), check for a VMS username mismatch between the value stored in the WQE and the value that your ACME agent would have provided for this principal-name. Return ACME$_AUTHFAILURE after checking ACME$_MAPCONFLICT in the ACMESB secondary status field.

# 7

# ACME Event and Query Callout Routines

This chapter describes the possible ACME request processing callout routines to support the single-shot function codes ACME$_FC_EVENT and ACME$_FC_QUERY, which are always directed to a single ACME agent and do not involve dialogue processing. Different ACME agents may each supply their own version of the same ACME agent control callout routine and only those for a single ACME agent will be called, as shown in Figure 7–1.

**Figure 7–1 ACME Event and Query Callout Routine Control Flow**




VM-0784A-AI

Your ACME agent can use the WQE argument provided to ACME callout routines ACME$CO_EVENT and ACME$CO_QUERY in the same fashion as for ACME authentication and password callout routines. But, its lifetime is limited to a single invocation, coupled with those follow-up invocations due to returning the following status codes:

- ACME$_WAITRESOURCE

- ACME$_PERFORMDIALOGUE
- ACME$_WAITAST

Both routines are optional for your ACME agent.

## 7.1 Arguments for Event and Query Callout Routines

The arguments are the same for each callout routine and are described in Table 7–1.

**Table 7–1 Arguments fo ACME Event and Query Callout Routines**

| Argument | Description |
|---|---|
| kcb_vector | Address of an array called the KCB (Kernel Callback) vector. To interact with the ACME server main image and other ACME agents, your ACME agent must invoke various ACME callback routines. The procedure values of those callback routines are stored in the KCB vector. |
| acme_context | Address of a common quadword provided on all invocations of your ACME agent, for whatever purpose you deem fit. |
| wqe | Address of the work queue entry for this request. Has the structure described in Section B.16. Your ACME agent can read information directly out of the work queue entry, but should only change the contents of the work queue entry through the designated ACME callback routines. See Section 4.2. |
| request_context | Address of a single quadword provided on invocatins of your ACME agent in support of a particular request for whatever purpose you deem fit. |
| common_item_list | Consists of one or more item list segments chained together. All item list segments follow the 32-bit addressing item list format, to facilitate operation between ACME agents written in various programming languages (some of which might not support 64-bit addressing).<br><br>The chaining between item list segments does not reflect any initial chaining provided by the caller of the SYS$ACM[W] system service, but instead is the result of segmentation in the transfer of the item list data from the ACM client process context to the ACME server process context.<br><br>In the case of the ACME callout routiens ACME$CO_INITIALIZE, ACME$CO_EVENT, and ACME$CO_QUERY the items provided include those from the initial call to the SYS$ACM[W] system service. For subsequent ACME authentication and password callout routines other than ACME$CO_INITIALIZE, the items include only those specifically requested in dialogue interaction from that ACME callout routine.<br><br>**Common item list** items are those provided during this ACME callout routine with common item codes (ACME$_codename). To inspect items provided to previous ACME authentication and password callout routines for this request, use the chain of item list segments pointed to by the work queue entry cell ACMEWQE$PS_ ITEMLIST. |

(continued on next page)

**Table 7–1 (Cont.) Arguments fo ACME Event and Query Callout Routines**

| Argument | Description |
|---|---|
| acme_item_list | Consists of one or more item list segments chained together. All item list segments follow the 32-bit addressing item list format, to facilitate operation between ACME agents written in various programming languages (some of which might not support 64-bit addressing).<br>The chaining between item list segments does not reflect any initial chaining provided by the caller of the SYS$ACM[W] system service, but instead is the result of segmentation in the transfer of the item list data from the ACM client process context to the ACME server process context.<br>In the case of the ACME callout routiens ACME$CO_INITIALIZE, ACME$CO_EVENT, and ACME$CO_QUERY the items provided include those from the initial call to the SYS$ACM[W] system service. For subsequent ACME authentication and password callout routines other than ACME$CO_INITIALIZE, the items include only those specifically requested in dialogue interaction from that ACME callout routine.<br>ACME-specific item list items are those provided during this ACME callout routine with ACME-specific item codes (not ACME$_ codename) targeted at this ACME agent. Your ACME agent will not see any ACME-specific item codes targeted at other ACME agents. To inspect items provided to previous ACME authentication and password callout routines for this request, use the chain of item list segments pointed to by work queue entry cell ACMEWQE$PS_ ACME_ITEMLIST. |

# ACME$CO_EVENT

This routine is the only one called for function code ACME$_FC_EVENT. Since that function code always requires targeting a particular ACME agent, all processing is specific to the needs and definitions of your ACME agent.

## Format

ACME$CO_EVENT kcb_vector, acme_context, wqe, request_context, item_lists

## Description

The general outline for structuring your ACME$_FC_EVENT support is that your ACME agent performs some logging or processing, and possibly returns data for output item ACME$_EVENT_DATA_OUT, in response to the following input items:

- ACME$_EVENT_TYPE
- ACME$_EVENT_DATA_IN

Unlike the situation for the ACME authentication and password callout routines when your ACME$CO_EVENT ACME callout routine is called, you can be assured that the request received was made by a caller who is specifically interested in your ACME agent and will conform to whatever conventions you establish, including which item codes are actually supported.

If your ACME agent engages in logging based on calls to ACME$CO_EVENT, you should consider checking for particular privileges or rights identifiers being held by the caller of the SYS$ACM[W] system service to defend against a denial of service attack that attempts to fill your storage space. Normal SYS$PERSONA_QUERY calls using the persona ID stored in ACMEWQE$L_WQE_REQUESTOR_PROFILE can provide that information.

# ACME$CO_QUERY

This routine is the only one called for function code ACME$_FC_QUERY. Since that function code always requires targeting a particular ACME agent, all processing is specific to the needs and definitions of your ACME agent.

## Format

ACME$CO_QUERY kcb_vector, acme_context, wqe, request_context, item_lists

## Description

The general outline for structuring your ACME$_FC_QUERY support is that your ACME agent performs an inquiry and returns data for output item ACME$_QUERY_DATA, in response to the following input items :

- ACME$_QUERY_TYPE
- ACME$_QUERY_KEY_TYPE
- ACME$_QUERY_KEY_VALUE

Unlike the situation for the ACME authentication and password callout routines, when your ACME$CO_QUERY ACME callout routine is called you can be assured that the request received was made by a caller who is specifically interested inyour ACME agent and will conform to whatever conventions you establish, including which item codes are actually supported.

# 8

# ACME Status Codes

This chapter describes ACME status codes and messages, which are organized into the following categories:

- Flow control codes (see Section 8.1)
- Agent failure codes (see Section 8.2)
- Secondary codes (password quality) (see Section 8.3)
- Secondary codes (privileged) (see Section 8.4)
- Logging messages (see Section 8.5)
- Callback codes (see Section 8.6)
- SYS$ACM[W] codes (see Section 8.7)
- SET SERVER and SHOW SERVER codes (see Section 8.8)

## 8.1 Flow Control Codes

These status codes are returned by your ACME agent to control operation of the ACM dispatcher. They are not returned to the ACM client program.

CONTACTSYSMGR, requested operation has failed; contact the system manager

**Facility:** ACME, ACM Dispatcher Flow Control Codes

**Other language symbol:** ACME$_CONTACTSYSMGR

**Severity:** Error

**Explanation:** The ACME server or an ACME agent encountered some failure not due to any action of the user.

**User Action:** Choose the proper role-based action as follows:

**End User:**

A failure beyond your control has prevented authentication. Report this to the system manager.

**System Manager:**

Look in the ACME$SERVER log file for a detailed description. If ACME$SERVER is not defined as a logical name, this file will be named SYS$MANAGER:ACME$SERVER.LOG.

**ACME Programmer:**

Return this code from your ACME request processing callout routines (that is, anything but an ACME agent control callout routine) only after using ACME callback routine ACME$CB_SEND_LOGFILE to record circumstances of the failure.

PERFORMDIALOGUE, perform dialogue processing

**Facility:** ACME, ACM Dispatcher Flow Control Codes

**Other language symbol:** ACME$_PERFORMDIALOGUE

**Severity:** Warning

**Explanation:** An ACME authentication and password routine returns this code to request that queued dialogue messages be processed.

**User Action:** Choose the proper role-based action as follows:

**End User:**

Make a note of this error and tell the system manager what you were doing when the error occurred.

**System Manager:**

This message should never be presented to the end user. Consult the author or vendor of the program being used. If the program was supplied by HP, contact an HP support representative.

**ACME Programmer:**

Return this code to the ACM dispatcher from your ACME agent callout routine when dialogue items you have queued should be sent to the ACM client program.

RETRYPWD, new password is invalid; retry operation

**Facility:** ACME, ACM Dispatcher Flow Control Codes

**Other language symbol:** ACME$_RETRYPWD

**Severity:** Warning

**Explanation:** An ACME authentication and password routine returns this code to request that new password processing be restarted.

**User Action:** Choose the proper role-based action as follows:

**End User:**

Make a note of this error and tell the system manager what you were doing when the error occurred.

**System Manager:**

This message should never be presented to the end user. Consult the author or vendor of the program being used. If the program was supplied by HP, contact an HP support representative.

**ACME Programmer:**

Return this status code from your ACME authentication and password callout routine when *both* of the following conditions occur:

- The new password selected by the user is unacceptable.
- No ACME agent has indicated that it cannot retry with a different password.

WAITAST, wait for AST event

**Facility:** ACME, ACM Dispatcher Flow Control Codes

**Other language symbol:** ACME$_WAITAST

**Severity:** Warning

**Explanation:** An ACME authentication and password callout routine returns this code to request that processing of this request be suspended pending AST delivery.

**User Action:** Choose the proper role-based action as follows:

**End User:**

Make a note of this error and tell the system manager what you were doing when the error occurred.

**System Manager:**

This message should never be presented to the end user. Consult the author or vendor of the program being used. If the program was supplied by HP, contact an HP support representative.

**ACME Programmer:**

Return this status code from your ACME callout routine when your ACME agent must wait for AST delivery.

WAITRESOURCE, wait for ACME-specific resource

**Facility:** ACME, ACM Dispatcher Flow Control Codes

**Other language symbol:** ACME$_WAITRESOURCE

**Severity:** Warning

**Explanation:** An ACME authentication and password callout routine returns this code to request that processing of this request be suspended pending availability of the most recently requested ACME-specific resource.

**User Action:** Choose the proper role-based action as follows:

**End User:**

Make a note of this error and tell the system manager what you were doing when the error occurred.

**System Manager:**

This message should never be presented to the end user. Consult the author or vendor of the program being used. If the program was supplied by HP, contact an HP support representative.

**ACME Programmer:**

Return this status code from your ACME callout routine when your ACME agent must wait for the most recently requested ACME-specific resource to become available.

## 8.2 Agent Failure Codes

These status codes can be returned by your ACME agent to indicate to the ACM client program the failure of a request.

Although it is possible to return status codes specific to your ACME agent for special circumstances, using these standard codes (perhaps with a secondary status specific to your ACME agent) increases the ability of ACM client programs to provide a smooth user interface.

In the particular case of ACME-E-AUTHFAILURE, the secondary status is treated confidentially by not being disclosed to nonprivileged clients and by not being displayed by privileged clients. This prevents an attacker from learning what aspect of an authentication attempt was faulty.

ACCEXPIRED, account has expired

**Facility:** ACME, ACME Agent Failure Codes

**Other language symbol:** ACME$_ACCEXPIRED

**Severity:** Error

**Explanation:** The Authenticate Principal or Change Password request failed because the account for the user has expired.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Contact the authorizing authority to determine whether a user request to reactivate the account should be honored.

**ACME Programmer:**

The VMS ACME returns this status code if an account is expired in the SYSUAF record for this user. You may want your ACME agents to do the same, based on their own account expiration information.

AUTHFAILURE, authentication has failed

**Facility:** ACME, ACME Agent Failure Codes

**Other language symbol:** ACME$_AUTHFAILURE

**Severity:** Error

**Explanation:** The Authenticate Principal or Change Password request failed in authentication.

**User Action:** Choose the proper role-based action as follows:

**End User:**

Your authentication information is not acceptable to the system. Authorization failure can be due to a number of causes, including but not limited to: invalid password or token, expired or locked account, too many failed login attempts, invalid or unknown principal name. Check your work and if subsequent attempts fail, contact the system manager. Do not reveal your password or token to anyone, including the system manager, except under exceptional circumstances as designated in your site's security policy. If you do divulge your password to anyone, change it as soon as possible.

**System Manager:**

The system accounting file and system audit log, if enabled, contain details regarding the precise cause of the authentication failure. The authenticating agent does not share this failure information with unprivileged users so that it cannot be used to guess passwords or plan attacks.

**ACME Programmer:**

Your ACME agent should return this status code after calling ACME$CB_SET_2ND_STATUS to indicate the precise cause of the authentication failure.

INVALIDTIME, access is denied at this time

**Facility:** ACME, ACME Agent Failure Codes

**Other language symbol:** ACME$_INVALIDTIME

**Severity:** Error

**Explanation:** The Authenticate Principal or Change Password request failed because the user is not allowed to log in at this time.

**User Action:** Choose the proper role-based action as follows:

**End User:**

Try again at a time when your use is authorized.

**ACME Programmer:**

The VMS ACME returns this status code if an account is not authorized for this time of day in the SYSUAF record for this user. You may want your ACME agents to do the same, based on their own controls. The ACME agent may first call ACME$CB_SET_2ND_STATUS with a status code indicating the precise nature of the time-of-day restriction before returning ACME$_INVALIDTIME.

INVNEWPWD, new password is invalid

**Facility:** ACME, ACME Agent Failure Codes

**Other language symbol:** ACME$_INVNEWPWD

**Severity:** Error

**Explanation:** The Authenticate Principal or Change Password request failed because a new password provided was not acceptable.

**User Action:** Choose the proper role-based action as follows:

**End User:**

The new password failed to meet password quality checks by one or more ACME agents. Password quality checks include minimum and/or maximum lengths, dictionary match, history match, or other site-specific password policy filters. Choose a different new password conforming to your site password policies. Check your work and if subsequent attempts fail, contact the system manager. Do not reveal your password or token to anyone, including the system manager, except under exceptional circumstances as designated in your site's security policy. If you do divulge your password to anyone, change it as soon as possible.

**ACME Programmer:**

When another ACME agent has indicated it cannot handle new password retries, your ACME agent should return this status code after calling ACME$CB_SET_2ND_STATUS with a secondary status code that indicates the precise problem with the quality of the proposed new password.

NOTAUTHORIZED, authentication failed due to account restrictions

**Facility:** ACME, ACME Agent Failure Codes

**Other language symbol:** ACME$_NOTAUTHORIZED

**Severity:** Error

**Explanation:** Authorization failed because of account restrictions enforced by one or more ACME agents.

**User Action:** Choose the proper role-based action as follows:

**End User:**

Contact the system manager and provide information about the type of operation and the principal name used.

**ACME Programmer:**

Your ACME agent can return this message to the end user for a general authorization failure. The ACME agent may first call ACME$CB_SET_2ND_STATUS with a status code indicating the precise nature of the authorization failure before returning ACME$_NOTAUTHORIZED.

PWDCANTCHANGE, password cannot be changed

**Facility:** ACME, ACME Agent Failure Codes

**Other language symbol:** ACME$_PWDCANTCHANGE

**Severity:** Error

**Explanation:** The Authenticate Principal or Change Password request failed because a password cannot be changed.

**User Action:** Choose the proper role-based action as follows:

**End User:**

Contact your system manager and provide information about the type of operation and the principal name used.

**System Manager:**

If the password was not locked for a valid reason, review the security policy that prevents this password from being changed by the end user.

**ACME Programmer:**

The VMS ACME returns this status code if the SYSUAF record for this user indicates the password is locked. You may want your ACME agents to do the same, based on their own controls.

PWDEXPIRED, password has expired

**Facility:** ACME, ACME Agent Failure Codes

**Other language symbol:** ACME$_PWDEXPIRED

**Severity:** Error

**Explanation:** The Authenticate Principal or Change Password request failed because a password provided has expired and a new password is required to

complete the request. The SYS$ACM[W] system service returns this when called in non-dialogue mode (non-interactive operation).

**User Action:** Choose the proper role-based action as follows:

**ACME Programmer:**

The VMS ACME returns this status code if the SYSUAF record for this user indicates the password has expired and a new password is required. You may want your own ACME agents to do the same, based on their own controls.

PWDNOTCHG, password not changed

**Facility:** ACME, ACME Agent Failure Codes

**Other language symbol:** ACME$_PWDNOTCHG

**Severity:** Error

**Explanation:** The specified password could not be changed due to policy restrictions or system error.

**User Action:** Choose the proper role-based action as follows:

**End User:**

Review the password policy restrictions for your site and verify that your choice meets those requirements. Check your work and try again. If all else fails, contact the system manager.

**System Manager:**

If a system error was the reason for this error, the ACME$SERVER error log file may contain more information. If ACME$SERVER is not defined as a logical name, this file will be named SYS$MANAGER:ACME$SERVER.LOG.

## 8.3 Secondary Codes (Password Quality)

Your ACME agent callout routines might provide these messages as failure details by way of the ACME callback routine ACME$CD_SET_2ND_STATUS. Otherwise, an ACME callout routine might return an ACME-specific code.

PWDINDICT, password exists in dictionary database

**Facility:** ACME, ACME Secondary Codes (Password Quality)

**Other language symbol:** ACME$_PWDINDICT

**Severity:** Error

**Explanation:** The Authenticate Principal or Change Password request failed because a proposed new password is found in an ACME agent's password dictionary.

**User Action:** Choose the proper role-based action as follows:

**End User:**

Choose a password that is not a common word or acronym.

**ACME Programmer:**

When another ACME agent has indicated it cannot handle new password retries, the VMS ACME calls ACME$CB_SET_2ND_STATUS with this status code and then returns ACME$_INVNEWPWD if a proposed new password is present in its password dictionary file. You may want your ACME agents to do the same, based on their own controls.

PWDINHISTORY, password exists in history database

**Facility:** ACME, ACME Secondary Codes (Password Quality)

**Other language symbol:** ACME$_PWDINHISTORY

**Severity:** Error

**Explanation:** The Authenticate Principal or Change Password request failed because a proposed new password has been used too recently.

**User Action:** Choose the proper role-based action as follows:

**End User:**

Choose a different password that you have not used recently (as defined by your site-specific policy).

**ACME Programmer:**

When another ACME agent has indicated it cannot handle new password retries, the VMS ACME calls ACME$CB_SET_2ND_STATUS with this status code and then returns ACME$_INVNEWPWD if the hash value of a proposed new password is present and unexpired in its password history file. You may want your ACME agents to do the same, based on their own controls.

PWDINVALID, unspecified password policy restriction

**Facility:** ACME, ACME Secondary Codes (Password Quality)

**Other language symbol:** ACME$_PWDINVALID

**Severity:** Error

**Explanation:** The Authenticate Principal or Change Password request failed because a proposed new password is unacceptable for an unspecified reason.

**User Action:** Choose the proper role-based action as follows:

**End User:**

Choose a different password.

**ACME Programmer:**

When another ACME agent has indicated it cannot handle new password retries, your ACME agent could call ACME$CB_SET_2ND_STATUS with this status code and then return ACME$_INVNEWPWD if a proposed new password is unacceptable in some manner not covered by any other password quality secondary status for the ACME facility.

PWDINVCHAR, password contains invalid characters

**Facility:** ACME, ACME Secondary Codes (Password Quality)

**Other language symbol:** ACME$_PWDINVCHAR

**Severity:** Error

**Explanation:** The Authenticate Principal or Change Password request failed because a proposed new password contains characters unacceptable to the ACME agent.

**User Action:** Choose the proper role-based action as follows:

**End User:**

Choose a password without the unacceptable characters.

**ACME Programmer:**

When another ACME agent has indicated it cannot handle new password retries, the VMS ACME calls ACME$CB_SET_2ND_STATUS with this status code and then returns ACME$_INVNEWPWD if a proposed new password contains characters that are unacceptable for the password hash algorithm indicated in the SYSUAF record for this user. You may want your ACME agents to do the same, based on their own controls.

PWDTOOEASY, password can be easily guessed

**Facility:** ACME, ACME Secondary Codes (Password Quality)

**Other language symbol:** ACME$_PWDTOOEASY

**Severity:** Error

**Explanation:** The Authenticate Principal or Change Password request failed because a proposed new password is too easy to guess.

**User Action:** Choose the proper role-based action as follows:

**End User:**

Choose a password that is harder to guess.

**ACME Programmer:**

When another ACME agent has indicated it cannot handle new password retries, the VMS ACME calls ACME$CB_SET_2ND_STATUS with this status code and then returns ACME$_INVNEWPWD if a proposed new password can be too easily guessed. You may want your ACME agents to do the same, based on their own standards.

PWDTOOLONG, password greater than maximum length

**Facility:** ACME, ACME Secondary Codes (Password Quality)

**Other language symbol:** ACME$_PWDTOOLONG

**Severity:** Error

**Explanation:** The Authenticate Principal or Change Password request failed because a proposed new password is longer than allowed by policy or implementation.

**User Action:** Choose the proper role-based action as follows:

**End User:**

Choose a shorter password that is within the range permitted by your site-specific policy or implementation.

**ACME Programmer:**

When another ACME agent has indicated it cannot handle new password retries, the VMS ACME calls ACME$CB_SET_2ND_STATUS with this status code and then returns ACME$_INVNEWPWD if a proposed new password is longer than 32 characters. You may want your ACME agents to do the same, based on their own controls.

PWDTOOSHORT, password less than minimum length

**Facility:** ACME, ACME Secondary Codes (Password Quality)

**Other language symbol:** ACME$_PWDTOOSHORT

**Severity:** Error

**Explanation:** The Authenticate Principal or Change Password request failed because a proposed new password is shorter than allowed by policy.

**User Action:** Choose the proper role-based action as follows:

**End User:**

Choose a longer password.

**ACME Programmer:**

When another ACME agent has indicated it cannot handle new password retries, the VMS ACME calls ACME$CB_SET_2ND_STATUS with this status code and then returns ACME$_INVNEWPWD if a proposed new password is shorter than allowed by the SYSUAF record for this user. You may want your ACME agents to do the same, based on their own controls.

## 8.4 Secondary Codes (Privileged)

Your ACME callout routines might provide these messages as failure details by way of the ACME callback routine ACME$CB_SET_2ND_STATUS. Otherwise, your ACME agent might return an ACME-specific code.

**Security Warning**

For security reasons, any code indicating the precise reason for authentication failure should be provided only as a secondary status by way of a call to ACME callback routing ACME$CB_SET_2ND_STATUS, with ACME$_AUTHFAILURE returned for the primary status.

ACCOUNTLOCK, account is disabled

**Facility:** ACME, ACME Secondary Codes (Privileged)

**Other language symbol:** ACME$_ACCOUNTLOCK

**Severity:** Error

**Explanation:** The Authenticate Principal or Change Password request failed because the account associated with the specified principal name is disabled.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Presence of this status code in the accounting file or security auditing log may warrant an investigation regarding a possible attack against the system.

If the account of a legitimate user was locked, reconsider the administrative decision to lock this account. For the VMS ACME, the control is the DISUSER flag in the SYSUAF record for this user. Consult the appropriate documentation for other ACME agents to determine when they might return this status code.

**ACME Programmer:**

For confidentiality, your ACME agent should call ACME$CB_SET_ 2ND_STATUS with this status code to indicate the precise cause of the authentication failure for accounting and security auditing. Then your ACME agent should return the status code ACME$_AUTHFAILURE to the user.

INTRUDER, record matching request was found in the intrusion database

**Facility:** ACME, ACME Secondary Codes (Privileged)

**Other language symbol:** ACME$_INTRUDER

**Severity:** Error

**Explanation:** Authentication failed because successive attempts from this source have failed, creating intruder status.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Presence of this status code in the accounting file or security auditing log may warrant an investigation regarding a possible attack against the system.

Alternatively, a legitimate user may be having difficulties logging in.

**ACME Programmer:**

For confidentiality, your ACME agent should call ACME$CB_SET_ 2ND_STATUS with this status code to indicate the precise cause of the authentication failure for accounting and security auditing. Then your ACME agent should return the status code ACME$_AUTHFAILURE to the user.

INVALIDPWD, password is invalid

**Facility:** ACME, ACME Secondary Codes (Privileged)

**Other language symbol:** ACME$_INVALIDPWD

**Severity:** Error

**Explanation:** The Authenticate Principal or Change Password request failed because a provided password was invalid.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Presence of this status code in the accounting file or security auditing log may warrant an investigation regarding a possible attack against the system.

Alternatively, a legitimate user may just be having a bad password day.

**ACME Programmer:**

For confidentiality, your ACME agent should call ACME$CB_SET_ 2ND_STATUS with this status code to indicate the precise cause of the authentication failure for accounting and security auditing. Then your ACME agent should return the status code ACME$_AUTHFAILURE to the user.

INVMAPPING, principal name does not map to an OpenVMS user name

**Facility:** ACME, ACME Secondary Codes (Privileged)

**Other language symbol:** ACME$_INVMAPPING

**Severity:** Error

**Explanation:** An ACME agent returned a user name that does not exist in SYSUAF.DAT, the OpenVMS system authorization file.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Review the mapping configuration of your ACME agents.

**ACME Programmer:**

For confidentiality, the VMS ACME calls ACME$CB_SET_2ND_STATUS with this status code to indicate the precise cause of the authentication failure for accounting and security auditing. Then the VMS ACME returns the status code ACME$_AUTHFAILURE to the user.

MAPCONFLICT, principal name maps to a different OpenVMS user name

**Facility:** ACME, ACME Secondary Codes (Privileged)

**Other language symbol:** ACME$_MAPCONFLICT

**Severity:** Error

**Explanation:** An unprivileged caller asked to merge credentials using a principal name that an ACME agent mapped to a different OpenVMS user name than that associated with the current process.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Presence of this status code in the accounting file or security auditing log may warrant an investigation regarding a possible attack against the system.

Investigate the software being used to see if a configuration error is causing this problem.

**ACME Programmer:**

For confidentiality, your ACME agent should call ACME$CB_SET_ 2ND_STATUS with this status code to indicate the precise cause of the authentication failure for accounting and security auditing. Then your ACME agent should return the status code ACME$_AUTHFAILURE to the user.

NOEXTAUTH, principal name cannot be authenticated externally

**Facility:** ACME, ACME Secondary Codes (Privileged)

**Other language symbol:** ACME$_NOEXTAUTH

**Severity:** Error

**Explanation:** The attempted use of external authentication is not authorized for the specified principal name. An ACME agent other than the VMS ACME successfully authenticated a principal name that mapped to a SYSUAF record that did not have the EXTAUTH flag set.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Presence of this status code in the accounting file or security auditing log may warrant an investigation regarding a possible attack against the system.

It is more likely that this message indicates a configuration error with one of the ACME agents.

**ACME Programmer:**

For confidentiality, the VMS ACME calls ACME$CB_SET_2ND_STATUS with this status code to indicate the precise cause of the authentication failure for accounting and security auditing. Then the VMS ACME returns the status code ACME$_AUTHFAILURE to the user.

NOLOCAUTH, not authorized to override external authentication

**Facility:** ACME, ACME Secondary Codes (Privileged)

**Other language symbol:** ACME$_NOLOCAUTH

**Severity:** Error

**Explanation:** The Authenticate Principal or Change Password request failed because the specified principal name maps to an OpenVMS user name that is not authorized to override external authentication. In particular, the EXTAUTH flag is set and the VMSAUTH flag is clear in the SYSUAF record.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

If this error did not occur for a legitimate reason, presence of this status code in the accounting file or security auditing log may warrant an investigation regarding a possible attack against the system.

**ACME Programmer:**

For confidentiality, the VMS ACME calls ACME$CB_SET_2ND_STATUS with this status code to indicate the precise cause of the authentication failure for accounting and security auditing. Then the VMS ACME returns the status code ACME$_AUTHFAILURE to the user.

NOSUCHUSER, principal name does not exist or is invalid

**Facility:** ACME, ACME Secondary Codes (Privileged)

**Other language symbol:** ACME$_NOSUCHUSER

**Severity:** Error

**Explanation:** The Authenticate Principal or Change Password request failed because none of the ACME agents recognized the principal name. For the VMS ACME agent, this error indicates that the principal name does not exist as a user name in SYSUAF.DAT, the OpenVMS system authorization file.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

If this error did not occur for a legitimate reason, presence of this status code in the accounting file or security auditing log may warrant an investigation regarding a possible attack against the system.

**ACME Programmer:**

For confidentiality, your ACME agent should call ACME$CB_SET_2ND_STATUS with this status code to indicate the precise cause of the authentication failure for accounting and security auditing. Then your ACME agent should return the status code ACME$_AUTHFAILURE to the user.

## 8.5 Logging Messages

The following message is intended for use only in the ACME server log.

NOMSGFND, no acceptable message found

**Facility:** ACME, ACME Logging Messages

**Other language symbol:** ACME$_NOMSGFND

**Severity:** Error

**Explanation:** This internal status code is used within the ACME server main image. The ACME server main image may enter this into the ACME$SERVER log file when certain types of tracing are active.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

This message should only be logged, never received. Contact an HP support representative.

## 8.6 Callback Codes

An ACME callback routine can return these codes to your ACME agent. If one of these codes causes a nonrecoverable failure, your ACME agent should include the circumstances surrounding the error in the ACME server log file when logging the error. Then the ACME agent should return the error ACME$_CONTACTSYSMGR to the ACM dispatcher to transmit it to the client program.

This code should not be presented to an end user. Consult the author or vendor of the client program being used. If the program was supplied by HP, contact an HP support representative.

ASTCTXNOTFND, AST context not found

**Facility:** ACME, ACME Callback Codes

**Other language symbol:** ACME$_ASTCTXNOTFND

**Severity:** Error

**Explanation:** It was not possible to locate the specified AST context when an ACME agent called an ACME callback routine.

**User Action:** Choose the proper role-based action as follows:

**ACME Programmer:**

This message indicates a programming error. Review the programming of your ACME agent.

BUFFEROVF, output buffer overflow

**Facility:** ACME, ACME Callback Codes

**Other language symbol:** ACME$_BUFFEROVF

**Severity:** Informational

**Explanation:** The call completed successfully, but data has been truncated because the specified user buffer is too small for at least one output item returned by the call.

**User Action:** Choose the proper role-based action as follows:

**ACME Programmer:**

Review the sizes of output buffers.

**ACM Client Programmer:**

Review the sizes of output buffers that your client program provides for the SYS$ACM[W] system service.

BUFTOOSMALL, buffer too small

**Facility:** ACME, ACME Callback Codes

**Other language symbol:** ACME$_BUFTOOSMALL

**Severity:** Error

**Explanation:** An internal buffer allocation failure was encountered by the ACME server main image.

**User Action:** Contact an HP support representative.

DIALOGFULL, dialogue queue is full

**Facility:** ACME, ACME Callback Codes

**Other language symbol:** ACME$_DIALOGFULL

**Severity:** Error

**Explanation:** Too many dialogue entries have been queued for the available buffer space within the ACME server main image.

**User Action:** Choose the proper role-based action as follows:

**ACME Programmer:**

Send the existing dialogue entries to the client process by returning the status code ACME$_PERFORMDIALOGUE. When you call SYS$ACM[W] again, repeat the queueing of the dialogue entry that received the error.

DUPCREDTYP, credentials of the specified type have already been issued

**Facility:** ACME, ACME Callback Codes

**Other language symbol:** ACME$_DUPCREDTYP

**Severity:** Error

**Explanation:** An ACME agent made a duplicate call to ACME callback routine ACME$CB_ISSUE_CREDENTIALS.

**User Action:** Choose the proper role-based action as follows:

**ACME Programmer:**

This message indicates a programming error. Review the programming of your ACME agent.

INCONSTATE, internal consistency error

**Facility:** ACME, ACME Callback Codes

**Other language symbol:** ACME$_INCONSTATE

**Severity:** Fatal

**Explanation:** The ACME server detected an internal error.

**User Action:** Contact an HP support representative.

INSFDIALSUPPORT, insufficient dialogue support

**Facility:** ACME, ACME Callback Codes

**Other language symbol:** ACME$_INSFDIALSUPPORT

**Severity:** Error

**Explanation:** An ACME agent required dialogue support that is not provided by the call to the SYS$ACM[W] system service. In most cases, this error occurs because an ACM client called the SYS$ACM[W] service in non-dialogue mode and one or more ACME agents required interaction with the user; for example, the user's password has expired and a new one must be entered.

**User Action:** Choose the proper role-based action as follows:

**ACME Programmer:**

Your ACME agent should return this code if it discovers (for example, by calling callback routine ACME$CB_QUEUE_DIALOGUE) that a necessary dialogue support capability is not provided by the client program.

INVCREDTYP, agent is not authorized to issue the specified type of credentials

**Facility:** ACME, ACME Callback Codes

**Other language symbol:** ACME$_INVCREDTYP

**Severity:** Error

**Explanation:** An ACME agent made a call to ACME callback routine ACME$CB_ISSUE_CREDENTIALS specifying a credentials type that was not specified when the ACME agent was configured. This status code will also be returned if the ACME agent specifies a credentials type of 0 (zero).

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Check the SET SERVER ACME/CONFIGURE command that was used to configure this ACME agent.

**ACME Programmer:**

This message indicates a programming error. Check the code that calls ACME$CB_ISSUE_CREDENTIALS.

INVFLAG, flag number is invalid

**Facility:** ACME, ACME Callback Codes

**Other language symbol:** ACME$_INVFLAG

**Severity:** Error

**Explanation:** An ACME agent made a call to an ACME callback routine specifying an incorrect flag number.

**User Action:** Choose the proper role-based action as follows:

**ACME Programmer:**

This message indicates a programming error. Correct the flag number passed to the ACME callback routine.

INVPARAMETER, parameter selector or descriptor is invalid

**Facility:** ACME, ACME Callback Codes

**Other language symbol:** ACME$_INVPARAMETER

**Severity:** Error

**Explanation:** An ACME agent made a call to an ACME callback routine specifying an incorrect parameter.

**User Action:** Choose the proper role-based action as follows:

**ACME Programmer:**

This message indicates a programming error. Correct the parameter passed to the ACME callback routine.

NORMAL, normal successful completion

**Facility:** ACME, ACME Callback Codes

**Other language symbol:** ACME$_NORMAL

**Severity:** Success

**Explanation:** The operation completed successfully. The ACME callback was successful.

**User Action:** Choose the proper role-based action as follows:

**End User:**

None.

**ACME Programmer:**

Continue processing.

NOTOUTITEM, item code does not reflect an output item

**Facility:** ACME, ACME Callback Codes

**Other language symbol:** ACME$_NOTOUTITEM

**Severity:** Error

**Explanation:** A call to an ACME callback routine specified an address that is not the address of an output item passed to the ACME agent by the ACM dispatcher.

**User Action:** Choose the proper role-based action as follows:

**ACME Programmer:**

This message indicates a programming error. Review the programming of your ACME agent.

NULLVALUE, NULL value is invalid

**Facility:** ACME, ACME Callback Codes

**Other language symbol:** ACME$_NULLVALUE

**Severity:** Error

**Explanation:** A call to an ACME callback routine specified a null ACME-specific resource type.

**User Action:** Choose the proper role-based action as follows:

**ACME Programmer:**

This message indicates a programming error. Review the programming of your ACME agent.

RESOURCENOTAVAIL, requested resource is not available

**Facility:** ACME, ACME Callback Codes

**Other language symbol:** ACME$_RESOURCENOTAVAIL

**Severity:** Error

**Explanation:** A call to ACME callback routine ACME$CB_ACQUIRE_ RESOURCE could not be fulfilled.

**User Action:** Choose the proper role-based action as follows:

**ACME Programmer:**

In the likely event that your ACME agent cannot proceed without the ACME-specific resource, return the code ACME$_WAITRESOURCE to have your ACME agent called again at this ACME callout routine after an ACME-specific resource of the specified type becomes available. (Note that even when the resource becomes available, another request might make use of the resource first, causing your ACME agent to have to wait again.)

UNSUPPORTED, requested operation is unsupported

**Facility:** ACME, ACME Callback Codes

**Other language symbol:** ACME$_UNSUPPORTED

**Severity:** Error

**Explanation:** An ACME agent attempted to call an ACME callback routine from an invalid ACME agent callout routine, or the requested SYS$ACM[W] service is not supported.

**User Action:** Choose the proper role-based action as follows:

**ACME Programmer:**

This message indicates a programming error. Review the programming of your ACME agent.

**ACM Client Programmer:**

Revise the SYS$ACM[W] service request to fix the error.

UNSUPREVLVL, unsupported structure revision level

**Facility:** ACME, ACME Callback Codes

**Other language symbol:** ACME$_UNSUPREVLVL

**Severity:** Error

**Explanation:** A data structure revision level is incorrect.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Revisit the documentation for the ACME agent that logged this problem, paying particular attention to version compatibility issues.

**ACME Programmer:**

Unless you have written your ACME agent to adapt to multiple structure revision levels, use ACME callback routine ACME$CB_SEND_LOGFILE to record circumstances of this failure and then return status code ACME$_CONTACTSYSMGR to the ACM dispatcher.

## 8.7 SYS$ACM[W] Codes

The SYS$ACM[W] system service can return these errors to indicate an improper call by the ACM client or an internal failure in communications among the SYS$ACM[W] system service, the ACME server main image, and one or more ACME agents. These messages represent conditions detected outside the scope of the ACME server process, and therefore are not written to the ACME$SERVER log file. For more information about coding the ACM client, refer to the *HP OpenVMS Programming Concepts Manual* and the *HP OpenVMS System Services Reference Manual*.

DOIUNAVAILABLE, the domain of interpretation is not processing requests

**Facility:** ACME, ACME SYS$ACM[W] Codes

**Other language symbol:** ACME$_DOIUNAVAILABLE

**Severity:** Error

**Explanation:** The DOI specified as a target in the call to the SYS$ACM[W] system service is not currently available.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Enable the ACME agent that supports the DOI, or notify users that the DOI is unavailable.

**ACM Client Programmer:**

If your client program requires this DOI, contact the system manager.

INVALIDCTX, context argument does not match request parameters

**Facility:** ACME, ACME SYS$ACM[W] Codes

**Other language symbol:** ACME$_INVALIDCTX

**Severity:** Error

**Explanation:** A subsequent call to the SYS$ACM[W] system service did not contain the same function code and function modifiers as the original call, or it did not contain the proper item list as specified in the communications buffer itemset.

**User Action:** Choose the proper role-based action as follows:

**ACM Client Programmer:**

Review the programming of your ACM client program, particularly where it evaluates the itemset within the context block and prepares the item list for the next call to SYS$ACM[W]. Also ensure that other parameters required to remain constant across successive SYS$ACM[W] do not vary.

INVALIDTLV, invalid or corrupted TLV

**Facility:** ACME, ACME SYS$ACM[W] Codes

**Other language symbol:** ACME$_INVALIDTLV

**Severity:** Error

**Explanation:** There was an internal communications error between the SYS$ACM[W] system service and the ACME server main image.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Contact an HP support representative.

INVITMSEQ, invalid item code sequence

**Facility:** ACME, ACME SYS$ACM[W] Codes

**Other language symbol:** ACME$_INVITMSEQ

**Severity:** Error

**Explanation:** The SYS$ACM[W] system service encountered an invalid combination of query type and data item codes.

**User Action:** Choose the proper role-based action as follows:

**ACM Client Programmer:**

Check that your call to SYS$ACM[W] using the query function pairs the type and data item codes. ACME$_QUERY_KEY_TYPE must be followed by ACME$_QUERY_KEY_VALUE, and ACME$_QUERY_TYPE must be followed by ACME$_QUERY_DATA.

INVPERSONA, persona does not exist or handle is invalid

**Facility:** ACME, ACME SYS$ACM[W] Codes

**Other language symbol:** ACME$_INVPERSONA

**Severity:** Error

**Explanation:** The ACME$_PERSONA_HANDLE_IN item has an incorrect value.

**User Action:** Choose the proper role-based action as follows:

**ACM Client Programmer:**

Review the code your client program uses to provide personas to the SYS$ACM[W] system service.

INVREQUEST, parameter is invalid within the context of the request

**Facility:** ACME, ACME SYS$ACM[W] Codes

**Other language symbol:** ACME$_INVREQUEST

**Severity:** Error

**Explanation:** A call to the SYS$ACM[W] system service provided an invalid combination of function codes, function modifiers, and item codes.

**User Action:** Choose the proper role-based action as follows:

**ACM Client Programmer:**

Review the arguments provided to the SYS$ACM[W] system service to determine which ones are causing the error.

NOACMECTX, no ACME context is active

**Facility:** ACME, ACME SYS$ACM[W] Codes

**Other language symbol:** ACME$_NOACMECTX

**Severity:** Error

**Explanation:** In initial request processing, the SYS$ACM[W] system service encountered an ACME-specific item code when no ACME context had been established.

**User Action:** Choose the proper role-based action as follows:

**ACM Client Programmer:**

Verify that your dialogue mode requests are properly sequenced. Establish an ACME context by specifying one of the following item codes: ACME$_CONTEXT_ACME_ID, ACME$_CONTEXT_ACME_NAME, ACME$_TARGET_DOI_ID, or ACME$_TARGET_DOI_NAME.

NOCREDENTIALS, no credentials were issued

**Facility:** ACME, ACME SYS$ACM[W] Codes

**Other language symbol:** ACME$_NOCREDENTIALS

**Severity:** Error

**Explanation:**

Credentials were not issued.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Review the ACME$SERVER log to look for a failure in ACME agent processing. If ACME$SERVER is not defined as a logical name, this file will be named SYS$MANAGER:ACME$SERVER.LOG.

NOPRIV, insufficient privileges for the requested operation

**Facility:** ACME, ACME SYS$ACM[W] Codes

**Other language symbol:** ACME$_NOPRIV

**Severity:** Error

**Explanation:** The calling process does not possess the necessary privileges to issue the requested SYS$ACM[W] service or perform the requested SET SERVER ACME operation.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Evaluate whether privileges should be granted if this is an exceptional circumstance.

**ACM Client Programmer:**

Review the arguments provided to the SYS$ACM[W] system service to determine which ones are causing the status code.

NOSUCHDOI, the domain of interpretation does not exist

**Facility:** ACME, ACME SYS$ACM[W] Codes

**Other language symbol:** ACME$_NOSUCHDOI

**Severity:** Error

**Explanation:** An ACME agent supporting the DOI specified as a target in the call to the SYS$ACM[W] system service has not been loaded.

**User Action:** Choose the proper role-based action as follows:

**End User:**

Contact your system manager and provide information about the type of operation and the principal name used, and indicate the system component or application used when this error was received.

**System Manager:**

Load the ACME agent that supports the DOI, or notify users that the service is unavailable.

NOTARGETCRED, specified credentials do not exist

**Facility:** ACME, ACME SYS$ACM[W] Codes

**Other language symbol:** ACME$_NOTARGETCRED

**Severity:** Error

**Explanation:** The specified type of target credentials do not exist. The persona returned by the SYS$ACM[W] service does not contain credentials for the specified domain of interpretation (DOI).

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Review your configuration of various ACME agents and their matching **persona executive images** to ensure that the proper credentials are available.

OPINCOMPL, operation incomplete; interaction required

**Facility:** ACME, ACME SYS$ACM[W] Codes

**Other language symbol:** ACME$_OPINCOMPL

**Severity:** Warning

**Explanation:** A programming error has occurred. The caller of the SYS$ACM[W] system service must perform more dialogue processing to complete the request.

**User Action:** Choose the proper role-based action as follows:

**ACM Client Programmer:**

Review your dialogue mode calling sequence to ensure that your client program continues to call SYS$ACM[W] until SYS$ACM[W] returns a status other than OPINCOMPL.

THREADERROR, thread RTL error; status = 'hex-number'

**Facility:** ACME, ACME SYS$ACM[W] Codes

**Other language symbol:** ACME$_THREADERROR

**Severity:** Fatal

**Explanation:**

The ACME server main image encountered a threading error.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Contact an HP support representative.

TIMEOUT, requested operation has timed out

**Facility:** ACME, ACME SYS$ACM[W] Codes

**Other language symbol:** ACME$_TIMEOUT

**Severity:** Error

**Explanation:** ACME server process operations took too long.

**User Action:** Choose the proper role-based action as follows:

**End User:**

Contact your system manager and provide information about the type of operation and the principal name used, and indicate the system component or application used when this error was received.

**System Manager:**

Consult the vendor documentation to evaluate the range of possible timeout configuration options for the various ACME agents.

**ACME Programmer:**

Consider whether programming changes can reduce the time required.

## 8.8 SET SERVER and SHOW SERVER Codes

These status codes can be returned by the following commands:

- SET SERVER ACME
- SHOW SERVER ACME

Since these commands can be issued only by users with particular privileges, these messages presume the user receiving them has privileges to read the ACME$SERVER log.

ACME agents never deal with these status codes.

ACTIVE, authentication server is already active

**Facility:** ACME, ACME SET SERVER and SHOW SERVER Codes

**Other language symbol:** ACME$_ACTIVE

**Severity:** Error

**Explanation:** The authentication server process (ACME_SERVER) was already active when an attempt was made to start the server.

This message can also occur during an attempt to disable the server for reconfiguration. Attempts to disable the server result in this error until the server enters the disabled state.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Reconsider the order in which you invoke SET SERVER and SHOW SERVER ACME commands.

AGENTDBFULL, attempted to register more than the number of agents allowed

**Facility:** ACME, ACME SET SERVER and SHOW SERVER Codes

**Other language symbol:** ACME$_AGENTDBFULL

**Severity:** Error

**Explanation:** An attempt was made to configure more than the limit of eight (8) ACME agents (including the VMS ACME agent).

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Shut down the ACME server. Then restart the ACME server and load your choice of no more than seven (7) ACME agents.

AGENTLOADFAIL, agent image load failure

**Facility:** ACME, ACME SET SERVER and SHOW SERVER Codes

**Other language symbol:** ACME$_AGENTLOADFAIL

**Severity:** Error

**Explanation:** An error occurred while loading an ACME agent. This is usually a configuration error. Refer to the ACME$SERVER error log file for more information. If ACME$SERVER is not defined as a logical name, this file will be named SYS$MANAGER:ACME$SERVER.LOG.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Identify the offending ACME agent and take corrective action to eliminate the error. Consult the author or vendor of the ACME agent being used. If the agent was provided by HP, contact an HP support representative.

AUTHDOWN, authentication server is unavailable

**Facility:** ACME, ACME SET SERVER and SHOW SERVER Codes

**Other language symbol:** ACME$_AUTHDOWN

**Severity:** Fatal

**Explanation:** The ACME server is not available.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Restart the ACME server.

BUSY, authentication server is busy

**Facility:** ACME, ACME SET SERVER and SHOW SERVER Codes

**Other language symbol:** ACME$_BUSY

**Severity:** Error

**Explanation:** The ACME server process is too busy to handle the command. An authentication request may be in progress.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Wait and retry the command. If the command still fails with the BUSY status, issue a SHOW SERVER ACME command and check the server state in the resulting display.

If the server state indicates that there are requests in progress, and you want to disable or shut down the server right away, you can specify /CANCEL with /DISABLE or /EXIT to cancel all pending requests so that your command will be processed immediately. For example:

```
$ SET SERVER ACME /DISABLE /CANCEL
```

If you want to allow all requests currently in progress to complete (but accept no new requests) when you suspend, disable, or shut down the server, you can use the /WAIT qualifier with /EXIT, /DISABLE, or /SUSPEND. For example, the following command suspends new requests to the server but allows currently pending requests to complete:

```
$ SET SERVER ACME /SUSPEND /WAIT
```

If it takes too long for the pending requests to complete, you can press Ctrl/Y to abort this command and enter a command that specifies /CANCEL.

**Note**

Any new requests are queued, but they are not dispatched to the server until the server resumes operation.

DUPACME, an agent with the specified name has already been loaded

**Facility:** ACME, ACME SET SERVER and SHOW SERVER Codes

**Other language symbol:** ACME$_DUPACME

**Severity:** Error

**Explanation:** An attempt was made to configure an ACME agent with the name of a previously configured ACME agent. This error typically results from trying to configure the same ACME agent twice.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Use the SHOW SERVER ACME command and consult your records and command procedures to correct the attempted misconfiguration.

ERRCLOSELOGFIL, error closing log file; status = 'hex-number'

**Facility:** ACME, ACME SET SERVER and SHOW SERVER Codes

**Other language symbol:** ACME$_ERRCLOSELOGFIL

**Severity:** Error

**Explanation:** The ACME$SERVER log file could not be closed.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Examine the error status. If the error is not due to a device-off-line condition or another correctable situation, contact an HP support representative.

ERROPENCONFIGSFIL, error opening configuration staging file; status = 'hex_number'

**Facility:** ACME, ACME SET SERVER and SHOW SERVER Codes

**Other language symbol:** ACME$_ERROPENCONFIGSFIL

**Severity:** Error

**Explanation:** The ACME server could not open the ACME$SERVER_CONFIG configuration logging file upon startup. Automatic reconfiguration on restart will not be possible.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Look at the value of the logical name ACME$SERVER_CONFIG to see if it points to a nonexistent directory or a full disk. If ACME$SERVER_CONFIG is not defined as a logical name, the file will be named SYS$SYSTEM:ACME$SERVER_CONFIG.TMP. Correct the problem and restart the ACME server. If you do not correct the problem, the server will continue to run but it will not be able to recover the server's previous

configuration state if you subsequently restart the server. You will have to establish the state explicitly.

ERROPENLOGFIL, error opening log file; status = 'hex-number'

**Facility:** ACME, ACME SET SERVER and SHOW SERVER Codes

**Other language symbol:** ACME$_ERROPENLOGFIL

**Severity:** Error

**Explanation:** The ACME$SERVER log file could not be opened.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Look at the value of the logical name ACME$SERVER to see if it points to a nonexistent directory or a full disk.

ERROPENRESTARTFIL, error opening configuration restart file; status = 'hex_number'

**Facility:** ACME, ACME SET SERVER and SHOW SERVER Codes

**Other language symbol:** ACME$_ERROPENRESTARTFIL

**Severity:** Error

**Explanation:** The ACME server was unable to create the ACME$SERVER_RESTART file during server rundown. Automatic server reconfiguration on restart will not be possible.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Look at the value of the logical name ACME$SERVER_RESTART to see if it points to a nonexistent directory or a full disk. If ACME$SERVER_RESTART is not defined as a logical name, the file will be named SYS$SYSTEM:ACME$SERVER_RESTART.DAT. If the file cannot be created, the server will restart but it may not be able to recover its previous configuration state. You may have to establish the state explicitly.

ERRWRITELOGFIL, error writing log file; status = 'hex-number'

**Facility:** ACME, ACME SET SERVER and SHOW SERVER Codes

**Other language symbol:** ACME$_ERRWRITELOGFIL

**Severity:** Error

**Explanation:** The ACME$SERVER log file could not be written.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Check the status value displayed in the message to learn details about the failure; then take the appropriate action.

FAILURE, operation failure; if logging is enabled, see details in the ACME$SERVER log file

**Facility:** ACME, ACME SET SERVER and SHOW SERVER Codes

**Other language symbol:** ACME$_FAILURE

**Severity:** Error

**Explanation:** The ACME server or an ACME agent encountered a failure not due to user action.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

If logging is enabled, look in the ACME$SERVER log file for a detailed description. If ACME$SERVER is not defined as a logical name, the log file will be named SYS$MANAGER:ACME$SERVER.LOG.

**Note**

By default, the ACME server does *not* start up with logging enabled. That is, the SET SERVER ACME /START command does not enable logging. To enable logging and create a log file, you must execute the command SET SERVER ACME /LOG. Note that the /LOG qualifier *is* included in the system startup procedure used to start the ACME server.

**ACME Programmer:**

Return this code from your ACME agent control callout routines only after using ACME callback routine ACME$CB_SEND_LOGFILE to record circumstances of the failure.

INACTIVE, authentication server is not active

**Facility:** ACME, ACME SET SERVER and SHOW SERVER Codes

**Other language symbol:** ACME$_INACTIVE

**Severity:** Error

**Explanation:** The authentication server process (ACME_SERVER) was not active when an attempt was made to stop, disable, or configure the server.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Reconsider the order in which you invoke SET SERVER and SHOW SERVER ACME commands.

INCOMPATSTATE, server state is incompatible with requested operation

**Facility:** ACME, ACME SET SERVER and SHOW SERVER Codes

**Other language symbol:** ACME$_INCOMPATSTATE

**Severity:** Error

**Explanation:** The ACME server process cannot process this request in its current state.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Reconsider the order in which you invoke SET SERVER and SHOW SERVER ACME commands.

NOAGENTINIT, no agent initialization routine found

**Facility:** ACME, ACME SET SERVER and SHOW SERVER Codes

**Other language symbol:** ACME$_NOAGENTINIT

**Severity:** Error

**Explanation:** The specified ACME agent could not be loaded because the ACME agent does not contain an agent initialization routine.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Identify the offending ACME agent and take corrective action to eliminate the error. Consult the author or vendor of the ACME agent being used. If the agent was provided by HP, contact an HP support representative.

NOTSTARTED, authentication server failed to start

**Facility:** ACME, ACME SET SERVER and SHOW SERVER Codes

**Other language symbol:** ACME$_NOTSTARTED

**Severity:** Error

**Explanation:** The ACME server failed to start in response to the command.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Examine the server log (file SYS$MANAGER:ACME$SERVER.LOG by default or as defined by the ACME$SERVER logical name) to determine what error condition prevented the server from completing its startup. If the server process could not be started at all (there is no server log), check the accounting file to determine the process termination status.

Take corrective action to resolve the problem and restart the server.

SERVEREXIT, ACME_SERVER exiting

**Facility:** ACME, ACME SET SERVER and SHOW SERVER Codes

**Other language symbol:** ACME$_SERVEREXIT

**Severity:** Informational

**Explanation:** The ACME server has been terminated by the command.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Proceed to your next planned activity.

SERVERSTART, ACME_SERVER starting

**Facility:** ACME, ACME SET SERVER and SHOW SERVER Codes

**Other language symbol:** ACME$_SERVERSTART

**Severity:** Informational

**Explanation:** The ACME server has been started by the command.

**User Action:** Choose the proper role-based action as follows:

**System Manager:**

Proceed to configure the ACME server.

# 9
# ACME Callback Routines

This chapter lists the ACME callback routines. You can call any of these routines from any ACME callout routine you implement (except where noted). Use ACME callback routines for the following functions:

- Managing ACME-specific resources
- Managing AST contexts
- Managing virtual memory
- Reporting status to the ACME server main image
- Reporting status to the operations staff
- Communicating with the ACM client process
- Coordinating activities with other ACME agents

The following sections provide tips for using ACME callback routines.

## 9.1 Managing ACME-Specific Resources

Use the following callback routines to manage ACME-specific resources:

- ACME$CB_ACQUIRE_RESOURCE
- ACME$CB_RELEASE_RESOURCE

## 9.2 Managing AST Contexts

Within an ACME agent, never specify an address in your code as an AST handler when calling an OpenVMS system service or intervening library routine (for example, an SNA access routine).

Instead, call one of the four following *acquire* routines to obtain the AST handler and AST context values to pass to the system service or intervening library routine:

- ACME$CB_ACQUIRE_ACME_AST
- ACME$CB_ACQUIRE_ACME_RMSAST
- ACME$CB_ACQUIRE_WQE_AST
- ACME$CB_ACQUIRE_WQE_RMSAST

The “ACME” entry points return an AST handler and AST context value suitable for general use throughout the callout routines of an ACME agent, while the “WQE” entry points return an AST handler and AST context value only useful within the processing context of a single request.

The "_AST" entry points return a quadword AST context that can be passed as the ASTPRM parameter to normal system services, even from 64-bit registers. Only the low-order 32 bits are actually meaningful.

The "_RMSAST" entry points return a longword AST context that can be stored in the "$L_CTX" field of an RMS RAB or FAB.

In the event your ACME determines that the operating system will not deliver an AST, such as when certain system services return a *synchronous* status code, you must release the AST context you received, using one of the following callback routines:

- ACME$CB_RELEASE_ACME_AST
- ACME$CB_RELEASE_ACME_RMSAST
- ACME$CB_RELEASE_WQE_AST
- ACME$CB_RELEASE_WQE_RMSAST

When an AST is delivered that automatically releases the associated AST context, your ACME agent should **not** make a call to the release routine.

## 9.3 Managing Virtual Memory

Your ACME agent must not stall during operations and must return to its caller, if it needs to wait for external events. Thus, use stack-based storage locations for short periods of time. Allocate heap-based storage for data that must be accessed on a subsequent invocation. You must allocate and deallocate that storage with the following four callbacks:

- ACME$CB_ALLOCATE_ACME_VM
- ACME$CB_ALLOCATE_WQE_VM
- ACME$CB_DEALLOCATE_ACME_VM
- ACME$CB_DEALLOCATE_WQE_VM

The "ACME" entry points allocate and deallocate memory suitable for general use throughout the callout routines of an ACME agent, while the "WQE" entry points allocate and deallocate memory only useful within the processing context of a single request.

## 9.4 Reporting Status to the ACME Server Main Image

Use the following callback routines to report general information about your ACME agent to the ACME server main image:

- ACME$CB_REPORT_ACTIVITY
- ACME$CB_REPORT_ATTRIBUTES

## 9.5 Reporting Status to the Operations Staff

At more secure sites, only members of the security staff are allowed to enable their terminals as security operator terminals, and ordinary operators do not have the privilege to read the ACME server log. Use the following callback routines to report error or tracing information to security staff or operators, depending on the site policy for the handling of such information.

- ACME$CB_SEND_LOGFILE

- ACME$CB_SEND_OPERATOR

## 9.6 Communicating with the ACM Client Process

Use the following ACME callback routines to provide final output to the ACM client program and also to provide intermediate output and solicit intermediate input:

- ACME$CB_CANCEL_DIALOGUE
- ACME$CB_FORMAT_DATE_TIME
- ACME$CB_ISSUE_CREDENTIALS
- ACME$CB_SET_2ND_STATUS
- ACME$CB_SET_ACME_STATUS
- ACME$CB_SET_LOGON_FLAG
- ACME$CB_SET_LOGON_STATS_DOI
- ACME$CB_SET_LOGON_STATS_VMS
- ACME$CB_QUEUE_DIALOGUE
- ACME$CB_SET_OUTPUT_ITEM

## 9.7 Coordinating Activities with Other ACME Agents

ACME agents use the following ACME callback routines to decide which ACME agent will handle various processing steps:

- ACME$CB_SET_DESIGNATED_DOI
- ACME$CB_SET_PHASE_EVENT
- ACME$CB_SET_WQE_FLAG
- ACME$CB_SET_WQE_PARAMETER

## 9.8 Callback Routine Reference Section

The rest of this chapter describes each ACME callback routine in detail. The routines are presented in alphabetical order.

**Note**

Callback routines that specify a return code of ACME$_NORMAL may also return SS$_NORMAL under certain situations.

# AST_ROUTINE

Template for your AST routine. Use this as the template for your AST_ROUTINE provided to ACME callout routines.

## Format

AST_ROUTINE kcb_vector, acme_context, wqe, request_context, ast_context, ast_parameter

## Arguments

**kcb_vector**
OpenVMS usage: ACM_CALLBACK_VECTOR
type: acmekcv
access: read only
mechanism: reference

Address of KCB vector

**acme_context**
OpenVMS usage: QUADWORD_UNSIGNED
type: acmewqe
access: modify
mechanism: reference
mechanism:

Address of ACME context quadword

**wqe**
OpenVMS usage: ACM_WORK_QUEUE_ENTRY
type: acmewqe
access: read only
mechanism: reference

Address of the work queue entry

**request_context**
OpenVMS usage: QUADWORD_UNSIGNED
type: unsigned quadword
access: read only
mechanism: reference

Address of request context quadword

**ast_context**
OpenVMS usage: QUADWORD_UNSIGNED
type: unsigned quadword
access: read only
mechanism: reference

Address of AST context

**ast_parameter**
OpenVMS usage: QUADWORD_UNSIGNED
type: unsigned quadword
access: read only

mechanism: reference

Address of AST parameter

## Description

This routine does whatever is appropriate for your ACME agent on completion of I/O. Unlike AST routines called by system services, this AST routine is called at non-AST level.

## Return Values

None.

# ACME$CB_ACQUIRE_RESOURCE

Acquire an ACME-specific resource (access via ACMEKCV$CB_ACQUIRE_RESOURCE).

Retrieves an ACME-specific resource your ACME agent had previously released to the ACME server main image.

## Format

ACME$CB_ACQUIRE_RESOURCE wqe, resource_type, resource_value

### Valid from ACME Callouts

All ACME callout routines

### Related Codes You Can Return

ACME$_WAITRESOURCE

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**resource_type**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | read only |
| mechanism: | value |

(ACME defined) type of ACME-specific resource to retrieve

**resource_value**

| | |
|---|---|
| OpenVMS usage: | QUADWORD_UNSIGNED |
| type: | unsigned quadword |
| access: | write only |
| mechanism: | reference |

Address of quadword to receive the value of the ACME-specific resource allocated

## Description

An ACME-specific resource is represented by a quadword value that your ACME originally *releases* to the ACME server main image and which your ACME can later *acquire* from the ACME server main image. In creating each ACME-specific resource you specify a 32-bit ACME-specific resource type number of your own choosing.

If you create only one ACME-specific resource with a given ACME-specific resource type, you can use that ACME-specific resource with the code ACME$_WAITRESOURCE to create a mutual exclusion semaphore.

If you create multiple ACME-specific resources with a given ACME-specific resource type, those ACME-specific resources can be used with the code ACME$_WAITRESOURCE to limit the number of simultaneous activities of a particular type. The ACME-specific resource value associated with each ACME-specific resource is chosen by your ACME when it first releases the ACME-specific resource to the ACME server main image. Thus, you can use that ACME- specific resource value to represent the address of a data structure in memory, a value in a hash table, a record in a disk file, or any similar reminder.

If you have provided any ACME-specific resources to the ACME server main image, you must perform this callback for each at least by the end of the next invocation of ACME$CO_FINISH.

If your ACME agent returns ACME$_WAITRESOURCE and then is called back again, it is not guaranteed that the next attempt to acquire the ACME-specific resource will succeed. In addition to the request that gives up an ACME-specific resource, there could be multiple requests being processed that are waiting for the ACME-specific resource. When a single instance of the ACME-specific resource is freed, all requests waiting for it may become active, but only one will be able to get the ACME-specific resource.

### Related Callbacks

ACME$CB_RELEASE_RESOURCE

### Alternative Callbacks

None.

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Resource has been acquired and removed from the list. |
| ACME$_RESOURCENOTAVAIL | Requested ACME-specific resource is not available. |
| ACME$_NULLVALUE | Resource type of NULL was specified and is invalid. |

# ACME$CB_ACQUIRE_ACME_AST

Acquire an ACME-wide AST context (access via ACMEKCV$CB_ACQUIRE_ACME_AST).

Returns a 64-bit AST context for use between diverse requests.

## Format

ACME$CB_ACQUIRE_ACME_AST wqe, ast_handler, ast_context, ast_routine, ast_parameter

### Valid from ACME Callouts

All ACME callout routines

### Related Codes You Can Return

ACME$_WAITAST

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**ast_handler**

| | |
|---|---|
| OpenVMS usage: | AST_PROCEDURE |
| type: | procedure value |
| access: | write only |
| mechanism: | reference |

Address of 64-bit pointer to receive AST handler interceptor procedure address

**ast_context**

| | |
|---|---|
| OpenVMS usage: | QUADWORD_UNSIGNED |
| type: | unsigned quadword |
| access: | write only |
| mechanism: | reference |

Address of quadword to receive the AST parameter that should be passed to system services using this AST context

**ast_routine**

| | |
|---|---|
| OpenVMS usage: | AST_PROCEDURE |
| type: | procedure value |
| access: | read only |
| mechanism: | call without stack unwinding |

Address of ACME's AST service routine to invoke upon AST delivery

**ast_parameter**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned quadword |
| access: | read only |
| mechanism: | reference |

Address of longword containing ACME specific value to pass to the ACME's AST service routine upon AST delivery

## Description

This callback creates an AST context that contains the address of the AST routine and AST parameter specified by your ACME agent. In the AST handler and AST context locations your ACME agent specifies, this callback stores the values your ACME agent should specify for ASTADR and ASTPRM in calling a system service or library routine.

### Related Callbacks

ACME$CB_RELEASE_ACME_AST

### Alternative Callbacks

ACME$CB_ACQUIRE_ACME_RMSAST
ACME$CB_ACQUIRE_WQE_AST
ACME$CB_ACQUIRE_WQE_RMSAST

## Return Values

| | |
|---|---|
| ACME$_NORMAL | AST context has been established. |
| ACME$_INVPARAMETER | No AST routine was specified. |
| other | Any failure condition from allocating memory. |

# ACME$CB_ACQUIRE_ACME_RMSAST

Acquire an ACME-Wide RMS ACME-wide AST context (access via ACMEKCV$CB_ACQUIRE_ACME_RMSAST).

Returns a 32-bit AST context for use between diverse requests.

## Format

ACME$CB_ACQUIRE_ACME_RMSAST wqe, ast_handler, ast_context ast_routine, ast_parameter

### Valid from ACME Callouts

All ACME callout routines

### Related Codes You Can Return

ACME$_WAITAST

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**ast_handler**

| | |
|---|---|
| OpenVMS usage: | AST_PROCEDURE |
| type: | procedure value |
| access: | write only |
| mechanism: | reference |

Address of 64-bit pointer to receive AST handler (interceptor procedure) address

**ast_context**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | write only |
| mechanism: | reference |

Address of longword to receive the AST parameter that should be passed to RMS services using this AST context

**ast_routine**

| | |
|---|---|
| OpenVMS usage: | AST_PROCEDURE |
| type: | procedure value |
| access: | read only |
| mechanism: | call without stack unwinding |

Address of ACME's AST service routine to invoke upon AST delivery

**ast_parameter**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | read only |
| mechanism: | reference |

Address of longword containing ACME-specific value to pass to the ACME's AST service routine upon AST delivery

## Description

This callback creates an AST context that contains the address of the AST routine and AST parameter specified by your ACME agent. In the AST handler and AST context locations your ACME agent specifies, this callback stores the values your ACME agent should specify for ASTADR and ASTPRM in calling a system service or library routine.

### Related Callbacks

ACME$CB_RELEASE_ACME_RMSAST

### Alternative Callbacks

ACME$CB_ACQUIRE_ACME_AST
ACME$CB_ACQUIRE_WQE_AST
ACME$CB_ACQUIRE_WQE_RMSAST

## Return Values

| | |
|---|---|
| ACME$_NORMAL | AST context has been established. |
| ACME$_INVPARAMETER | No AST routine was specified. |
| other | Any failure condition from allocating memory. |

# ACME$CB_ACQUIRE_WQE_AST

Acquire a request-specific ACME-wide AST context (access via ACMEKCV$CB_ACQUIRE_WQE_RMSAST).

Returns a 64-bit AST context for use within a single request.

## Format

ACME$CB_ACQUIRE_WQE_AST wqe, ast_handler, ast_context, ast_routine, ast_parameter

### Valid from ACME Callouts

All ACME callout routines

### Related Codes You Can Return

ACME$_WAITAST

## Returns

VMS Usage: cond_value
type: integer
access: read only
mechanism: by value in R0

## Arguments

**wqe**
OpenVMS usage: ACM_WORK_QUEUE_ENTRY
type: acmewqe
access: modify
mechanism: reference

Address of the work queue entry

**ast_handler**
OpenVMS usage: AST_PROCEDURE
type: procedure value
access: write only
mechanism: reference

Address of 64-bit pointer to receive AST handler (interceptor procedure) address

**ast_context**
OpenVMS usage: QUADWORD_UNSIGNED
type: unsigned quadword
access: write only
mechanism: reference

Address of quadword to receive the AST parameter that should be passed to system services using this AST context

**ast_routine**

| | |
|---|---|
| OpenVMS usage: | AST_PROCEDURE |
| type: | procedure value |
| access: | read only |
| mechanism: | call without stack unwinding |

Address of ACME's AST service routine to invoke upon AST delivery

**ast_parameter**

| | |
|---|---|
| OpenVMS usage: | QUADWORD_UNSIGNED |
| type: | unsigned quadword |
| access: | read only |
| mechanism: | reference |

Address of quadword containing ACME specific value to pass to the ACME's AST service routine upon AST delivery

## Description

This callback creates an AST context that contains the address of the AST routine and AST parameter specified by your ACME agent. In the AST handler and AST context locations your ACME agent specifies, this callback stores the values your ACME agent should specify for ASTADR and ASTPRM in calling a system service or library routine.

Specifying an AST routine address of zero is not recommended and may not be supported in the future.

### Related Callbacks

ACME$CB_RELEASE_WQE_AST

### Alternative Callbacks

ACME$CB_ACQUIRE_ACME_AST
ACME$CB_ACQUIRE_ACME_RMSAST
ACME$CB_ACQUIRE_WQE_RMSAST

## Return Values

| | |
|---|---|
| ACME$_NORMAL | AST context has been established. |
| ACME$_UNSUPPORTED | Unsupported execution context. |
| other | Any failure condition from allocating memory. |

# ACME$CB_ACQUIRE_WQE_RMSAST

Acquire a request-specific RMS ACME-wide AST context (access via ACMEKCV$CB_ACQUIRE_WQE_RMSAST).

Returns a 32-bit AST context for use within a single request.

## Format

| | |
|---|---|
| ACME$CB_ACQUIRE_WQE_RMSAST | wqe, ast_handler, ast_context ast_routine, ast_parameter |

### Valid from ACME Callouts

All ACME callout routines

### Related Codes You Can Return

ACME$_WAITAST

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**ast_handler**

| | |
|---|---|
| OpenVMS usage: | AST_PROCEDURE |
| type: | procedure value |
| access: | write only |
| mechanism: | reference |

Address of 64-bit pointer to receive AST handler interceptor procedure) address

**ast_context**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | write only |
| mechanism: | reference |

Address of longword to receive the AST parameter that should be passed to RMS services using this AST context

**ast_routine**

| | |
|---|---|
| OpenVMS usage: | AST_PROCEDURE |
| type: | procedure value |
| access: | read only |
| mechanism: | call without stack unwinding |

Address of ACME's AST service routine to invoke upon AST delivery

**ast_parameter**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | read only |
| mechanism: | reference |

Address of longword containing ACME specific value to pass to the ACME's AST service routine upon AST delivery

## Description

This callback creates an AST context that contains the address of the AST routine and AST parameter specified by your ACME agent. In the AST handler and AST context locations your ACME agent specifies, this callback stores the values your ACME agent should specify for ASTADR and ASTPRM in calling a system service or library routine.

Specifying an AST routine address of zero is not recommended and may not be supported in the future.

### Related Callbacks

ACME$CB_RELEASE_WQE_RMSAST

### Alternative Callbacks

ACME$CB_ACQUIRE_ACME_AST
ACME$CB_ACQUIRE_ACME_RMSAST
ACME$CB_ACQUIRE_WQE_AST

## Return Values

| | |
|---|---|
| ACME$_NORMAL | AST context has been established. |
| ACME$_UNSUPPORTED | Unsupported execution context. |
| other | Any failure condition from allocating memory. |

# ACME$CB_ALLOCATE_ACME_VM

Allocate ACME-wide virtual memory (access via ACMEKCV$CB_ALLOCATE_ACME_VM).

Allocates virtual memory for use between diverse requests.

## Format

ACME$CB_ALLOCATE_ACME_VM wqe, segment_size, segment_address

### Valid from ACME Callouts

All ACME callout routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**segment_size**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | read only |
| mechanism: | reference |

Number of bytes to allocate

**segment_address**

| | |
|---|---|
| OpenVMS usage: | QUADWORD_UNSIGNED |
| type: | unsigned quadword |
| access: | write only |
| mechanism: | reference |

Address of a longword to receive the address of the memory allocated by this procedure.

## Description

You can retain memory you allocate with this call until the next invocation of ACME$CO_AGENT_SHUTDOWN and you can access it from any ACME callout routine in your ACME agent.

### Related Callbacks

ACME$CB_DEALLOCATE_ACME_VM

### Alternative Callbacks

ACME$CB_ALLOCATE_WQE_VM

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Memory has been allocated. |
| other | Any failure condition from allocating memory. |

# ACME$CB_ALLOCATE_WQE_VM

Allocate request-specific virtual memory (access via ACMEKCV$CB_ALLOCATE_WQE_VM).

Allocates virtual memory for use within a single request.

## Format

ACME$CB_ALLOCATE_WQE_VM wqe, segment_size, segment_address

### Valid from ACME Callouts

All request processing routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**segment_size**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | read only |
| mechanism: | reference |

Number of bytes to allocate

**segment_address**

| | |
|---|---|
| OpenVMS usage: | QUADWORD_UNSIGNED |
| type: | unsigned quadword |
| access: | write only |
| mechanism: | reference |

Address of a longword to receive the address of the memory allocated by this procedure.

## Description

You can only retain memory you allocate with this call for the duration of a single request, and you can only access it from an ACME callout routine that is servicing this request for which it was allocated. In the case of ACME$CO_EVENT or ACME$CO_QUERY, this means you must deallocate the memory before your final return to the ACM dispatcher.

### Related Callbacks

ACME$CB_DEALLOCATE_WQE_VM

### Alternative Callbacks

ACME$CB_ALLOCATE_ACME_VM

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Memory has been allocated. |
| ACME$_UNSUPPORTED | Unsupported execution context. |
| other | Any failure condition from allocating memory. |

# ACME$CB_CANCEL_DIALOGUE

Cancel pending dialogue items queued by ACME$CB_QUEUE_DIALOGUE (access via ACMEKCV$CB_CANCEL_DIALOGUE).

Cancels dialogue items that have been queued but not yet transmitted to the ACM client process.

## Format

ACME$CB_CANCEL_DIALOGUE wqe

### Valid from ACME Callouts

Authenticate Principal and Change Password routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

## Description

Until your ACME callout routine returns the code ACME$_PERFORMDIALOGUE, dialogue information you provide with ACME$CB_QUEUE_DIALOGUE is retained by the ACME server main image and not transmitted to the ACM client process. This ACME callback routine ACME$CB_CANCEL_DIALOGUE causes that untransmitted dialogue information to be discarded and not transmitted to the ACM client process.

### Related Callbacks

ACME$CB_QUEUE_DIALOGUE

### Alternative Callbacks

None.

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Dialogue entries have been cancelled. |
| ACME$_UNSUPPORTED | Unsupported execution context. |

# ACME$CB_DEALLOCATE_ACME_VM

Deallocate ACME-wide virtual memory (access via ACMEKCV$CB_DEALLOCATE_ACME_VM).

Deallocates virtual memory for use between diverse requests.

## Format

ACME$CB_DEALLOCATE_ACME_VM wqe, segment_size, segment_address

### Valid from ACME Callouts

All ACME callout routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**segment_size**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | read only |
| mechanism: | reference |

Number of bytes to deallocate

**segment_address**

| | |
|---|---|
| OpenVMS usage: | QUADWORD_UNSIGNED |
| type: | unsigned quadword |
| access: | read only |
| mechanism: | reference |

Address of first byte to deallocate

## Description

You can use this ACME callback routine during any ACME callout routine to deallocate memory allocated with ACME$CB_ALLOCATE_ACME_VM. You must perform this deallocation at least by the end of the next invocation of ACME$CO_AGENT_SHUTDOWN.

### Related Callbacks

ACME$CB_ALLOCATE_ACME_VM

### Alternative Callbacks

ACME$CB_DEALLOCATE_WQE_VM

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Memory has been released. |
| other | Any failure condition from deallocating memory. |

# ACME$CB_DEALLOCATE_WQE_VM

Deallocate request-specific virtual memory (access via ACMEKCV$CB_DEALLOCATE_WQE_VM).

Deallocates virtual memory for use within a single request.

## Format

ACME$CB_DEALLOCATE_WQE_VM wqe, segment_size, segment_address

### Valid from ACME Callouts

All request processing routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**segment_size**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | read only |
| mechanism: | reference |

Number of bytes to deallocate

**segment_address**

| | |
|---|---|
| OpenVMS usage: | QUADWORD_UNSIGNED |
| type: | unsigned quadword |
| access: | read only |
| mechanism: | reference |

Address of first byte to deallocate

## Description

You can use this ACME callback routine during ACME callout routines associated with the same request from which the memory was allocated with ACME$CB_ALLOCATE_ACME_VM. You must perform this deallocation at least by the end of the corresponding invocation of ACME$CO_FINISH.

### Related Callbacks

ACME$CB_ALLOCATE_WQE_VM

### Alternative Callbacks

ACME$CB_DEALLOCATE_ACME_VM

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Memory has been released. |
| other | Any failure condition from deallocating memory. |

# ACME$CB_FORMAT_DATE_TIME

Format date and time (access via ACMEKCV$CB_FORMAT_DATE_TIME).

Formats a date or time string.

## Format

ACME$CB_FORMAT_DATE_TIME wqe, [dt_value], dt_string, [dt_len], [flags]

### Valid from ACME Callouts

All ACME callout routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**dt_value**

| | |
|---|---|
| OpenVMS usage: | UTC_DATE_TIME |
| type: | utcblk |
| access: | read only |
| mechanism: | reference |
| mechanism: | optional |

Address of UTC date/time value

**dt_string**

| | |
|---|---|
| OpenVMS usage: | CHAR_STRING |
| type: | character string |
| access: | write only |
| mechanism: | string descriptor |

Address of descriptor describing buffer to receive the formatted date/time string

**dt_len**

| | |
|---|---|
| OpenVMS usage: | WORD_UNSIGNED |
| type: | unsigned word |
| access: | write only |
| mechanism: | reference |
| mechanism: | optional |

Address of word to receive the length (in bytes) of the formatted date/time string

**flags**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | read only |
| mechanism: | value |
| mechanism: | optional |

Formatting control flags

## Description

This ACME callback routine invokes SYS$ASCUTC on behalf of your ACME agent. You should use this ACME callback routine rather than calling SYS$ASCUTC directly, because in the future this ACME callback routine may provide localization services to match the user's preferences.

### Related Callbacks

None.

### Alternative Callbacks

ACME$CB_SEND_LOGFILE

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Date/time conversion successful. |
| ACME$_BUFFEROVF | Output did not all fit into buffer provided. |
| other | Any condition returned by SYS$ASCUTC. |

# ACME$CB_ISSUE_CREDENTIALS

Issue credentials for the client (access via ACMEKCV$CB_ISSUE_ CREDENTIALS).

Provides credentials to be transmitted back to the ACM client process.

## Format

ACME$CB_ISSUE_CREDENTIALS wqe, type, credentials

### Valid from ACME Callouts

ACME$CO_CREDENTIALS

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**type**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | read only |
| mechanism: | reference |

Ignored. The credential type is derived from the credential information that was registered for the ACME agent.

**credentials**

| | |
|---|---|
| OpenVMS usage: | VECTOR_BYTE_UNSIGNED |
| type: | byte array |
| access: | read only |
| mechanism: | string descriptor |

Address of descriptor describing security credentials

## Description

If your ACME agent provides credentials for use in a matching persona extension you have implemented (as described in Chapter 10, use this ACME callback routine during ACME$CO_CREDENTIALS to transmit those credentials back to the ACM client process.

Within the ACM client process, the SYS$ACM[W] system service uses persona services to attach the credentials to the persona it is creating for the caller. The byte array you provide to this ACME callback routine must be entirely self-contained and must not make any references to particular memory addresses, since it will be used in a different process context than where it was created.

ACME$CB_ISSUE_CREDENTIALS( ) restricts the size of the credentials to a maximum of 8192 and returns ACME$_INVPARAMETER for any larger specified size.

### Related Callbacks

ACME$CB_SET_LOGON_FLAG
ACME$CB_SET_LOGON_STATS_DOI
ACME$CB_SET_OUTPUT_ITEM

### Alternative Callbacks

None.

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Credentials have been accepted. |
| ACME$_DUPCREDTYP | Credentials of the specified type have already been issued. |
| ACME$_INVCREDTYP | Calling agent is not registered to issue credentials. |
| ACME$_INVPARAMETER | Invalid credentials type or data descriptor. |
| ACME$_UNSUPPORTED | Unsupported execution context. |
| other | Any failure condition from allocating memory. |
| other | Any failure condition from LIB$ANALYZE_SDESC_64. |

# ACME$CB_QUEUE_DIALOGUE

Queue dialogue to be sent to the client (access via ACMEKCV$CB_QUEUE_DIALOGUE).

Queues dialogue material for transmission back to the ACM client process.

## Format

ACME$CB_QUEUE_DIALOGUE wqe, [flags], [item_code], [max_length], [data_1], [data_2]

### Valid from ACME Callouts

Authenticate Principal and Change Password routines

### Related Codes You Can Return

ACME$_PERFORMDIALOGUE

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**flags**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | read only |
| mechanism: | value |
| mechanism: | optional |

Dialogue control flags

**item_code**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | read only |
| mechanism: | value |
| mechanism: | optional |

Item code to use to tag response

**max_length**
OpenVMS usage: LONGWORD_UNSIGNED
type: unsigned longword
access: read only
mechanism: value
mechanism: optional

Maximum length of response data

**data_1**
OpenVMS usage: VECTOR_BYTE_UNSIGNED
type: byte array
access: read only
mechanism: string descriptor
mechanism: optional

Prompt/message text

**data_2**
OpenVMS usage: VECTOR_BYTE_UNSIGNED
type: byte array
access: read only
mechanism: string descriptor
mechanism: optional

Prompt/response/message text

## Description

This ACME callback routine creates an itemset entry in the itemset that the SYS$ACM[W] system service presents to the ACM client process within the ACM communications buffer. Note the following:

- The WQE parameter is the same WQE parameter received as an input parameter by the ACME callout routine from which the ACME callback routine is invoked.
- The Flags parameter contains the dialogue flags required to support this request. Currently defined flags are the following:

  ACMEDLOGFLG$V_INPUT
  ACMEDLOGFLG$V_NOECHO

  If the client has not specified support for at least the corresponding flags (as indicated in ACMEWQE$L_DIALOGUE_FLAGS), then the call fails with code ACME$_INSFDIALSUPPORT.

  A side effect of the Flags parameter is that it controls whether this itemset entry is for input or output.
- The item code parameter indicates the item code that the ACM client process should use to respond to an input itemset entry.

  For both input itemset entries and output itemset entries, the item code parameter groups the output with adjacent entries for display by the client. The exact effect of this grouping depends on the nature of the client.
- The max_length parameter indicates for an input itemset entry the maximum length of the input provided by the ACM client process.

- In the case of an output itemset entry, the max_length parameter becomes the msg_type parameter for specifying the nature of the output.

  These msg_type or message category values exist to classify the nature of output data that might be displayed or acted upon by ACM cilent programs. For binary output provided in data_1 or data_2, this value is the only way for an ACM client program to know the nature of the data[1] it receives in an itemset entry. For text output provided in data_1 or data_2 independent of the ACM client program, the value should be one from the common definitions.

  The common definitions for all function codes are the following:

| Symbol | Meaning |
|---|---|
| acmemc$k_general | General text |
| acmemc$k_header | Header text |
| acmemc$k_trailer | Trailer text |
| acmemc$k_selection | Acceptable choices |
| acmemc$k_dialogue_alert | Alert (advisory) |

  The common definitions specific to Authenticate Principal are the following:

| Symbol | Meaning |
|---|---|
| acmemc$k_system_ identification | System identification text |
| acmemc$k_system_notices | System notices |
| acmemc$k_welcome_notices | Welcome notices, |
| acmemc$k_logon_notices | Logon notices |
| acmemc$k_password_notices | Password notices |
| acmemc$k_mail_notices | MAIL notices |

  These definitions are documented in the *OpenVMS System Services Reference Manual* description of the SYS$ACM[W] system service.

  As with item codes, ACME-specific message category values are those with the topmost of the 16 bits set to 1.

- The data_1 parameter is a string descriptor, where null address with null length means “no string” and non-zero address with null length means “blank line”, if that is meaningful in the context of the ACM client program.

  In the case of an output itemset entry, the string is to be output; whereas in the case of an input itemset entry, the string is the "prompt" to be provided to the user, if appropriate.

- The data_2 parameter is a string descriptor, where null address with null length means “no string” and non-zero address with null length meaning “blank line”, if that is meaningful in the context of the ACM client program.

[1] Your ACME agent should only provide such binary data to an ACM client program known to be able to handle that particular category of data.

In the case of an output itemset entry, the string is to be output; whereas in the case of an input itemset entry, the meaning of the data_2 parameter depends on the setting of the ACMEDLOGFLG$V_NOECHO flag:

If ACMEDLOGFLG$V_NOECHO is clear, the string indicates the "default" that the ACME agent presumes, if a null string is returned. If ACMEDLOGFLG$V_NOECHO is set, the string indicates the "prompt" to be used for a second confirming entry of the unechoed data.

When the ACM client program is programmed to match your ACME agent, you can use data_1 and data_2 to send binary information rather than text, through the use of special msg_type values or special item codes.

Your ACME agent must always supply item code information when calling ACME callback routine ACME$CB_QUEUE_DIALOGUE, even for output itemset entries when no input can be provided. This is because the ACM client program uses the item code in the itemset entry to determine whether output data is in text form (bit ACMEIC$V_UCS set) or not, and thus how to present it to a user.

### Related Callbacks

ACME$CB_CANCEL_DIALOGUE

### Alternative Callbacks

None.

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Dialogue entry has been queued. |
| ACME$_DIALOGFULL | Pending dialogue queue is full. |
| ACME$_ INSFDIALSUPPORT | Dialogue not possible; context buffer parameter was not specified or requested capability exceeds that indicated by client. |
| ACME$_INVPARAMETER | Invalid item code or data descriptor. |
| ACME$_UNSUPPORTED | Unsupported execution context. |
| other | Any failure condition from allocating memory. |
| other | Any failure condition from LIB$ANALYZE_ SDESC_64. |

# ACME$CB_RELEASE_ACME_AST

Release an ACME-wide AST context (access via ACMEKCV$CB_RELEASE_ACME_AST).

Frees a 64-bit AST context for use between diverse requests, indicating the operating system will not deliver the AST after all.

## Format

ACME$CB_RELEASE_ACME_AST wqe, ast_context

### Valid from ACME Callouts

All ACME callout routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**ast_context**

| | |
|---|---|
| OpenVMS usage: | QUADWORD_UNSIGNED |
| type: | unsigned quadword |
| access: | read only |
| mechanism: | reference |

Address of quadword containing the AST context to be released

## Description

You can use this ACME callback routine during any ACME callout routine to indicate an AST context allocated with ACME$CB_ACQUIRE_ACME_AST is not needed after all since the operating system will not deliver the AST. If you need to make this callback, you must do so at least by the end of the next invocation of ACME$CO_AGENT_SHUTDOWN.

### Related Callbacks

ACME$CB_ACQUIRE_ACME_AST

### Alternative Callbacks

ACME$CB_RELEASE_ACME_RMSAST
ACME$CB_RELEASE_WQE_AST
ACME$CB_RELEASE_WQE_RMSAST

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Specified context has been deleted. |
| ACME$_ASTCTXNOTFND | Specified context was not found. |

# ACME$CB_RELEASE_ACME_RMSAST

Release an ACME-wide RMS AST context (access via ACMEKCV$CB_RELEASE_ACME_RMSAST).

Frees a 32-bit AST context for use within a single request, indicating the operating system will not deliver the AST after all.

## Format

ACME$CB_RELEASE_ACME_RMSAST wqe, ast_context

### Valid from ACME Callouts

All ACME callout routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**ast_context**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | read only |
| mechanism: | reference |

Address of longword containing the AST context to be released
AST parameter that would be passed to RMS services using this AST context

## Description

You can use this ACME callback routine during any ACME callout routine to indicate that an AST context allocated with ACME$CB_ACQUIRE_ACME_RMSAST is not needed after all since the operating system will not deliver the AST. If you need to make this callback, you must do so at least by the end of the next invocation of ACME$CO_AGENT_SHUTDOWN.

### Related Callbacks

ACME$CB_ACQUIRE_ACME_RMSAST

### Alternative Callbacks

ACME$CB_RELEASE_ACME_AST
ACME$CB_RELEASE_WQE_AST
ACME$CB_RELEASE_WQE_RMSAST

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Specified context has been deleted. |
| ACME$_ASTCTXNOTFND | Specified context was not found. |

# ACME$CB_RELEASE_RESOURCE

Release an ACME-specific resource (access via ACMEKCV$CB_RELEASE_RESOURCE).

Feeds an ACME-specific resource to the ACME server main image.

## Format

ACME$CB_RELEASE_RESOURCE wqe, resource_type, resource_value

### Valid from ACME Callouts

All ACME callout routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**resource_type**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | read only |
| mechanism: | value |

(ACME defined) type of the ACME-specific resource being released

**resource_value**

| | |
|---|---|
| OpenVMS usage: | QUADWORD_UNSIGNED |
| type: | unsigned quadword |
| access: | read only |
| mechanism: | reference |

Address of quadword containing the value of the ACME-specific resource allocated

## Description

You can use this ACME callback routine during any ACME callout routine to pass an ACME- specific resource to the ACME server main image for storage.

The type of ACME-specific resources cached with ACM cannot be zero (null).

### Related Callbacks

ACME$CB_ACQUIRE_RESOURCE

### Alternative Callbacks

None.

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Resource has been added to the list. |
| ACME$_NULLVALUE | Resource type of NULL was specified and is invalid. |

# ACME$CB_RELEASE_WQE_AST

Release a request-specific AST context (access via ACMEKCV$CB_RELEASE_WQE_AST).

Frees a 64-bit AST context for use within a single request, indicating the operating system will not deliver the AST after all.

## Format

ACME$CB_RELEASE_WQE_AST wqe, ast_context

### Valid from ACME Callouts

All ACME callout routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**ast_context**

| | |
|---|---|
| OpenVMS usage: | QUADWORD_UNSIGNED |
| type: | unsigned quadword |
| access: | read only |
| mechanism: | reference |

Address of quadword containing the AST context to be released

## Description

You can use this ACME callback routine during ACME callout routines associated with the same request from which the AST context was allocated with ACME$CB_ACQUIRE_WQE_AST. Making this call indicates that the AST context is not needed after all since the operating system will not deliver the AST. If you need to make this callback, you must do so at least by the end of the next invocation of ACME$CO_FINISH.

### Related Callbacks

ACME$CB_ACQUIRE_WQE_AST

### Alternative Callbacks

ACME$CB_RELEASE_ACME_AST
ACME$CB_RELEASE_ACME_RMSAST
ACME$CB_RELEASE_WQE_RMSAST

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Specified context has been deleted. |
| ACME$_UNSUPPORTED | Unsupported execution context. |
| ACME$_INCONSTATE | AST for specified context has already arrived. |
| ACME$_ASTCTXNOTFND | Specified context was not found. |

# ACME$CB_RELEASE_WQE_RMSAST

Release a request-specific RMS AST context (access via ACMEKCV$CB_ RELEASE_WQE_RMSAST).

Frees a 32-bit AST context for use within a single request, indicating the operating system will not deliver the AST after all.

## Format

ACME$CB_RELEASE_WQE_RMSAST wqe, ast_context

### Valid from ACME Callouts

All ACME callout routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**ast_context**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | read only |
| mechanism: | reference |

Address of longword containing the RMS AST context to be released

## Description

You can use this ACME callback routine during ACME callout routines associated with the same request from which the AST context was allocated with ACME$CB_ACQUIRE_WQE_RMSAST. Making this call indicates that the AST context is not needed after all since the operating system will not deliver the AST. If you need to make this callback, you must do so at least by the end of the next invocation of ACME$CO_FINISH.

### Related Callbacks

ACME$CB_ACQUIRE_WQE_RMSAST

### Alternative Callbacks

ACME$CB_RELEASE_ACME_AST
ACME$CB_RELEASE_ACME_RMSAST
ACME$CB_RELEASE_WQE_AST

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Specified context has been deleted. |
| ACME$_UNSUPPORTED | Unsupported execution context. |
| ACME$_INCONSTATE | AST for specified context has already arrived. |
| ACME$_ASTCTXNOTFND | Specified context was not found. |

# ACME$CB_REPORT_ACTIVITY

Report ACME activity (access via ACMEKCV$CB_REPORT_ACTIVITY).

Provides a text string to the ACME server main image describing the current status of your ACME agent for display by the command SHOW SERVER ACME/FULL.

## Format

ACME$CB_REPORT_ACTIVITY wqe, activity

### Valid from ACME Callouts

All ACME callout routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**activity**

| | |
|---|---|
| OpenVMS usage: | CHAR_STRING |
| type: | character string |
| access: | read only |
| mechanism: | string descriptor |

Address of ACME activity (status information) string descriptor

## Description

Your ACME agent can provide the text string from any ACME callout routine, but if you choose to provide it from a request processing callout routine you should not do so *too often*, since calling this routine causes synchronization between all simultaneous requests.

One way to avoid those synchronization bottlenecks is to compare the proposed text with the last text provided and not make the call if the change is insignificant or nonexistent.

### Related Callbacks

ACME$CB_REPORT_ATTRIBUTES

### Alternative Callbacks

None.

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Activity string has been stored. |
| ACME$_INVPARAMETER | Activity description was missing or null. |

# ACME$CB_REPORT_ATTRIBUTES

Report ACME identity and quota requirements (access via ACMEKCV$CB_ REPORT_ATTRIBUTES).

Provides the quota requirements for your ACME agent to the ACME server main image allowing calculation of the number of simultaneous requests that can be handled, the privileges that must be enabled, and so on.

It also provides a text string to the ACME server main image identifying your ACME agent for display by the command SHOW SERVER ACME/FULL.

## Format

ACME$CB_REPORT_ATTRIBUTES wqe, ident, resource_req

### Valid from ACME Callouts

ACME$CO_AGENT_INITIALIZE

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**ident**

| | |
|---|---|
| OpenVMS usage: | CHAR_STRING |
| type: | character string |
| access: | read only |
| mechanism: | string descriptor |

Address of descriptor specifying the identification string

**resource_req**

| | |
|---|---|
| OpenVMS usage: | ACME_RESOURCE_REQUIREMENTS |
| type: | acmersrc |
| access: | read only |
| mechanism: | reference |

Address of ACME resource requirements structure

## Description

The quota requirements you provide are used together with those from other ACME agents to calculate how many simultaneous requests can be processed, what privileges must be enabled, and so on.

### Related Callbacks

ACME$CB_REPORT_ACTIVITY

### Alternative Callbacks

None.

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Information has been stored. |
| ACME$_INVPARAMETER | Identity description was missing or null. |
| ACME$_UNSUPPORTED | Unsupported execution context. |
| ACME$_UNSUPREVLVL | Resource requirements structure level does not reflect a supported version. |

# ACME$CB_SEND_LOGFILE

Log a message in the ACME$SERVER log file (access via ACMEKCV$CB_SEND_LOGFILE).

Formats a message and writes it to the ACME$SERVER log file.

## Format

ACME$CB_SEND_LOGFILE wqe, msgvec, [actrtn], [actprm]

### Valid from ACME Callouts

All ACME callout routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**msgvec**

| | |
|---|---|
| OpenVMS usage: | VECTOR_LONGWORD_UNSIGNED |
| type: | unsigned longword array |
| access: | read only |
| mechanism: | reference |

Address of $PUTMSG style message vector

**actrtn**

| | |
|---|---|
| OpenVMS usage: | PROCEDURE |
| type: | procedure value |
| access: | read only |
| mechanism: | call without stack unwinding |
| mechanism: | optional |

Address of action routine

**actprm**

| | |
|---|---|
| OpenVMS usage: | USER_ARG |
| type: | unsigned quadword |
| access: | read only |
| mechanism: | value |

mechanism: optional

Parameter to pass to action routine

## Description

This ACME callback routine invokes system service SYS$PUTMSG on behalf of your ACME agent.

### Related Callbacks

ACME$CB_SEND_OPERATOR

### Alternative Callbacks

ACME$CB_FORMAT_DATE_TIME

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Message has been written. |
| other | Any failure condition from SYS$PUTMSG. |
| other | Any failure condition from LIB$CALLG. |

# ACME$CB_SEND_OPERATOR

Send a message to security operator terminals (access via ACMEKCV$CB_SEND_OPERATOR).

Sends a message to security operator terminals.

## Format

ACME$CB_SEND_OPERATOR wqe, msgtxt

### Valid from ACME Callouts

All ACME callout routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**msgtxt**

| | |
|---|---|
| OpenVMS usage: | CHAR_STRING |
| type: | character string |
| access: | read only |
| mechanism: | string descriptor |

Address of descriptor describing message text

## Description

This ACME callback routine sends the text string you provide to terminals enabled as security operator, via the OPCOM facility.

### Related Callbacks

ACME$CB_SEND_LOGFILE

### Alternative Callbacks

None.

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Message has been sent. |
| other | Any failure condition from LIB$ANALYZE_SDESC_64. |
| other | Any failure condition from SYS$SNDOPR. |

# ACME$CB_SET_2ND_STATUS

Record a secondary status for the request (access via ACMEKCV$CB_SET_2ND_STATUS).

Stores a longword into the secondary status for possible return to the ACM client process.

## Format

ACME$CB_SET_2ND_STATUS wqe, status_value

### Valid from ACME Callouts

All request processing routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**status_value**

| | |
|---|---|
| OpenVMS usage: | COND_VALUE |
| type: | longword |
| access: | read only |
| mechanism: | value |

Specific status value/condition code

## Description

This ACME callback returns the status you provide to the ACM client process as the secondary status. However, the SYS$ACM[W] system service returns the ACME$_AUTHFAILURE status code as the secondary status also, (not giving details of what went wrong) in cases where both of the following are true:

- Your ACME agent returns a primary status to the ACM dispatcher of ACME$_AUTHFAILURE
- The ACM client process called the SYS$ACM[W] system service without the security privilege.

However, in such cases the VMS ACME audits the secondary status your ACME agent provided so that security officers can review it.

### Related Callbacks

ACME$CB_SET_ACME_STATUS

### Alternative Callbacks

None.

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Secondary/protected status has been recorded. |
| ACME$_UNSUPPORTED | Unsupported execution context. |

# ACME$CB_SET_ACME_STATUS

Record an ACME-specific status for the request (access via ACMEKCV$CB_SET_ACME_STATUS).

Stores a longword into the ACME-specific status for possible return to the ACM client process.

## Format

ACME$CB_SET_ACME_STATUS wqe, status_value

### Valid from ACME Callouts

All request processing routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**status_value**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | read only |
| mechanism: | value |

ACME specific status value/condition code

## Description

This ACME callback returns the status you provide to the ACM client process as the ACME status.

### Related Callbacks

ACME$CB_SET_ACME_STATUS

### Alternative Callbacks

None.

## Return Values

| | |
|---|---|
| ACME$_NORMAL | ACME specific status has been recorded. |
| ACME$_UNSUPPORTED | Unsupported execution context. |

# ACME$CB_SET_DESIGNATED_DOI

Assume responsibility for handling a request (access via ACMEKCV$CB_SET_DESIGNATED_DOI).

Declares that your ACME agent takes responsibility for handling the current request.

## Format

ACME$CB_SET_DESIGNATED_DOI wqe

### Valid from ACME Callouts

Authenticate Principal and Change Password routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

## Description

This ACME callback routine denotes your ACME agent as controlling the Domain of Interpretation for this authentication.

### Related Callbacks

None.

### Alternative Callbacks

None.

## Return Values

| | |
|---|---|
| ACME$_NORMAL | WQE is tagged. |
| ACME$_UNSUPPORTED | Unsupported execution context. |

# ACME$CB_SET_LOGON_FLAG

Set a flag to report authentication activity to the client (access via ACMEKCV$CB_SET_LOGON_FLAG).

Sets one of a series of flags that the SYS$ACM[W] system service returns to the ACM client process on successful authentication.

## Format

ACME$CB_SET_LOGON_FLAG wqe, flag

### Valid from ACME Callouts

Authenticate Principal and Change Password routines (only for Authenticate Principal)

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

### wqe

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

### flag

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | read only |
| mechanism: | value |

Flag number to set

## Description

This ACME callback routine sets one of the flags in Section A.4, indicating a particular condition was encountered on this authentication. The symbolic names available for the FLAG parameter are:

- ACMELGIFLG$K_NEW_MAIL_AT_LOGIN
- ACMELGIFLG$K_PASSWORD_CHANGED
- ACMELGIFLG$K_PASSWORD_EXPIRED
- ACMELGIFLG$K_PASSWORD_WARNING

- ACMELGIFLG$K_PASSWORD2_CHANGED
- ACMELGIFLG$K_PASSWORD2_EXPIRED
- ACMELGIFLG$K_PASSWORD2_WARNING

#### Related Callbacks

ACME$CB_ISSUE_CREDENTIALS
ACME$CB_SET_LOGON_STATS_DOI
ACME$CB_SET_OUTPUT_ITEM

#### Alternative Callbacks

None.

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Specified flag has been set. |
| ACME$_INVFLAG | Invalid flag number. |
| ACME$_UNSUPPORTED | Unsupported execution context. |

# ACME$CB_SET_LOGON_STATS_DOI

Set non-native (non-OpenVMS) logon statistics (access via ACMEKCV$CB_SET_LOGON_STATS_DOI).

Provides authentication statistics to the ACM client process from the ACME agent that controls the Domain of Interpretation (the one that called ACME$CB_SET_DESIGNATED_DOI).

## Format

```
ACME$CB_SET_LOGON_STATS_DOI  wqe, logon_data
```

### Valid from ACME Callouts

Authenticate Principal and Change Password routines (only for Authenticate Principal)

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**logon_data**

| | |
|---|---|
| OpenVMS usage: | ACM_LOGON_INFORMATION_DOI |
| type: | acmelidoi |
| access: | read only |
| mechanism: | reference |

Address of DOI logon statistics buffer

## Description

If your ACME agent is the one that controls the Domain of Interpretation (the one that called ACME$CB_SET_DESIGNATED_DOI for a particular authentication, use this ACME callback routine to specify the contents of the DOI portion of the data returned to the ACM client process in response to the ACME$_LOGON_INFORMATION item code (as specified in Section A.5).

While the ACM dispatcher fills in the ACMELIDIO$L_ACME_ID and ACMELIDIO$L_PHASE fields, your ACME agent must fill in all other fields, including ACMELIDIO$W_SIZE and ACMELIDIO$W_REVISION_LEVEL.

### Related Callbacks

ACME$CB_ISSUE_CREDENTIALS
ACME$CB_SET_LOGON_FLAG
ACME$CB_SET_OUTPUT_ITEM

### Alternative Callbacks

None.

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Logon statistics have been recorded in the WQE. |
| ACME$_UNSUPPORTED | Unsupported execution context. |
| ACME$_UNSUPREVLVL | Logon statistics structure level does not reflect a supported version. |

# ACME$CB_SET_LOGON_STATS_VMS

Set native (OpenVMS) logon statistics (access via ACMEKCV$CB_SET_LOGON_STATS_VMS).

Provides authentication statistics to the ACM client process from the VMS ACME.

## Format

ACME$CB_SET_LOGON_STATS_VMS wqe, logon_data

### Valid from ACME Callouts

Authenticate Principal and Change Password routines (only for Authenticate Principal)

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

### wqe

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

### logon_data

| | |
|---|---|
| OpenVMS usage: | ACM_LOGON_INFORMATION_VMS |
| type: | acmelivms |
| access: | read only |
| mechanism: | reference |

Address of VMS logon statistics buffer

## Description

The VMS ACME uses this ACME callback routine to specify the contents of the VMS portion of the data returned to the ACM client process in response to the ACME$_LOGON_INFORMATION item code (as specified in Section A.6).

While the ACM dispatcher fills in the ACMELIVMS$L_ACME_ID and ACMELIVMS$L_PHASE fields, the VMS ACME must fill in all other fields, including ACMELIVMS$W_SIZE and ACMELIVMS$W_REVISION_LEVEL.

### Related Callbacks

None.

### Alternative Callbacks

None.

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Information has been recorded in the WQE. |
| ACME$_UNSUPPORTED | Unsupported execution context. |
| ACME$_UNSUPREVLVL | Logon statistics structure level does not reflect a supported version. |

# ACME$CB_SET_OUTPUT_ITEM

Provide data to fulfill a client output item (access via ACMEKCV$CB_SET_OUTPUT_ITEM).

Provides results for ACME-specific output items requested by the ACM client process.

## Format

ACME$CB_SET_OUTPUT_ITEM wqe, entry, data

### Valid from ACME Callouts

All request processing routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

### wqe

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

### entry

| | |
|---|---|
| OpenVMS usage: | ITEM_LIST |
| type: | ile3 |
| access: | read only |
| mechanism: | reference |

Address of the item list entry to set

### data

| | |
|---|---|
| OpenVMS usage: | VECTOR_BYTE_UNSIGNED |
| type: | unsigned byte array |
| access: | read only |
| mechanism: | string descriptor |

Address of descriptor describing data for field

## Description

Your ACME agent should call this ACME callback routine with the address of the original item list entry being fulfilled, as well as a string descriptor of the data to fulfill that output item.

If the string descriptor you provide contains a non-zero length but a zero address, the ACME server main image allocates memory for the response but does not fill it. This can be used for cases where you want to use data directly as it comes from some other service. After you call this ACME callback routine, the item list output item data buffer (described in Section B.3) addressed by the ILE3$PS_ BUFADDR field of the output item list entry contains the amount of space you specified. You must ensure that the ACMEOUTITM$B_DATA field is filled with the data and the ACMEOUTITM$W_LENGTH field is filled with the length actually used before request processing completes.

### Related Callbacks

ACME$CB_ISSUE_CREDENTIALS
ACME$CB_SET_LOGON_FLAG
ACME$CB_SET_LOGON_STATS_DOI

### Alternative Callbacks

None.

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Parameter field has been set. |
| ACME$_NOTOUTITEM | Item list entry does not reflect an output item code.. |
| ACME$_INVPARAMETER | Data provided is longer than allowed by the client item. |
| other | Any failure condition from allocating memory. |
| other | Any failure condition from LIB$ANALYZE_ SDESC_64. |

# ACME$CB_SET_PHASE_EVENT

Set phase notification event (access via ACMEKCV$CB_SET_PHASE_EVENT).

Used to synchronize activity between a tightly coupled ACME agent and ACM client program. It is used by the VMS ACME with LOGINOUT to support the traditional LGI callout interface interface.

## Format

ACME$CB_SET_PHASE_EVENT wqe, [event_data]

### Valid from ACME Callouts

Authenticate Principal and Change Password routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**event_data**

| | |
|---|---|
| OpenVMS usage: | VECTOR_BYTE_UNSIGNED |
| type: | unsigned byte array |
| access: | read only |
| mechanism: | string descriptor |
| mechanism: | optional |

Prompt/message text

## Description

If an ACME agent invokes this ACME callback routine, the SYS$ACM[W] system service notifies the ACM client process after each authentication or password change phase.

### Related Callbacks

None.

### Alternative Callbacks

None.

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Notification has been queued. |
| ACME$_INVREQUEST | Notification was previously queued. |
| ACME$_ INSFDIALSUPPORT | Client cannot handle dialogue. |
| ACME$_UNSUPPORTED | Unsupported execution context. |
| other | Any failure condition from allocating memory. |
| other | Any failure condition from LIB$ANALYZE_ SDESC_64. |

# ACME$CB_SET_WQE_FLAG

Set a WQE flag to communicate with the ACME server main image and with other ACME agents (access via ACMEKCV$CB_SET_WQE_FLAG).

Sets one of a group of flags to communicate with the ACME server main image and with other ACME agents.

## Format

ACME$CB_SET_WQE_FLAG wqe, flag

### Valid from ACME Callouts

All request processing routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

### wqe

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

### flag

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | read only |
| mechanism: | value |

Flag number to set

## Description

The flags used for this communication are defined in Section B.13. ACME agents can read the flags directly, but to set them ACME agents must invoke this ACME callback routine.

### Related Callbacks

None.

### Alternative Callbacks

None.

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Specified flag has been set. |
| ACME$_INVFLAG | Invalid flag number. |
| ACME$_UNSUPPORTED | Unsupported execution context. |

# ACME$CB_SET_WQE_PARAMETER

Set a WQE parameter to communicate with the ACME server main image and with other ACME agents (access via ACMEKCV$CB_SET_WQE_PARAMETER).

Sets one of a group of string values to communicate with the ACME server main image and with other ACME agents.

## Format

ACME$CB_SET_WQE_PARAMETER wqe, id, data

### Valid from ACME Callouts

All request processing routines

### Related Codes You Can Return

None.

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**wqe**

| | |
|---|---|
| OpenVMS usage: | ACM_WORK_QUEUE_ENTRY |
| type: | acmewqe |
| access: | modify |
| mechanism: | reference |

Address of the work queue entry

**id**

| | |
|---|---|
| OpenVMS usage: | LONGWORD_UNSIGNED |
| type: | unsigned longword |
| access: | read only |
| mechanism: | value |

ID number of parameter (item) field to set

**data**

| | |
|---|---|
| OpenVMS usage: | VECTOR_BYTE_UNSIGNED |
| type: | unsigned byte array |
| access: | read only |
| mechanism: | string descriptor |

Address of descriptor specifying string for the field

## Description

The possible ID values are the following:

- ACMEWQE$K_SYSTEM_PASSWORD
- ACMEWQE$K_PRINCIPAL_NAME
- ACMEWQE$K_PRINCIPAL_NAME_OUT
- ACMEWQE$K_VMS_USERNAME
- ACMEWQE$K_PASSWORD_1
- ACMEWQE$K_PASSWORD_2
- ACMEWQE$K_NEW_PASSWORD_1
- ACMEWQE$K_NEW_PASSWORD_2

ACME agents can read the strings directly, but to set them ACME agents must invoke this ACME callback routine.

### Related Callbacks

None.

### Alternative Callbacks

None.

## Return Values

| | |
|---|---|
| ACME$_NORMAL | Parameter field has been set. |
| ACME$_INVFLAG | Invalid flag number. |
| ACME$_UNSUPPORTED | Unsupported execution or function context. |
| other | Any failure condition from allocating memory. |
| other | Any failure condition from LIB$ANALYZE_SDESC_64. |

# 10
# Persona Extensions Overview

This chapter discusses the use of custom persona extensions for implementing specialized security policies. The initial provision of persona extensions is most easily handled from an ACME agent. The programming techniques for writing your own persona extension are different from those for writing ACME agents and are described in this chapter, Chapter 11, and Chapter 12.

After a client process has authenticated with the SYS$ACM[W] system service, you may want the client process to retain DOI-specific information so it can be retrieved or used later on during the life of that process or other processes that it creates.

The role of a persona extension is to store the credentials information provided by a particular ACME agent. When a process calls the SYS$ACM[W] system service it can specify item code ACME$_PERSONA_HANDLE_OUT and appropriate function modifiers to indicate it wants certain persona services. The SYS$ACM[W] system service fills the longword described by the ACME$_ PERSONA_HANDLE_OUT item with a persona ID. That persona ID represents a persona that contains credentials from all ACME agents that chose to supply them.

The ultimate example of this is the persona (with accompanying persona extensions) created when LOGINOUT calls the SYS$ACM[W] system service. That persona is installed as the natural persona and persists for the life of the process.

Suitably privileged programs can create personas using other system services, but the SYS$ACM[W] system service is the most comprehensive and foolproof mechanism.

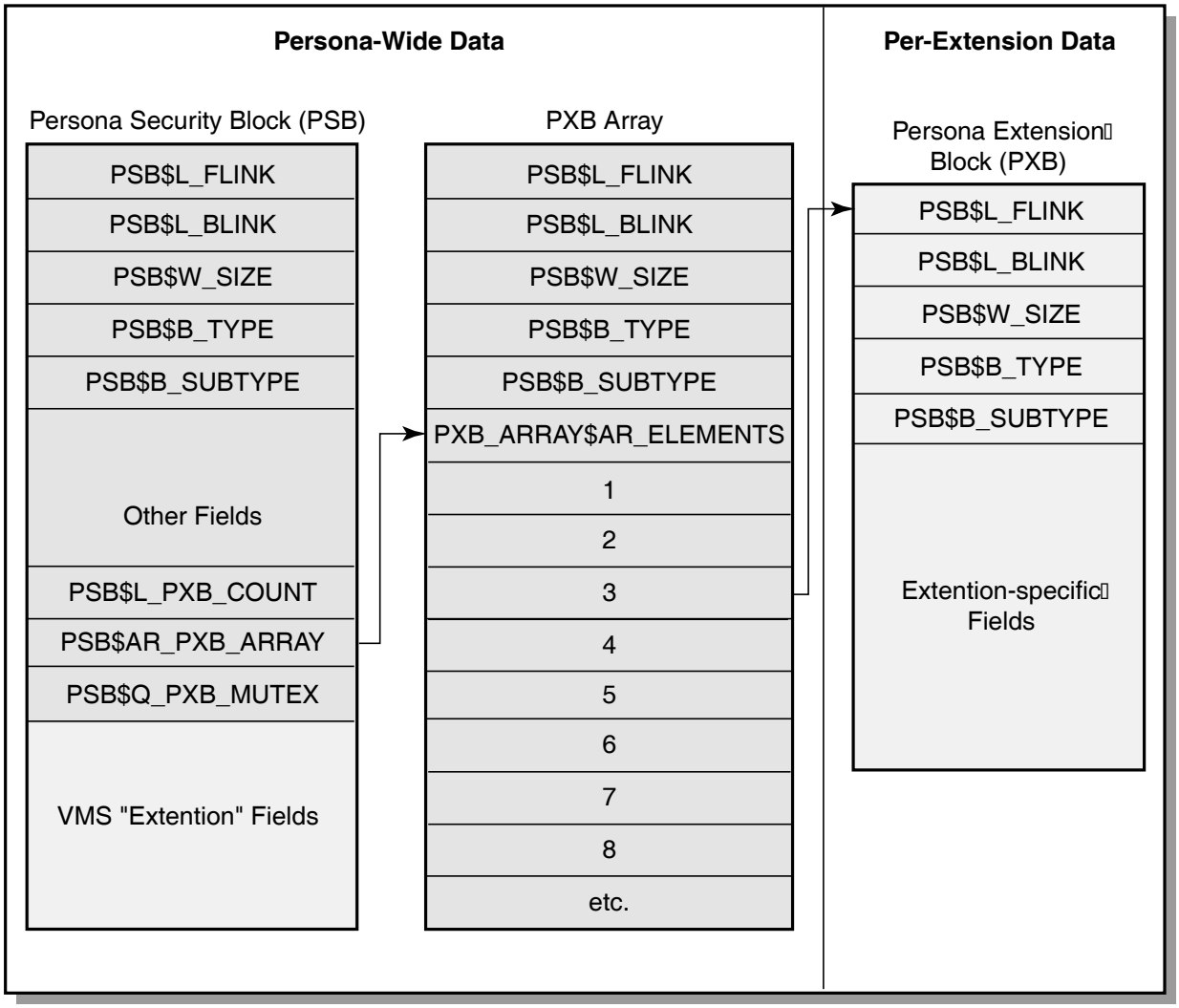
To add support for a custom persona extension you must implement a **persona extension image**, providing entry points discussed in Chapter 11. The entry points in your persona extension image are always called in kernel mode. Although it may be possible to write a persona extension image in some other language, the only programming language currently supported for this purpose is BLISS.

For information on an example persona extension image, see Appendix D.

## 10.1 Persona Data Structures

The data structures used in persona processing are provided in Appendix C. Three are listed here that are of primary interest to those who are writing a persona extension image. These data structures are all stored in system nonpaged pool.

**Figure 10–1 Some Persona Data Structures**




VM-0786A-AI

## 10.1.1 Persona Security Block (PSB)

The **Persona Security Block** (shown in Section C.1) is the manifestation of a single persona held by a process. User programs refer to a persona by a persona ID or in most cases, just default to using the natural persona. In addition to general fields for persona management, the PSB concludes with the fields that implement the VMS “persona extension”. Since every persona must always have a VMS persona extension, data for that persona extension is bundled right into the PSB.

### 10.1.2 PXB_ARRAY

Within the Persona Security Block (PSB) cell PSB$AR_PXB_ARRAY points to an element within a PXB_ARRAY structure (shown in Section C.2). In particular, it points to cell PXB_ARRAY$AR_ELEMENTS, the start of an array of longwords indexed by the **extension ID** for the various persona extensions available on a system. The prior cells of the PXB_ARRAY are only for memory allocation tracking purposes, and your persona extension image does not need to refer to them.

Element 0 of the PXB_ARRAY is special. While non-zero contents in the other elements are pointers to a **Persona Extension Block (PXB)**, non-zero contents in Element 0 indicate the extension ID for the persona extension that is the primary extension for the subject persona services. The primary extension will be targeted for any subsequent SYS$ACM[W] system service call that specifies modifier ACME$M_DEFAULT_PRINCIPAL.

The size of the PXB_ARRAY is enough to hold the maximum number of persona extensions that might be present on a system. On a typical system most of the elements will contain zero (the null pointer).

### 10.1.3 Persona Extension Block (PXB)

An indexed entry within the PXB_ARRAY points to the Persona Extension Block (PXB) (if any) for the subject persona implementing the persona extension corresponding to that index number. The fact that a persona extension image for a particular extension ID has been declared on a system does not necessarily mean that all personas on the system have that persona extension. Some personas might have been created before that persona extension image was registered, while others may not have been provided with credentials for that persona extension for reasons private to a particular ACME agent.

The description in Section C.3 gives the format of the header of a PXB. Data after the header is in a format specific to the persona extension.

### 10.1.4 Persona Extension Cloning and Delegation

In certain cases there is a need to create an additional persona identical to an existing one. The general persona code in the OpenVMS executive handles the PSB and PXB_ARRAY in this case, but your persona extension image must handle the PXB. The following two types of calls are made to your persona extension image for this purpose:

- Clone (for a new persona in the same process)
- Delegate (for a new persona in a different process on the same system)

For each of those calls, your persona extension image can use either of the following two techniques to propagate the PXB information to the new persona:

- Allocate a new PXB and copy the data into it. If some of the data consists of pointers to other data structures, you must decide whether to duplicate those as well or support reference counts within those structures and just copy the pointers.
- Increment a reference count in the existing PXB and provide the address of that PXB for use by the new persona.

For the clone call, reference counting activity is synchronized by the fact that persona system services operate under the protection of the *inner mode semaphore*. Since personas are specific to a single process, there will only be one stream of activity calling your persona extension image for a particular persona at a time. Thus, two calls to your persona extension image entry points for a given PXB are always handled sequentially, rather than simultaneously, Two modify calls, or a modify call and a delete call, will not conflict with one another.

For the delegate call, a reference-counting approach is not protected by the *inner mode semaphore*, since that is a per-process interlock and delegation of a persona involves two processes. To engage in reference counting, rather than straight copying, in a delegation situation requires that your persona extension image provide its own interlocking. One approach is to use a mutex you have created within your PXB in the same fashion you use the PSB mutex PSB$Q_PXB_MUTEX to protect manipulation of the entries in the PXB_ARRAY data structure (described in Section 11.6).

Given the complexity of a reference counting approach, why would the writer of a persona extension image ever choose that method? The answer is related to the cost and relative frequency of clone and delegate calls to your persona extension image, as follows:

- Cost

  If your persona extension only takes 24 bytes (in addition to the 12 header bytes for a PXB), the additional overhead in memory usage is minimal. Some persona extensions, however, could take a kilobyte or more to store public key certificates, for example.

- Frequency

  The OpenVMS executive calls your clone routine every time a user makes an explicit call to system service SYS$PERSONA_CLONE specifying a persona that has your persona extension. But it will also make such a call when passing control information to the Magtape ACP or when RMS must stall an operation and retain persona information as it was at the time the request was made.

  The OpenVMS executive calls your delegate routine every time a user makes an explicit call to the SYS$PERSONA_DELEGATE system service specifying a persona that has your persona extension. But it will also make such a call when the user calls the SYS$CREPRC system service to create a new process running under the same UIC.

Whether the added risk of a reference counting implementation provides sufficient performance improvement to make it worthwhile is something only you can decide.

## 10.2 Persona Item Codes

There are several ranges of item codes:

- OpenVMS item codes

  The possible range is from 1 to 1023, with currently defined codes identified by the symbol range ISS$_MIN_ITEM_CODE through ISS$_MAX_ITEM_CODE.

- Common item codes

These are separately implemented for each persona extension. The possible range is from 1024 to 8191, with currently defined codes identified by the symbol range ISS$_MIN_COMMON_ITEM_CODE through ISS$_MAX_COMMON_ITEM_CODE.

Modify support for certain codes is mandatory, as indicated in Section 11.6.

Query support for certain codes is mandatory, as indicated in Section 11.7.

- Extension-specific item codes

  These are separately defined for each persona extension. The possible range is from 8192 to 65535. Currently defined codes vary according to persona extension. This is where you should define any item codes specific to your persona extension. It is expected that your codes will overlap values used by other persona extensions.

# 11
# Persona Extensions Entry Points

Your persona extension image can supply any of the entry points described below. Five of them are required (as noted in Section 12.5). Note the following rules:

- Code for all persona extension routines (except the initialization routine) must be in the Psect EXEC$NONPAGED_CODE, declared with the BLISS macro $DECLARE_PSECT from LIB.REQ.
- Data for all routines must be in the Psect EXEC$NONPAGED_DATA, declared with the BLISS macro $DECLARE_PSECT from LIB.REQ.
- These entry points need not be globally visible to the linker, since their addresses are passed to the OpenVMS executive via other mechanisms.
- The names of these entry points do not matter.

## 11.1 Initialization Routine

This required persona extension routine declares the addresses of other persona extension routines by calling NSA$REGISTER_PSB_EXTENSION, as described in Chapter 12.

Code for this routine must be in the Psect EXEC$INIT_CODE, declared with the BLISS macro $DECLARE_PSECT from LIB.REQ.

You must use the BLISS macro $INITIALIZATION_ROUTINE, specifying the name of your initialization routine, to indicate that your initialization routine should be invoked when your persona extension image is loaded into memory.

This routine accepts no arguments.

## 11.2 Create Routine

This required persona extension routine creates a new persona extension using a specified set of credentials. It accepts the following arguments:

1. PSB - address of the PSB to which the new PXB will be attached
2. PXB - address of a longword into which the address of the new PXB should be stored
3. TLV - address of the byte string containing credentials for the persona extension
4. TLV_SIZE - length of the octet string containing credentials for the persona extension

Allocate space for your PXB from nonpaged pool and populate it, including the following three header fields:

- PXB$B_TYPE - DYN$C_SECURITY
- PXB$B_SUBTYPE - DYN$C_SECURITY_PXB_GENERIC

- PXB$W_SIZE - size allocated

For the size field you must use the size actually allocated (which can be larger than the size requested, due to rounding).

## 11.3 Clone Routine

This persona extension routine is called when copying an existing persona extension for use by the same process, such as in support of the SYS$PERSONA_CLONE system service. It accepts the following arguments:

1. PSB - address of the PSB to which this PXB is attached
2. PXB - address of the PXB for this persona extension
3. NEW_PXB - address of a longword into which the address of the new PXB should be stored

If your persona extension image supplies this persona extension routine, it can either duplicate the existing PXB and any supporting structures or it can store the address of the existing PXB while maintaining a reference count within it.

For copying extremely simple PXBs, it is sufficient to copy the contents of the original PXB into a NEW_PXB. For more complex cases, some cells in the PXB will be pointers to other data structures that must be duplicated as well.

If your persona extension image does not support this operation, cloned personas will not contain your persona extension.

## 11.4 Delegate Routine

This persona extension routine is called when copying an existing persona extension for use by a different process, such as in support of the SYS$PERSONA_DELEGATE system service. It accepts the following arguments:

1. PSB - address of the PSB to which this PXB is attached
2. PXB - address of the PXB for this persona extension
3. unused
4. NEW_PXB - address of a longword into which the address of the new PXB should be stored
5. NEW_PSB - address of the PSB to which the new PXB will be attached

If your persona extension image supplies this persona extension routine, it can either duplicate the existing PXB and any supporting structures, or it can return the same value as the existing PXB while maintaining a reference count within it.

**Note**

While you can depend on the inner mode semaphore to protect against multiple simultaneous access to a PXB shared through cloning, that does *not* work for the case of a PXB shared through delegation and you will need your own synchronization code if you take that approach.

For copying extremely simple PXBs, it is sufficient to copy the contents of the original PXB into a NEW_PXB. For more complex cases, some cells in the PXB will be pointers to other data structures that must be cloned as well.

If your persona extension image does not support this operation, delegated personas will not contain your persona extension.

## 11.5 Delete Routine

This required persona extension routine is called to perform a logical deletion of a persona extension, typically in response to the system service SYS$PERSONA_DELETE or SYS$PERSONA_EXTENSION_DELETE. It accepts the following arguments:

1. PSB - address of the PSB to which this PXB is attached
2. PXB - address of the PXB for this persona extension

Whether your persona extension routine always deletes the PXB, or first decrements a reference count and only deletes when all references are gone, depends on the method you used for the clone and delegate operations.

## 11.6 Modify Routine

This persona extension routine is called to modify a value stored in a persona extension typically in response to the system service SYS$$PERSONA_MODIFY. It accepts the following arguments:

1. PSB - address of the PSB to which this PXB is attached
2. PXB - address of the PXB for this persona extension
3. ITEMCODE - the ITEM_CODE to be modified
4. BUF_ADDRESS - address of the new value
5. BUF_LENGTH - length of the new value

Each persona extension image is required to support the ISS$_DOI item code in its modify routine.

If the ITEMCODE is inappropriate, your persona extension routine should return the error code SS$_BADITMCOD. If the BUF_LENGTH cannot be supported for that ITEMCODEI, it should return the error code SS$_BADBUFLEN.

Whether your persona extension image copies persona extensions, or reference counts them, determines whether a modification affects both the original and the cloned (or delegated) persona. Your decision regarding how to implement clone and delegate operations affects the security policy achieved by your persona extension.

The persona extension's modify routine must support the ISS$_DOI item code unless the extension's DOI field was already loaded by the ACME agent which created the extension's contents and provided it to SYS$ACM using the ACME$CB_ISSUE_CREDENTIALS callback.

ISS$_DOI represents a quadword integer whose low-order longword contains the identifier of the creating ACME agent. The high-order longword is ignored.

**Note**

Do not be concerned with the policy effect of your decision on the required modify support for ISS$_DOI. That code is only modified at the time that the SYS$ACM[W] system service adds a persona extension to a persona, before any clone or delegate operation can take place.

If your persona extension image reference counts a cloned or delegated persona extension and the reference count is greater than 1, it is not possible to expand the size of a persona extension.

Otherwise, if your persona extension can be of variable size, and it is necessary to increase the size to handle this particular modification, take the following steps:

1. Allocate and populate a replacement PXB.
2. Use system service SYS$PERSONA_EXTENSION_LOOKUP to determine the persona extension ID (index) for your persona extension.
3. Increment the reference count on the PSB with the BLISS macro NSA$REFERENCE_PSB.
4. Return failure SS$_NOSUCHID if that index is greater than cell PSB$L_ PXB_COUNT in the PSB (decrementing the reference count and deallocating the new PXB on the way out).
5. Lock the PSB extension array with the BLISS macro SCH$LOCKW_QUAD.
6. Read any existing entry for this persona extension ID (index) from the array pointed to by cell PSB$AR_PXB_ARRAY in the PSB and deallocate that block if non-null.
7. Store the new PXB in that array location.
8. Unlock the PSB extension array with the BLISS macro SCH$UNLOCK_ QUAD.
9. Decrement the reference count with the BLISS macro NSA$DEREFERENCE_ PSB.
10. Use that new PXB address for further references to the PXB during this call of your persona extension routine.

If you have the Source Listings Kit, an example of that technique can be found in facility SYS module NT_EXTENSION.

## 11.7 Query Routine

This required persona extension routine is called to retrieve contents of a persona extension, typically in response to the system service SYS$PERSONA_QUERY or SYS$PERSONA_EXTENSION_DELETE. It supports the following arguments:

1. PSB - address of the PSB to which this PXB is attached
2. PXB - address of the PXB for this persona extension
3. ITEMCODE - the ITEM_CODE to be retrieved
4. BUF_ADDRESS - address of the comparison value or output buffer
5. BUF_LENGTH - length of the comparison value or output buffer
6. RETURN_LENGTH - address of a 16-bit variable to receive the length
7. QUERYFLAGS - low-order bit means compare to a value, rather than query
8. INPUT_DATA - address of a descriptor for input data

Each persona extension image is required to support the following item codes in its Query routine:

- ISS$_COMMON_FLAGS

- ISS$_DOI
- ISS$_COMMON_USERNAME
- ISS$_DOMAIN
- ISS$_COMMON_PRINCIPAL
- ISS$_COMMON_ACCOUNT
- ISS$_EXTENSION

For an unsupported item code, return SS$_BADITMCOD. Use this persona extension routine (depending on the value of the parameter QUERYFLAGS) either to retrieve a value or to compare to an existing value. Returns True (1) or False (0).

If the INPUT_DATA descriptor address is non-zero, it describes a string to be used as input for deriving output data. This could be used, for example, when your persona extension contained a signature key that could be freely used to sign input data on behalf of the process, but which was not to be divulged to non-privileged callers (guarding against a Trojan horse attack to obtain the key).

Your Persona Extension Block (PXB) is specially treated by the process dump capabilities on OpenVMS Alpha to avoid disclosing information to unprivileged users, since even their own data might be compromised if they ran a Trojan Horse program.

## 11.8 Make_TLV Routine

This persona extension routine must be supplied, but your persona extension image is free to return the code SS$_UNSUPPORTED. It supports the following arguments:

1. PSB - address of the PSB to which this PXB is attached
2. PXB - address of the PXB for this persona extension
3. Itemcode - ISS$_MAKE_TLV
4. BUF_ADDRESS - address of the output buffer
5. BUF_LENGTH - length of the output buffer
6. RETURN_LENGTH - address of a 16-bit variable to receive the length
7. FLAGS - for future use

This routine packages all the information from your persona extension into a position- independent string suitable for feeding into your create routine when a batch job is started. Whether your security model can or should support having a persona extension in a batch job derived from the persona extension in effect when the batch job was submitted is for you to decide.

# 12

# Connecting Your Persona Extension Image to the OpenVMS Executive

## 12.1 Compiling

Ensure your code uses the Psects specified in Chapter 11.

## 12.2 Linking

To access required support routines, your persona extension image must be linked against the OpenVMS Executive. On OpenVMS Alpha use the /SYSEXE qualifier.

Additionally, you must link in module SYS$DOINIT from library VMS$VOLATILE_PRIVATE_INTERFACES so your initialize routine will be called.

You must specify linker options for the program sections EXEC$NONPAGED_CODE, EXEC$NONPAGED_DATA and EXEC$INIT_CODE, as shown in the example of a persona extension provided with the ACME software distribution kit. For more information, see Appendix D.

## 12.3 Testing

Test your completed persona extension image with the SYS$ETC:CHECK_SECTIONS.COM command procedure.

## 12.4 Installing

Resolve - common system disks, PCSI action routines, SYSMAN, etc.

To set up your persona extension image to be included when the system boots, use a command in the folowing format:

```
$ MCR SYSMAN
SYSMAN> SYS_LOADABLE ADD product image
```

Then, run the following command procedure:

```
$ @SYS$UPDATE:VMS$SYSTEM_IMAGES.COM
```

## 12.5 Declaring Your Persona Extension Image

The initialize routine of your persona extension image must call the routine NSA$REGISTER_PSB_EXTENSION, described on the following pages.

# NSA$REGISTER_PSB_EXTENSION

Declare a set of persona extension routines.

This routine adds your persona extension to the set of valid persona extensions during the current boot of the system.

## Format

NSA$REGISTER_PSB_EXTENSION extension_name, pxdv

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | integer |
| access: | read only |
| mechanism: | by value in R0 |

## Arguments

**extension_name**

| | |
|---|---|
| OpenVMS usage: | CHAR_STRING |
| type: | character string |
| access: | read only |
| mechanism: | string descriptor |

Name of the persona extension.

**pxdv**

| | |
|---|---|
| OpenVMS usage: | PXDV_TYPE |
| type: | pxdv |
| access: | modify |
| mechanism: | reference |

Address of persona security extension dispatch vector block.

## Description

The executive image initialization routine for your persona extension image should allocate a Persona Security Extension Dispatch Vector Block of size PXDV$S_PXDV and populate it with the following:

- PXDV$L_FLAGS
- PXDV$A_CREATE - CREATE routine address[1]
- PXDV$A_CLONE - CLONE routine address
- PXDV$A_DELEGATE - DELEGATE routine address
- PXDV$A_DELETE - DELETE routine address[1]
- PXDV$A_MODIFY - MODIFY routine address[1]
- PXDV$A_QUERY - QUERY routine address[1]

- PXDV$A_MAKE_TLV - MAKE_TLV routine address[2]

| In | Use |
|---|---|
| PXDV$L_VERSION | PXDV$K_VERSION[1] |
| PXDV$L_FLAGS | 0 |
| All unused cells | 0 |

[1]For this version of the structure.

Currently a maximum of 15 extensions can be registered.

## Return Values

| | |
|---|---|
| SS$_NORMAL | Extension has been registered. |
| SS$_BUFFEROVF | The limit on registered extensions has already been reached. |
| SS$_DUPLNAM | An extension by the specified name has already been registered. |
| SS$_INVARG | One of the required routine addresses was not supplied. |
| other | Failure allocating nonpaged pool. |

[1] This routine address must be supplied.

[2] This routine address must be supplied, although the routine itself can return SS$_UNSUPPORTED.

# A

# SYS$ACM[W] Data Structures

## A.1 ACM Communications Buffer (ACMECB)

not available

**acmecb**

| | | |
|---|---|---|
| acmecb$q_context_id | | 0 |
| acmecb$w_revision_level | acmecb$w_size | 8 |
| acmecb$l_acme_id | | 12 |
| acmecb$l_item_set_count | | 16 |
| acmecb$ps_item_set | | 20 |

**acmedlogflg**

**acmefc**

## A.2 ACM Hardware Address Type (ACMEHAT)

not available

**acmehat**

| acmehat$w_facility | acmehat$w_protocol | 0 |
|---|---|---|

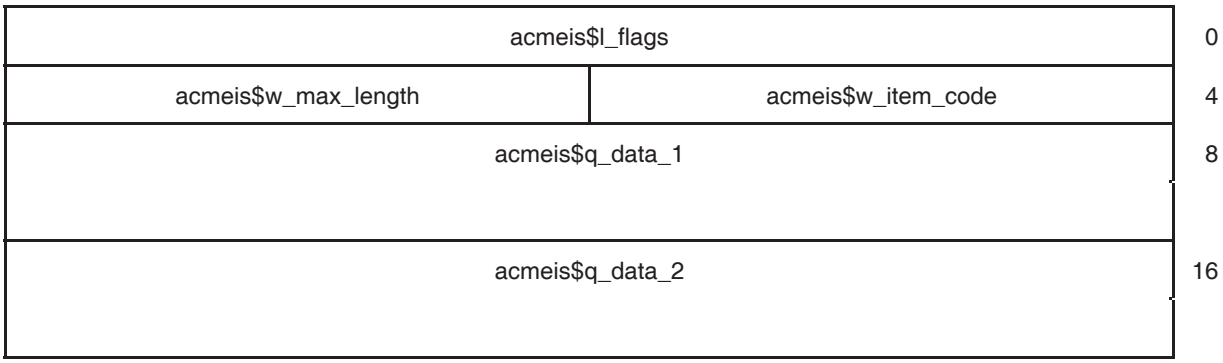
**acmeic**

**acmeid**

## A.3 ACME Item Set Entry (ACMEITMSET)

not available

**acmeitmset**



## A.4 ACME Logon Flags (ACMELGIFLG)

not available

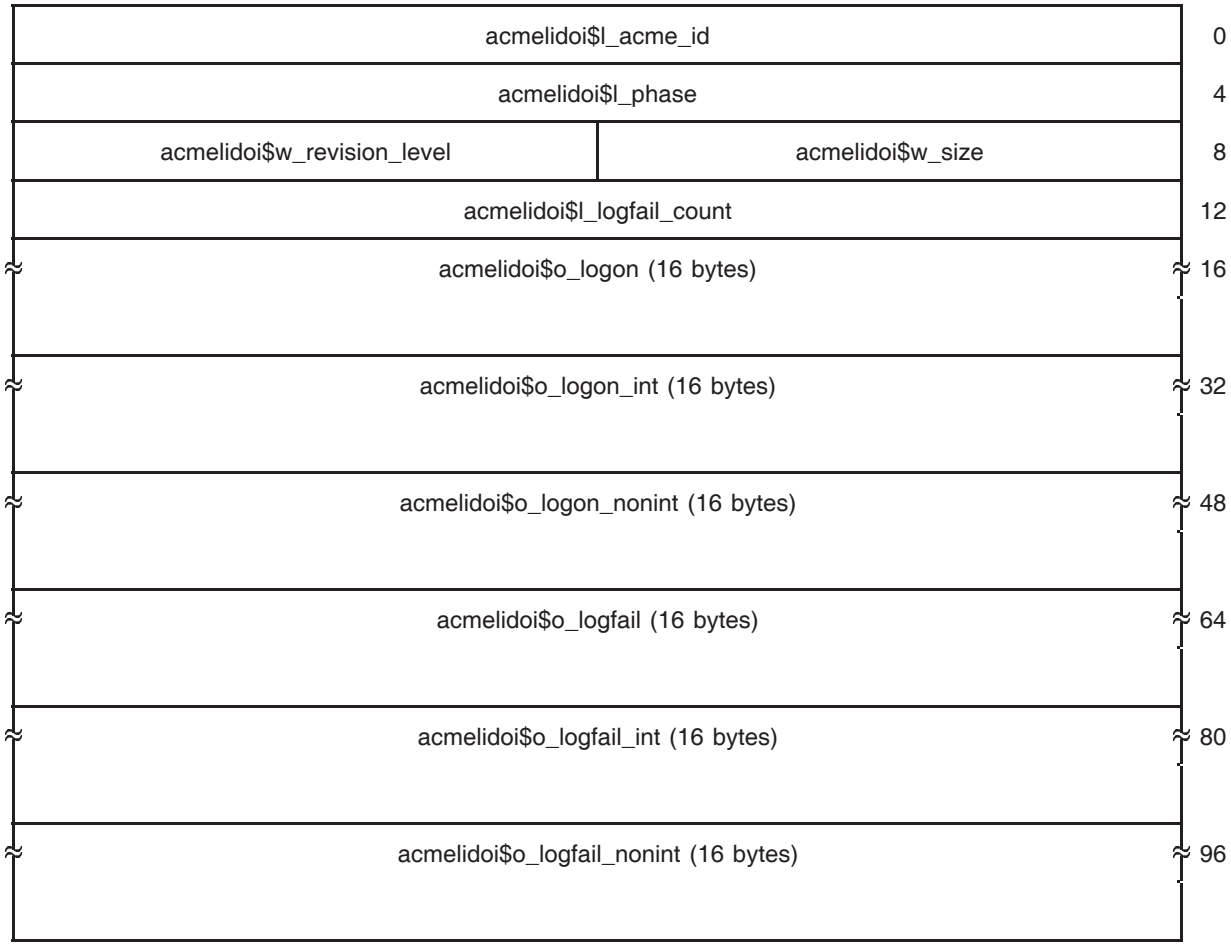
**acmelgiflg**

| | |
|---|---|
| acmelgiflg$l_logon_flags | 0 |

## A.5 ACME Logon Information for the Domain of Interpretation (ACMELIDOI)

not available

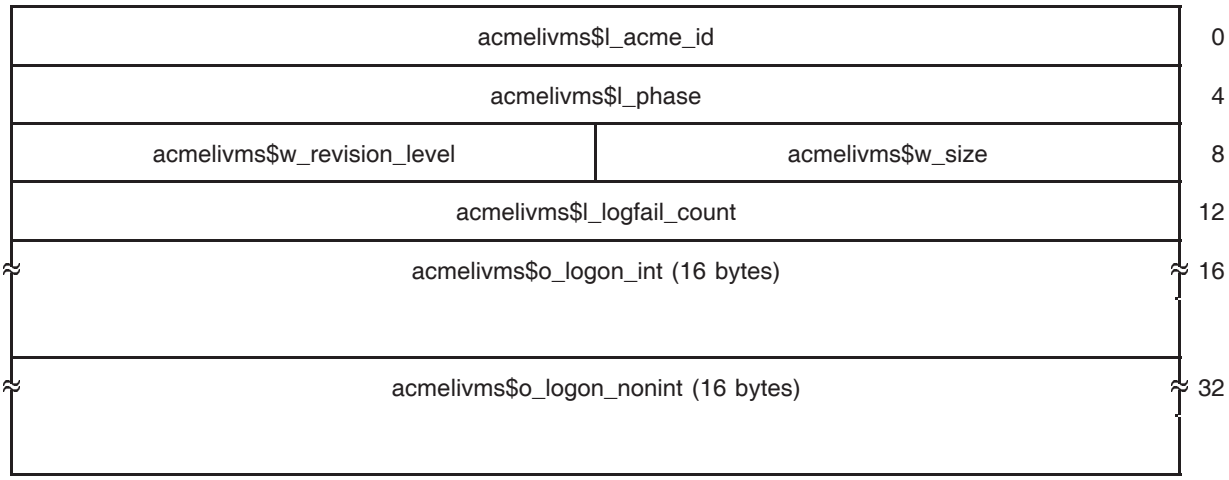
**acmelidoi**



## A.6 ACME Logon Information for VMS (ACMELIVMS)

not available

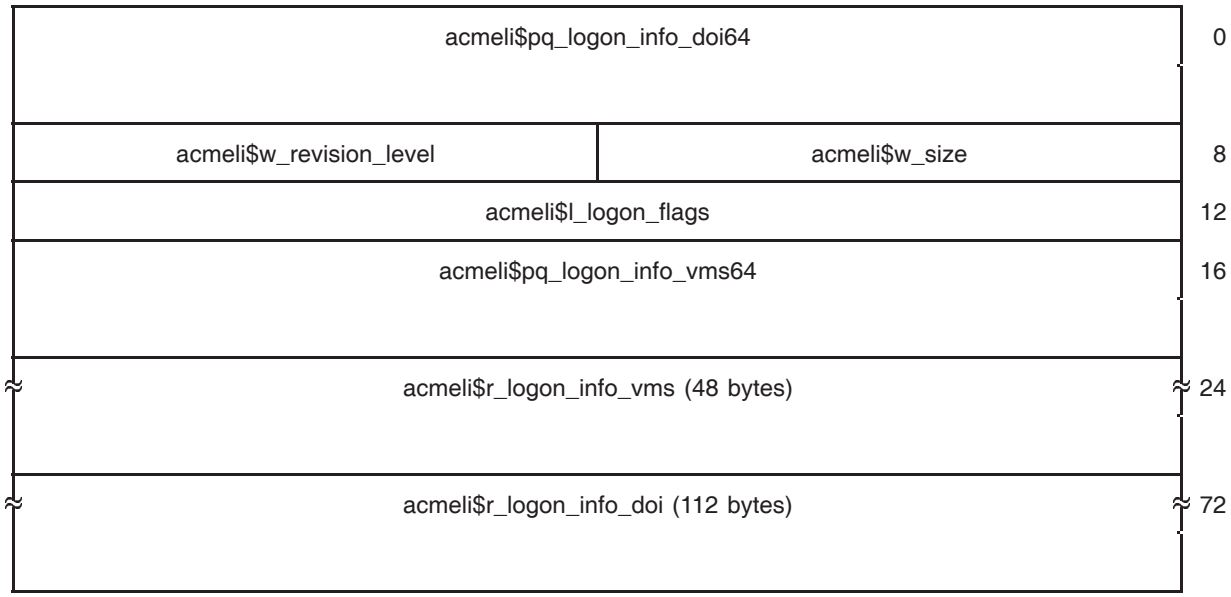
**acmelivms**



## A.7 ACME Logon Information (ACMELI)

not available

**acmeli**




**acmemc**

## A.8 ACME Authentication Mechanism (ACMEMECH)

not available

**acmemech**

| | | |
|---|---|---|
| acmemech$w_facility | acmemech$w_mechanism | 0 |

**acmepwdflg**

## A.9 ACME Revision Level (ACMEREVLVL)

not available

**acmerevlvl**

| |
|---|
| acmerevlvl$w_revision_level |

# A.10 ACM Status Block (ACMESB)

not available

**acmesb**

| | |
|---|---|
| acmesb$l_status | 0 |
| acmesb$l_secondary_status | 4 |
| acmesb$l_acme_id | 8 |
| acmesb$l_acme_status | 12 |

## A.11 Universal Coordinated Time (UTCBLK)

not available

**UTCBLK**

# B

# ACME Agent Interface Data Structures

## B.1 ACME Date Time Formatting Control Flags (ACMEDTFLG)

The following are the contents of the aggregate structure acmedtflg:

| Field | Use |
|---|---|
| acmedtflg$l_control_flags | Composite field |

**acmedtflg**

| | |
|---|---|
| acmedtflg$l_control_flags | 0 |

## B.2 ACME Kernel Callback Vector (ACMEKCV)

The following are the contents of the aggregate structure acmekcv:

| Field | Use |
|---|---|
| acmekcv$w_acm_revision_level | ACM kernel revision level |
| acmekcv$w_revision_level | Structure revision level |
| acmekcv$cb_report_attributes | Report resource requirements |
| acmekcv$cb_send_operator | Send a message to the operator |
| acmekcv$cb_send_logfile | Write a message in the log file |
| acmekcv$cb_allocate_acme_vm | Allocate a block of memory |
| acmekcv$cb_deallocate_acme_vm | Deallocate a block of memory |
| acmekcv$cb_allocate_wqe_vm | Allocate a block of memory |
| acmekcv$cb_deallocate_wqe_vm | Deallocate a block of memory |
| acmekcv$cb_set_designated_doi | Declare DOI accepting request |
| acmekcv$cb_set_2nd_status | Report secondary (protected) status |
| acmekcv$cb_set_acme_status | Report ACME specific status |
| acmekcv$cb_set_wqe_flag | Set WQE status/control flag |
| acmekcv$cb_set_wqe_parameter | Set WQE data item |
| acmekcv$cb_set_output_item | Set output item |
| acmekcv$cb_set_logon_flag | Set logon status flag |
| acmekcv$cb_set_logon_stats_vms | Report native (OpenVMS) logon statistics |
| acmekcv$cb_set_logon_stats_doi | Reprot non-native (non-OpenVMS) logon statistics |
| acmekcv$cb_set_phase_event | Set phase transition notification |
| acmekcv$cb_queue_dialogue | Queue a dialogue item set |
| acmekcv$cb_cancel_dialogue | Dismiss pending dialogue |
| acmekcv$cb_acquire_acme_ast | Establish a non-RMS AST context |
| acmekcv$cb_release_acme_ast | Dismiss non-RMS AST context |
| acmekcv$cb_acquire_wqe_ast | Establish a non-RMS AST context |
| acmekcv$cb_release_wqe_ast | Dismiss non-RMS AST context |
| acmekcv$cb_acquire_acme_rmsast | Establish an RMS AST context |
| acmekcv$cb_release_acme_rmsast | Dismiss RMS AST context |
| acmekcv$cb_acquire_wqe_rmsast | Establish an RMS AST context |
| acmekcv$cb_release_wqe_rmsast | Dismiss RMS AST context |
| acmekcv$cb_acquire_resource | Acquire an ACME specific resource |
| acmekcv$cb_release_resource | Release an ACME specific resource |
| acmekcv$cb_issue_credentials | Issue security credentials |
| acmekcv$cb_format_date_time | Format date and time |
| acmekcv$cb_report_activity | Report resource requirements |

The following cells are only meaningful if ACMEKCV$W_REVISION_LEVEL contains ACMEKCV$K_MAJOR_ID_001/ACMEKCV$K_MINOR_ID_001 or higher.

| Field | Use |
|---|---|
| acmekcv$cb_ucs_to_latin1 | Convert UCS2_4 to Latin1 |
| acmekcv$cb_latin1_to_ucs | Convert Latin1 to UCS2_4 |

The following constants are defined in conjunction with acmekcv:

| Constant | Value | Use |
|---|---|---|
| acmersrc$k_length | 80 | |
| acme$k_minor_id_000 | 0 | |
| acme$k_minor_id_001 | 0 | |
| acme$k_minor_id | 0 | The default is still 000 |
| acme$k_major_id_001 | 1 | |
| acme$k_major_id | 1 | |
| acme$k_revision | 256 | |
| acmekcv$k_minor_id_000 | 0 | original V7.2-1 callback list |
| acmekcv$k_minor_id_001 | 1 | supporting Latin1<->UCS conversion |
| acmekcv$k_minor_id | 0 | |
| acmekcv$k_major_id_001 | 1 | |
| acmekcv$k_major_id | 1 | |
| acmekcv$k_revision | 256 | |
| acme$k_report_attributes | 0 | |
| acme$k_send_operator | 1 | |
| acme$k_send_logfile | 2 | |
| acme$k_allocate_acme_vm | 3 | |
| acme$k_deallocate_acme_vm | 4 | |
| acme$k_allocate_wqe_vm | 5 | |
| acme$k_deallocate_wqe_vm | 6 | |
| acme$k_set_designated_doi | 7 | |
| acme$k_set_2nd_status | 8 | |
| acme$k_set_acme_status | 9 | |
| acme$k_set_wqe_flag | 10 | |
| acme$k_set_wqe_parameter | 11 | |
| acme$k_set_output_item | 12 | |
| acme$k_set_logon_flag | 13 | |
| acme$k_set_logon_stats_vms | 14 | |
| acme$k_set_logon_stats_doi | 15 | |
| acme$k_set_phase_event | 16 | |
| acme$k_queue_dialogue | 17 | |
| acme$k_cancel_dialogue | 18 | |
| acme$k_acquire_acme_ast | 19 | |

| Constant | Value | Use |
|---|---|---|
| acme$k_release_acme_ast | 20 | |
| acme$k_acquire_wqe_ast | 21 | |
| acme$k_release_wqe_ast | 22 | |
| acme$k_acquire_acme_rmsast | 23 | |
| acme$k_release_acme_rmsast | 24 | |
| acme$k_acquire_wqe_rmsast | 25 | |
| acme$k_release_wqe_rmsast | 26 | |
| acme$k_acquire_resource | 27 | |
| acme$k_release_resource | 28 | |
| acme$k_issue_credentials | 29 | |
| acme$k_format_date_time | 30 | |
| acme$k_report_activity | 31 | |
| acme$k_ucs_to_latin1 | 32 | |
| acme$k_latin1_to_ucs | 33 | |
| acme$k_kcv_count | 34 | |

## acmekcv

| | | |
|---|---|---|
| acmekcv$w_revision_level | acmekcv$w_acm_revision_level | 0 |
| acmekcv$cb_report_attributes | | 4 |
| acmekcv$cb_send_operator | | 8 |
| acmekcv$cb_send_logfile | | 12 |
| acmekcv$cb_allocate_acme_vm | | 16 |
| acmekcv$cb_deallocate_acme_vm | | 20 |
| acmekcv$cb_allocate_wqe_vm | | 24 |
| acmekcv$cb_deallocate_wqe_vm | | 28 |
| acmekcv$cb_set_designated_doi | | 32 |
| acmekcv$cb_set_2nd_status | | 36 |
| acmekcv$cb_set_acme_status | | 40 |
| acmekcv$cb_set_wqe_flag | | 44 |
| acmekcv$cb_set_wqe_parameter | | 48 |
| acmekcv$cb_set_output_item | | 52 |
| acmekcv$cb_set_logon_flag | | 56 |

(continued on next page)

| | |
|---|---|
| acmekcv$cb_set_logon_stats_vms | 60 |
| acmekcv$cb_set_logon_stats_doi | 64 |
| acmekcv$cb_set_phase_event | 68 |
| acmekcv$cb_queue_dialogue | 72 |
| acmekcv$cb_cancel_dialogue | 76 |
| acmekcv$cb_acquire_acme_ast | 80 |
| acmekcv$cb_release_acme_ast | 84 |
| acmekcv$cb_acquire_wqe_ast | 88 |
| acmekcv$cb_release_wqe_ast | 92 |
| acmekcv$cb_acquire_acme_rmsast | 96 |
| acmekcv$cb_release_acme_rmsast | 100 |
| acmekcv$cb_acquire_wqe_rmsast | 104 |
| acmekcv$cb_release_wqe_rmsast | 108 |
| acmekcv$cb_acquire_resource | 112 |
| acmekcv$cb_release_resource | 116 |
| acmekcv$cb_issue_credentials | 120 |
| acmekcv$cb_format_date_time | 124 |
| acmekcv$cb_report_activity | 128 |
| acmekcv$cb_ucs_to_latin1 | 132 |
| acmekcv$cb_latin1_to_ucs | 136 |

## B.3 Item List Output Item Data Buffer (ACMEOUTITM)

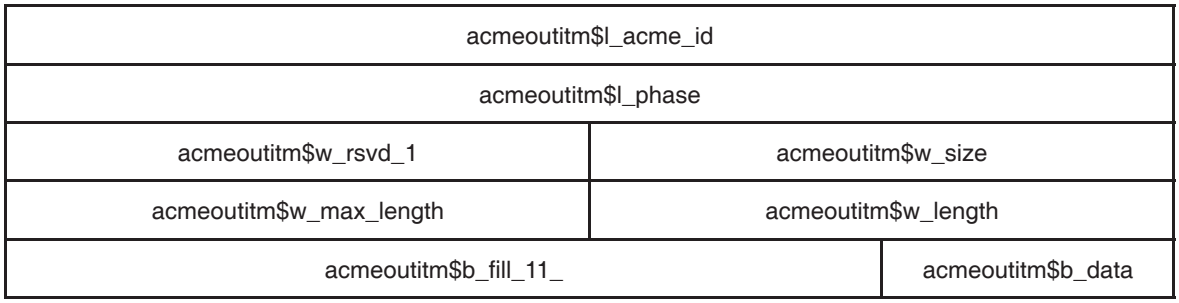
The following are the contents of the aggregate structure acmeoutitm:

| Field | Use |
|---|---|
| acmeoutitm$l_acme_id | ID of ACME which set the item entry |
| acmeoutitm$l_phase | Phase during which item was set |
| acmeoutitm$w_size | Structure size, in bytes |
| acmeoutitm$w_rsvd_1 | |
| acmeoutitm$w_length | Actual size, in bytes, of data |
| acmeoutitm$w_max_length | Size, in bytes, of data buffer |
| acmeoutitm$b_data | Data |
| acmeoutitm$b_fill_11_ | |

The following constants are defined in conjunction with acmeoutitm:

| Constant | Value | Use |
|---|---|---|
| acmewqefdx$k_length | 296 | |
| acmeoutitm$k_length | 16 | Length of fixed portion |

**acmeoutitm**



## B.4 ACME Process Quota Resource Requirements Block(ACMEPQ)

The following are the contents of the aggregate structure acmepq:

| Field | Use |
|---|---|
| acmepq$l_memory | Virtual address space use |
| acmepq$l_channel | I/O channels |
| acmepq$l_direct_io | Direct I/O count |
| acmepq$l_buffer_io | Buffered I/O count |
| acmepq$l_buffer_io_mem | Buffered I/O memory usage |
| acmepq$l_ast | AST count |
| acmepq$l_tqe | TQE count |
| acmepq$l_lock | Lock count |

**acmepq**

| | |
|---|---|
| acmepq$l_memory | 0 |
| acmepq$l_channel | 4 |
| acmepq$l_direct_io | 8 |
| acmepq$l_buffer_io | 12 |
| acmepq$l_buffer_io_mem | 16 |
| acmepq$l_ast | 20 |
| acmepq$l_tqe | 24 |
| acmepq$l_lock | 28 |

## B.5 ACME Agent Resource Requirements Block (ACMERSRC)

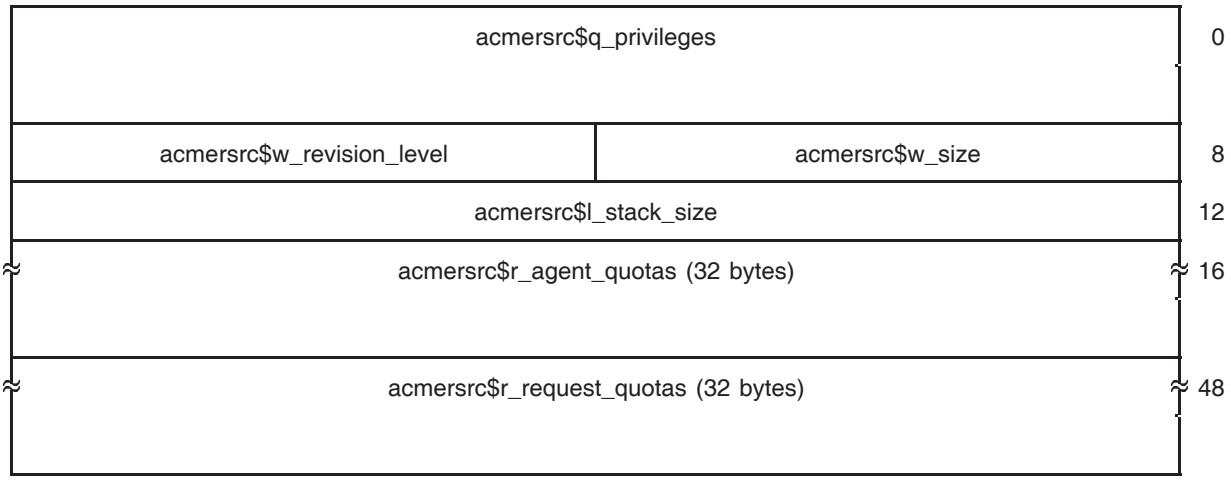
The following are the contents of the aggregate structure acmersrc:

| Field | Use |
|---|---|
| General resource requirements | |
| acmersrc$q_privileges | Operating privilege |
| acmersrc$w_size | Structure size, in bytes |
| acmersrc$w_revision_level | Structure revision level |
| acmersrc$l_stack_size | Maximum operating stack |
| acmersrc$r_agent_quotas | General process quotas |
| Per-request resource requirements | |
| acmersrc$r_request_quotas | Per-request process quotas |

The following constants are defined in conjunction with acmersrc:

| Constant | Value | Use |
|---|---|---|
| acmersrc$k_minor_id_000 | 0 | |
| acmersrc$k_minor_id | 0 | |
| acmersrc$k_major_id_001 | 1 | VMS V7.2-1 |
| acmersrc$k_major_id | 1 | |
| acmersrc$k_revision | 256 | |

**acmersrc**



## B.6 ACME WQE Extension for Agent Shutdown (ACMEWQEADX)

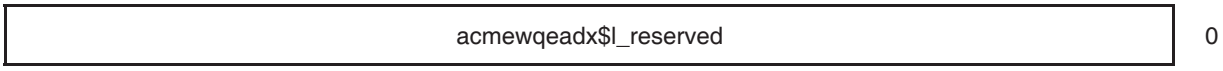
The following are the contents of the aggregate structure acmewqeadx:

| Field | Use |
|---|---|
| acmewqeadx$l_reserved | Null structure |

The following constants are defined in conjunction with acmewqeadx:

| Constant | Value | Use |
|---|---|---|
| acmewqeaex$k_length | 4 | |

### acmewqeadx



## B.7 ACME WQE Extension for Agent Startup (ACMEWQEAEX)

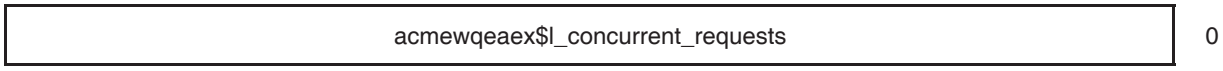
The following are the contents of the aggregate structure acmewqeaex:

| Field | Use |
|---|---|
| acmewqeaex$l_concurrent_requests | maximum at a time |

The following constants are defined in conjunction with acmewqeaex:

| Constant | Value | Use |
|---|---|---|
| acmewqeaix$k_length | 4 | |

**acmewqeaex**



## B.8 ACME WQE Extension for Agent Initialize (ACMEWQEAIX)

The following are the contents of the aggregate structure acmewqeaix:

| Field | Use |
|---|---|
| acmewqeaix$ps_agent_name | address of string descriptor |

The following constants are defined in conjunction with acmewqeaix:

| Constant | Value | Use |
|---|---|---|
| acmewqeax$k_length | 296 | |

**acmewqeaix**

## B.9 ACME WQE Extension for Agent Standby (ACMEWQEASX)

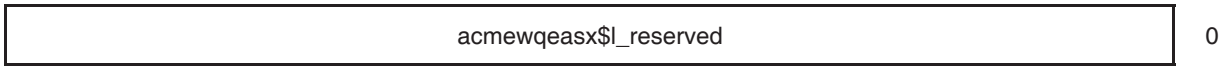
The following are the contents of the aggregate structure acmewqeasx:

| Field | Use |
|---|---|
| acmewqeasx$l_reserved | Null structure |

The following constants are defined in conjunction with acmewqeasx:

| Constant | Value | Use |
|---|---|---|
| acmewqeadx$k_length | 4 | |

**acmewqeasx**



## B.10 ACME WQE Extension for Authentication (ACMEWQEAX)

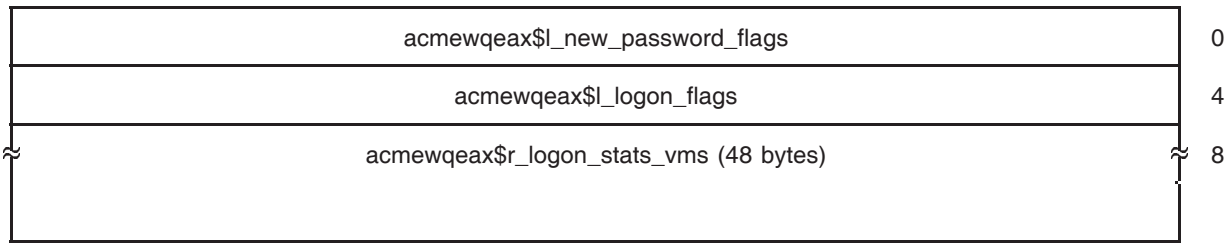
The following are the contents of the aggregate structure acmewqeax:

| Field | Use |
|---|---|
| acmewqeax$l_new_password_flags | Password change request flags |
| acmewqeax$l_logon_flags | Logon flags |
| acmewqeax$r_logon_stats_vms | Native (OpenVMS) logon statistics |
| acmewqeax$r_logon_stats_doi | Non-native (non-OpenVMS) logon statistics |
| acmewqeax$r_system_password | System password |
| acmewqeax$r_principal_name | Raw (unprocessed) principal name |
| acmewqeax$r_principal_name_out | Principal name |
| acmewqeax$r_vms_username | Mapped OpenVMS user name |
| acmewqeax$r_password_1 | Password 1 |
| acmewqeax$r_password_2 | Password 2 |
| acmewqeax$r_new_password_1 | New password 1 |
| acmewqeax$r_new_password_2 | New password 2 |

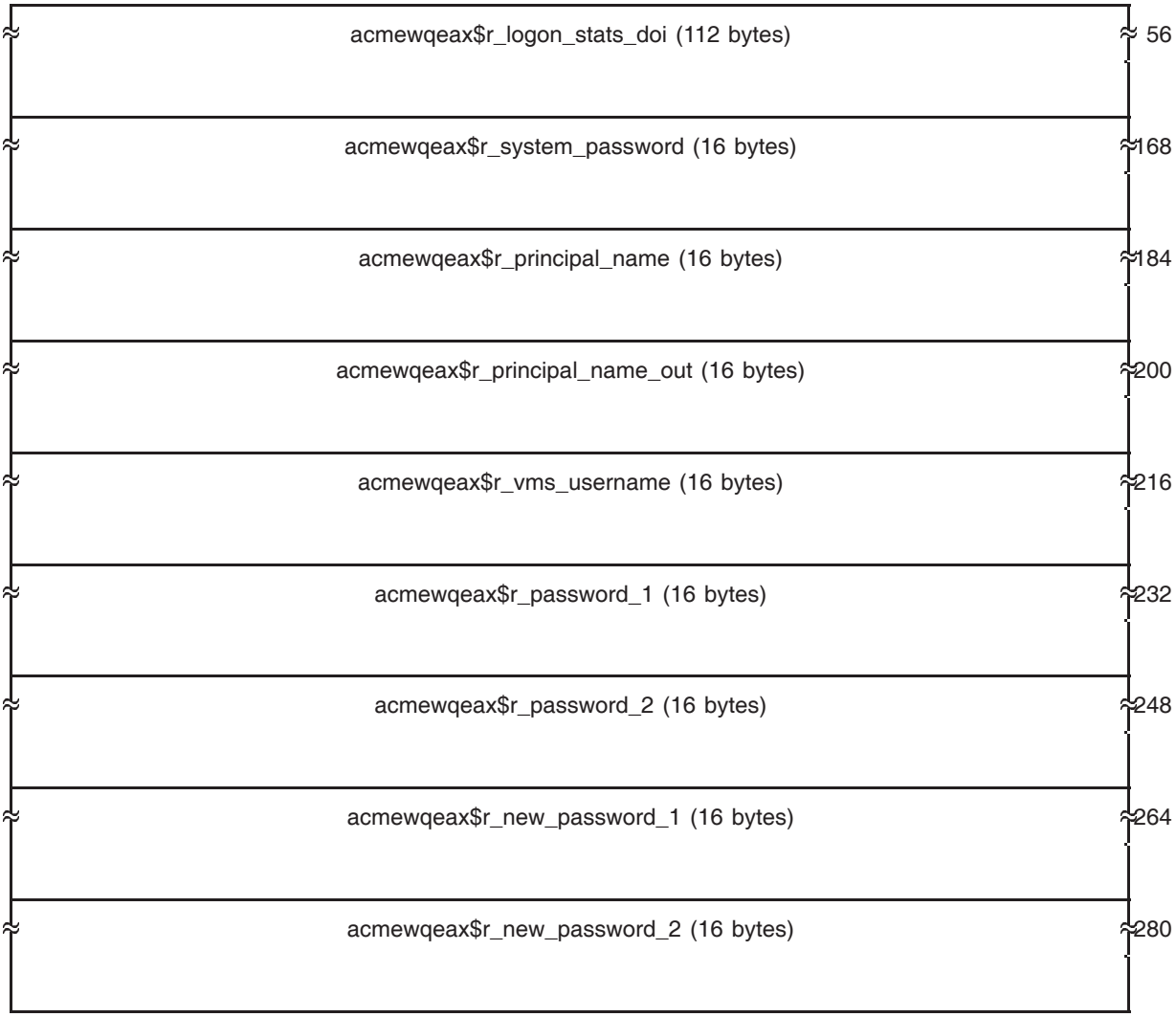
The following constants are defined in conjunction with acmewqeax:

| Constant | Value | Use |
|---|---|---|
| acmewqe$k_min_auth_param | 256 | |
| acmewqe$k_system_password | 256 | |
| acmewqe$k_principal_name | 257 | |
| acmewqe$k_principal_name_out | 258 | |
| acmewqe$k_vms_username | 259 | |
| acmewqe$k_password_1 | 260 | |
| acmewqe$k_password_2 | 261 | |
| acmewqe$k_new_password_1 | 262 | |
| acmewqe$k_new_password_2 | 263 | |
| acmewqe$k_max_auth_param | 263 | |

**acmewqeax**




(continued on next page)


acmewqeax$r_logon_stats_doi (112 bytes)
56
acmewqeax$r_system_password (16 bytes)
168
acmewqeax$r_principal_name (16 bytes)
184
acmewqeax$r_principal_name_out (16 bytes)
200
acmewqeax$r_vms_username (16 bytes)
216
acmewqeax$r_password_1 (16 bytes)
232
acmewqeax$r_password_2 (16 bytes)
248
acmewqeax$r_new_password_1 (16 bytes)
264
acmewqeax$r_new_password_2 (16 bytes)
280

## B.11 Work Queue Entry Function Dependent Extension (ACMEWQEFDX)

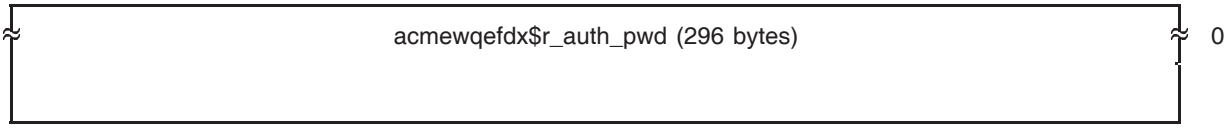
The following are the contents of the aggregate structure acmewqefdx:

| Field | Use |
|---|---|
| acmewqefdx$r_auth_pwd | AUTHENTICATE_ PRINCIPAL/CHANGE_ PASSWORD |

The following constants are defined in conjunction with acmewqefdx:

| Constant | Value | Use |
|---|---|---|
| acmewqeasx$k_length | 4 | |

**acmewqefdx**



## B.12 ACME Work Queue Entry Function Independent Extension (ACMEWQEFIX)

The following are the contents of the aggregate structure acmewqefix:

| Field | Use |
|---|---|
| acmewqefix$l_reserved | Null structure |

The following constants are defined in conjunction with acmewqefix:

| Constant | Value | Use |
|---|---|---|
| acmewqeitm$k_length | 16 | |
| acmewqefix$k_length | 0 | |

**acmewqefix**

| | |
|---|---|
| acmewqefix$l_reserved | 0 |

## B.13 ACME Work Queue Entry Flags (ACMEWQEFLG)

The following are the contents of the aggregate structure acmewqeflg:

| Field | Use |
|---|---|
| acmewqeflg$l_flags_struct | Composite field |

The following constants are defined in conjunction with acmewqeflg:

| Constant | Value | Use |
|---|---|---|
| acmewqeflg$k_min_acme_flag | 0 | |
| acmewqeflg$k_max_acme_flag | 15 | |
| acmewqeflg$k_min_fi_flag | 0 | |
| acmewqeflg$k_phase_done | 0 | |
| acmewqeflg$k_no_retry | 1 | |
| acmewqeflg$k_max_fi_flag | 1 | |
| acmewqeflg$k_min_auth_flag | 12 | |
| acmewqeflg$k_preauthenticated | 12 | |
| acmewqeflg$k_no_external_auth | 13 | |
| acmewqeflg$k_skip_new_password | 14 | |
| acmewqeflg$k_null_net_user | 15 | |
| acmewqeflg$k_max_auth_flag | 15 | |

**acmewqeflg**

## B.14 ACME Work Queue Entry Item (ACMEWQEITM)

The following are the contents of the aggregate structure acmewqeitm:

| Field | Use |
|---|---|
| acmewqeitm$l_acme_id | ID of ACME which set the item |
| acmewqeitm$l_phase | Phase during which item was set |
| acmewqeitm$l_length | Size, in bytes, of data |
| acmewqeitm$ps_pointer | Address of data |

The following constants are defined in conjunction with acmewqeitm:

| Constant | Value | Use |
|---|---|---|
| acmewqeval$k_length | 12 | |

**acmewqeitm**

| | |
|---|---|
| acmewqeitm$l_acme_id | 0 |
| acmewqeitm$l_phase | 4 |
| acmewqeitm$l_length | 8 |
| acmewqeitm$ps_pointer | 12 |

## B.15 ACME Work Queue Entry Value (ACMEWQEVAL)

The following are the contents of the aggregate structure acmewqeval:

| Field | Use |
|---|---|
| acmewqeval$l_acme_id | ID of ACME which set the value |
| acmewqeval$l_phase | Phase during which value was set |
| acmewqeval$L_VALUE | Value |

The following constants are defined in conjunction with acmewqeval:

| Constant | Value | Use |
|---|---|---|
| acme$k_maxchar_acme_ident | 64 | Maximum length (in characters) of |
| acme$k_maxchar_acme_activity | 64 | Maximum length (in characters) of |

**acmewqeval**

| | |
|---|---|
| acmewqeval$l_acme_id | 0 |
| acmewqeval$l_phase | 4 |
| acmewqeval$L_VALUE | 8 |

## B.16 ACME Work Queue Entry (ACMEWQE)

The following are the contents of the aggregate structure acmewqe:

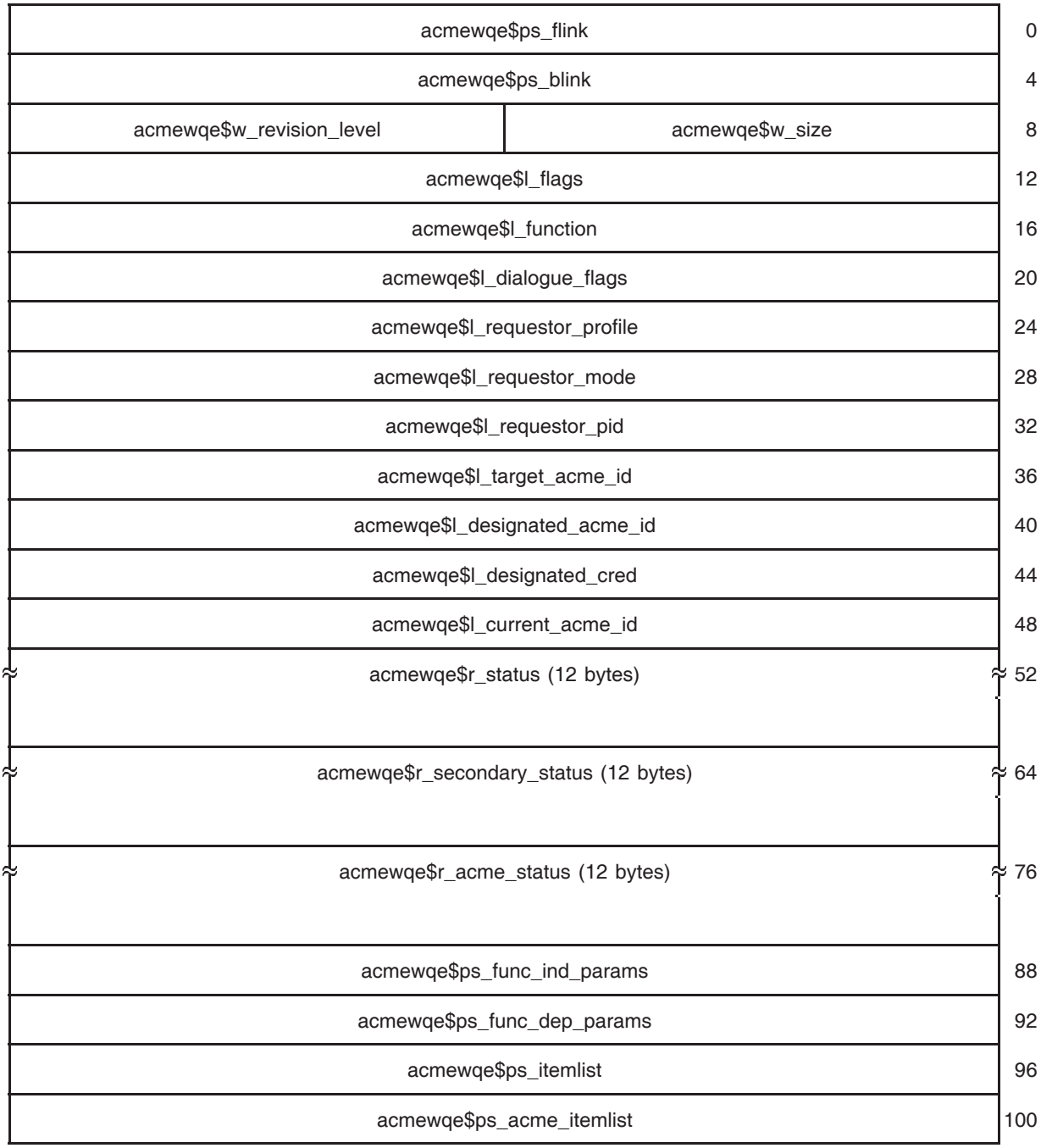
| Field | Use |
|---|---|
| acmewqe$ps_flink | WQE list forward link |
| acmewqe$ps_blink | WQE list backward link |
| acmewqe$w_size | Structure size, in bytes |
| acmewqe$w_revision_level | Structure revision level |
| acmewqe$l_flags | Status/control flags |
| acmewqe$l_function | Function code/modifiers |
| acmewqe$l_dialogue_flags | Dialogue support flags |
| acmewqe$l_requestor_profile | Requestor's security profile (Persona ID) |
| acmewqe$l_requestor_mode | Requestor's mode |
| acmewqe$l_requestor_pid | Requestor's PID |
| acmewqe$l_target_acme_id | Agent ID of ACME at which this request is directed |
| acmewqe$l_designated_acme_id | Agent ID of ACME that assumed processing control |
| acmewqe$l_designated_cred | Type of credentials associated with the designated ACME |
| acmewqe$l_current_acme_id | Agent ID of current ACME |
| acmewqe$r_status | First non-success status returned by an ACME |
| acmewqe$r_secondary_status | Secondary (protected) status |
| acmewqe$r_acme_status | ACME specific status |
| acmewqe$ps_func_ind_params | Function independent extension |
| acmewqe$ps_func_dep_params | Function dependent extension |
| acmewqe$ps_itemlist | ACME independent item list |
| acmewqe$ps_acme_itemlist | ACME specific item list |
| acmewqe$q_ast_context | AST context for which the AST has been recieved |
| acmewqe$r_locale | Locale specifier |
| acmewqe$r_service_name | Service (client) specifier |
| acmewqe$l_timeout_seconds | Seconds since system boot at which this request can be timed out |
| acmewqe$b_fill_9_ | |

The following constants are defined in conjunction with acmewqe:

| Constant | Value | Use |
|---|---|---|
| acmewqe$k_minor_id_000 | 0 | original V7.2-1 WQE supporting only COM |
| acmewqe$k_minor_id_001 | 1 | subsequent extension of the WQE |
| acmewqe$k_minor_id | 0 | |
| acmewqe$k_major_id_001 | 1 | |

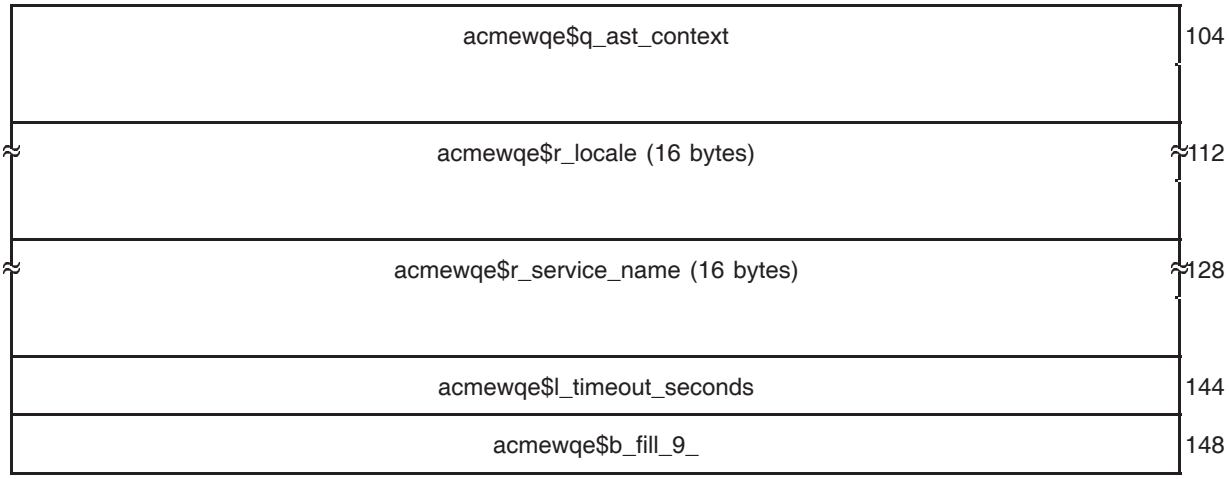
| Constant | Value | Use |
|---|---|---|
| acmewqe$k_major_id | 1 | |
| acmewqe$k_revision | 256 | |

## acmewqe




(continued on next page)

## B.16 ACME Work Queue Entry (ACMEWQE)



# C

# Persona Extension Interface Data Structures

## C.1 Persona Security Block (PSB)

This is the basic structure for a persona, including OpenVMS data.

The following are the contents of the aggregate structure PSB:

| Field | Use |
|---|---|
| PSB$L_FLINK | Standard listhead Forward link |
| PSB$L_BLINK | Standard listhead Backward link |
| PSB$W_SIZE | Standard structure size, in bytes |
| PSB$B_TYPE | Standard Type code for PSB (DYN$C_ SECURITY) |
| PSB$B_SUBTYPE | Standard Subtype code (DYN$C_SECURITY_ PSB) |
| PSB$L_DEBUG_FLINK | Forward link to previous PSB (DEBUG) |
| PSB$L_DEBUG_BLINK | Backward link to previous PSB (DEBUG) |
| PSB$L_DEBUG_PID | PID of process that allocated this PSB (DEBUG) |
| PSB$L_FLAGS | Every structure needs a set of flags |
| PSB$L_PERSONA_ID | Persona id assigned to this PSB |
| PSB$L_REFCOUNT | Number of attached execution contexts to this PSB |
| PSB$O_UID | Universal identifier assigned to persona |
| PSB$T_CLONE_REGION | Start of area copied when PSB is cloned |
| PSB$L_MODE | Access level for this PSB (User, Supervisor, Exec, Kernel) |
| End of the reserve PSB (runt). | |
| PSB$T_USERNAME | Persona Username |
| PSB$T_ACCOUNT | Persona Account name |
| PSB$L_NOAUDIT | Non-zero implies no audit status |
| PSB$L_UIC | UIC of persona |
| UIC_FLAGS must always follow UIC | |
| PSB$L_UIC_FLAGS | UIC identifier flags longword |
| (should always be zero) | |
| PSB$L_CONCEALED_COUNT | Hidden reference tracking |
| PSB$Q_RESERVED | Used for quadword alignment |
| PSB$Q_DOI | Domain Of Interpretation |
| PSB$L_RIGHTS_ENABLED | Bitmap of enabled/disabled rights pointers |

| Field | Use |
|---|---|
| PSB$AR_AUTHRIGHTS | Array of pointers to authorized rights chains |
| PSB$AR_RIGHTS | Array of pointers to active rights chains |
| PSB$AR_CLASS | Pointer to classification data block |
| PSB$L_PXB_COUNT | Number of extensions registered when this persona is created. |
| PSB$AR_PXB_ARRAY | Pointer to array of pointers to this PSB's extension blocks. |
| PSB$Q_PXB_MUTEX | Mutex to sych access to this PSB's fields. |

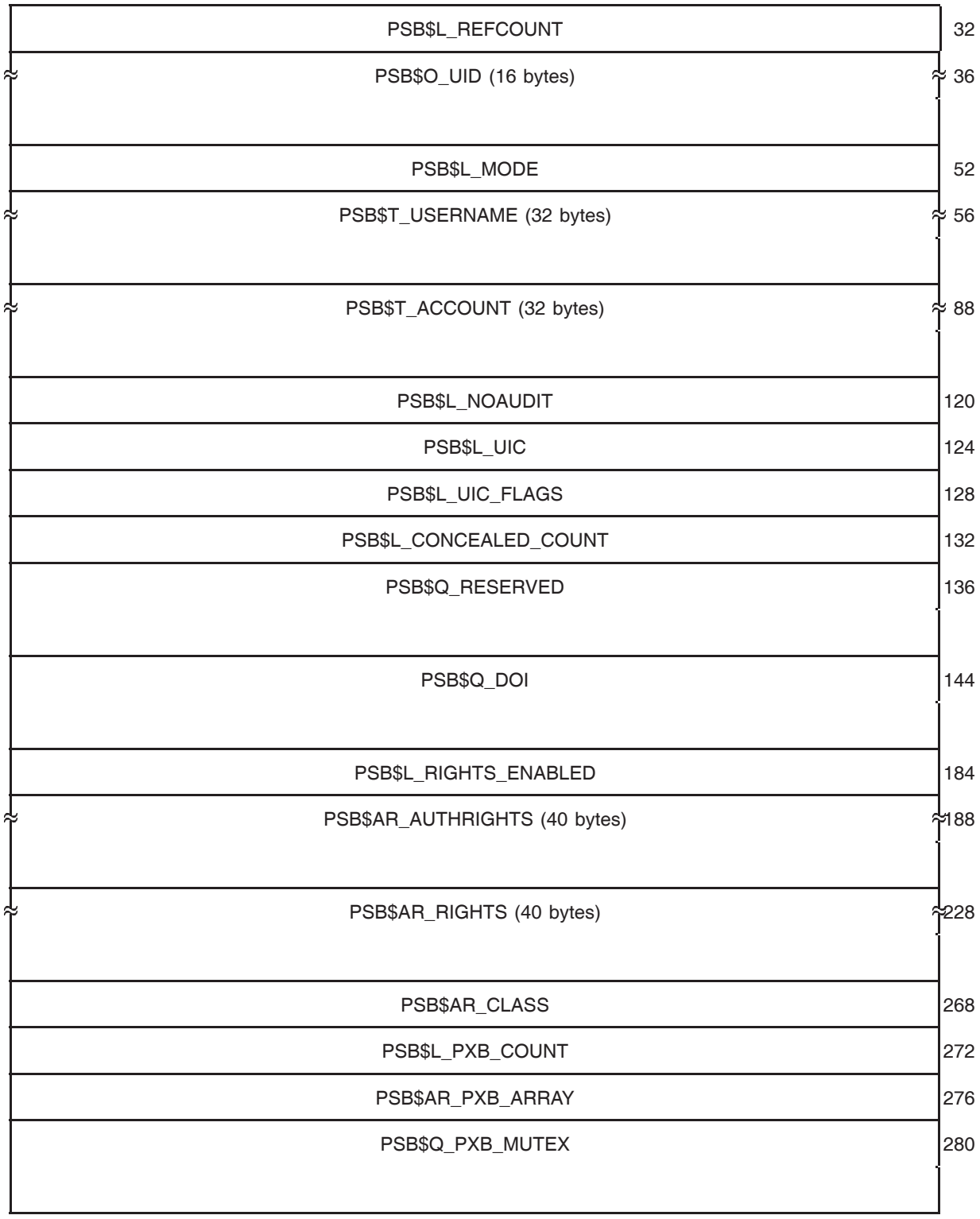
The following constants are defined in conjunction with PSB:

| Constant | Value | Use |
|---|---|---|
| PSB$K_SIZE_ACCOUNT | 32 | |
| PSB$K_SIZE_USERNAME | 32 | |
| PSB$K_COUNT_PERSONAE | 8 | |
| PSB$K_COUNT_RIGHTS | 10 | |
| PSB$K_RIGHTS_PERSONA | 0 | |
| PSB$K_RIGHTS_SYSTEM | 1 | |
| PSB$K_RIGHTS_INSTALLED | 2 | |
| PSB$K_RIGHTS_SUBSYSTEM | 3 | |
| PSB$K_RIGHTS_TEMPORARY | 4 | |
| PSB$K_RIGHTS_CHKPRO | 9 | |
| PSB$K_FLAGS_BIT_COUNT | 13 | Number of flags |
| PSB$K_RESERVE_PERSONA_ LENGTH | 56 | |
| PSB$K_CLONE_REGION_SIZE | 220 | |

## psb

| | | | |
|---|---|---|---|
| PSB$L_FLINK | | | 0 |
| PSB$L_BLINK | | | 4 |
| PSB$B_SUBTYPE | PSB$B_TYPE | PSB$W_SIZE | 8 |
| PSB$L_DEBUG_FLINK | | | 12 |
| PSB$L_DEBUG_BLINK | | | 16 |
| PSB$L_DEBUG_PID | | | 20 |
| PSB$L_FLAGS | | | 24 |
| PSB$L_PERSONA_ID | | | 28 |

(continued on next page)


PSB$L_REFCOUNT
32
PSB$O_UID (16 bytes)
36
PSB$L_MODE
52
PSB$T_USERNAME (32 bytes)
56
PSB$T_ACCOUNT (32 bytes)
88
PSB$L_NOAUDIT
120
PSB$L_UIC
124
PSB$L_UIC_FLAGS
128
PSB$L_CONCEALED_COUNT
132
PSB$Q_RESERVED
136
PSB$Q_DOI
144
PSB$L_RIGHTS_ENABLED
184
PSB$AR_AUTHRIGHTS (40 bytes)
188
PSB$AR_RIGHTS (40 bytes)
228
PSB$AR_CLASS
268
PSB$L_PXB_COUNT
272
PSB$AR_PXB_ARRAY
276
PSB$Q_PXB_MUTEX
280

## C.2 Persona Extension Block Array (PXB_ARRAY)

The PSB$AR_PXB_ARRAY cell of the Persona Security Block points to the PXB_ARRAY$AR_ELEMENTS field of this structure. PXB_ARRAY$AR_ELEMENTS contains pointers to all of the persona extension blocks associated with the persona.

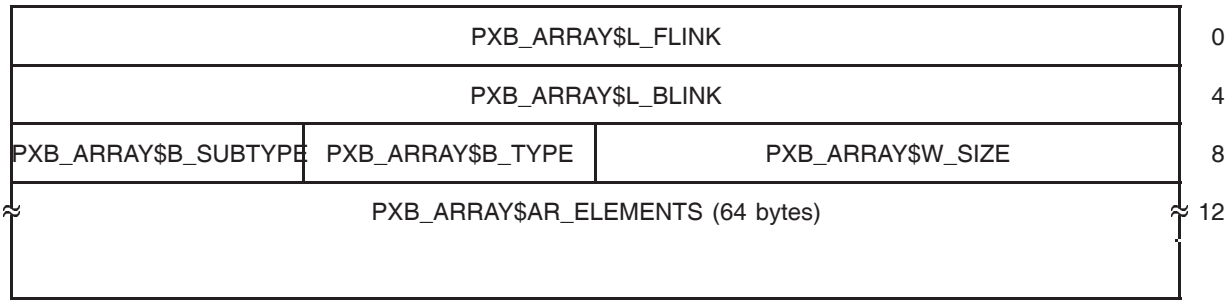
The following are the contents of the aggregate structure PXB_ARRAY:

| Field | Use |
|---|---|
| PXB_ARRAY$L_FLINK | Forward link |
| PXB_ARRAY$L_BLINK | Backward link |
| PXB_ARRAY$W_SIZE | Size |
| PXB_ARRAY$B_TYPE | Type code |
| PXB_ARRAY$B_SUBTYPE | Subtype code |
| PXB_ARRAY$AR_ELEMENTS | Array cells |

The following constants are defined in conjunction with PXB_ARRAY:

| Constant | Value | Use |
|---|---|---|
| PXB$K_LENGTH | 12 | Length of header |
| PXB_ARRAY_ELEMENTS | 16 | |
| PXB_ARRAY$K_PXB_ARRAY_HEADER | 12 | |

**pxb_array**



## C.3 Persona Extension Block (PXB)

This structure shows the header for your Persona Extension Block structure. PXB$B_TYPE should contain DYN$C_SECURITY. PXB$B_SYBTYPE should contain DYN$C_SECURITY_PXB_GENERIC. PXB$W_SIZE should contain the allocated size for this structure, including this header and all the subsequent fields you define.

The following are the contents of the aggregate structure PXB:

| Field | Use |
|---|---|
| PXB$L_FLINK | Forward link |
| PXB$L_BLINK | Backward link |
| PXB$W_SIZE | Size |
| PXB$B_TYPE | Type code |
| PXB$B_SUBTYPE | Subtype code |

**pxb**

| | | | |
|---|---|---|---|
| PXB$L_FLINK | | | 0 |
| PXB$L_BLINK | | | 4 |
| PXB$B_SUBTYPE | PXB$B_TYPE | PXB$W_SIZE | 8 |

The above fields describe only the fixed portion of the Persona Extension Block, common to all persona extensions. Following those cells you should include the data specific to your extension, noting the total resultant size of the structure in cell PXB$W_SIZE.

## C.4 Persona Extension Creation Flags (PXB_FLAGS)

If flag PXB$V_DEBIT is set, the user process is charged for the memory used in this this PXB, even if it is not being charged for the memory used in the parent PSB.

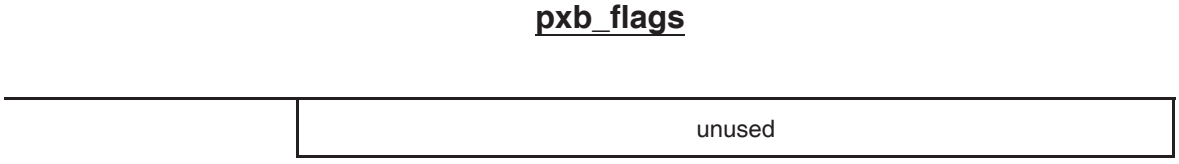
This flag should be returned by your persona extension image in response to a query for ISS$_COMMON_FLAGS.

Typically direct system service requests will charge the user process for memory usage, while implicit requests from within the TCB will not do so. If you want to cause even implict requests to charge the user, use the PXB$V_DEBIT flag.

The following are the contents of the aggregate structure PXB_FLAGS:

| Field | Length | Starts at | Use |
|---|---|---|---|
| PXB$V_FILL_1 | 1 bit | Bit 0 | Clone operation |
| PXB$V_FILL_2 | 1 bit | Bit 1 | Delegate operation |
| PXB$V_FILL_3 | 1 bit | Bit 2 | |
| PXB$V_FILL_4 | 1 bit | Bit 3 | |
| PXB$V_FILL_5 | 1 bit | Bit 4 | |
| PXB$V_FILL_6 | 1 bit | Bit 5 | |
| PXB$V_FILL_7 | 1 bit | Bit 6 | |
| PXB$V_DEBIT | 1 bit | Bit 7 | DEBIT[1] |

[1]This bit must be in synch with PSB flag.



## C.5 Persona Extension Dispatch Vector (PXVD)

Your persona extension image provides this specification of persona extension routines when it calls NSA$REGISTER_PSB_EXTENSION from its initialization routine, as described in Chapter 12. The following are the contents of the aggregate structure PXDV:

| Field | Use |
|---|---|
| PXDV$L_VERSION | Version of dispatch vector |
| PXDV$L_FLAGS | |
| PXDV$A_CREATE | Address to CREATE routine |
| PXDV$A_CLONE | Address to CLONE routine |
| PXDV$A_DELEGATE | Address to DELEGATE routine |
| PXDV$A_DELETE | Address to DELETE routine |
| PXDV$A_MODIFY | Address to MODIFY routine |
| PXDV$A_QUERY | Address to QUERY routine |
| PXDV$A_MAKE_TLV | Address to MAKE_TLV routine |

The following constants are defined in conjunction with PXDV:

| Constant | Value | Use |
|---|---|---|
| PXRB$K_LENGTH | 88 | Length of PXRB struct |
| PXDV$K_version | 1 | |
| PXDV$K_min_version | 1 | |
| PXDV$K_max_version | 1 | |

**pxdv**

| | |
|---|---|
| PXDV$L_VERSION | 0 |
| PXDV$L_FLAGS | 4 |
| PXDV$A_CREATE | 8 |
| PXDV$A_CLONE | 12 |
| PXDV$A_DELEGATE | 16 |
| PXDV$A_DELETE | 20 |
| PXDV$A_MODIFY | 24 |
| PXDV$A_QUERY | 28 |
| PXDV$A_MAKE_TLV | 32 |
| unused | 36 |

## C.6 Persona Extension Registration Block (PXRB)

This data structure in nonpaged pool provides ongoing storage for the addresses providedinteh PSDV by your persona extension image.

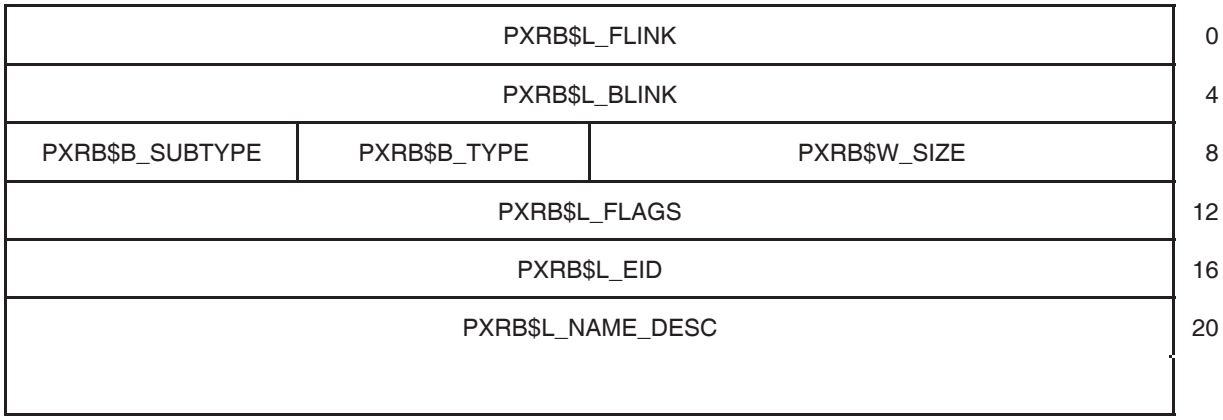
The following are the contents of the aggregate structure PXRB:

| Field | Use |
|---|---|
| PXRB$L_FLINK | Forward link |
| PXRB$L_BLINK | Backward link |
| PXRB$W_SIZE | Size |
| PXRB$B_TYPE | Type code |
| PXRB$B_SUBTYPE | Subtype code |
| PXRB$L_FLAGS | |
| PXRB$L_EID | Extension ID |
| PXRB$L_NAME_DESC | Extension name descriptor |
| PXRB$AR_NAME | Extension name string |
| PXRB$A_CREATE | Pointer to CREATE routine |
| PXRB$A_CLONE | Pointer to CLONE routine |
| PXRB$A_DELEGATE | Pointer to DELEGATE routine |
| PXRB$A_DELETE | Pointer to DELETE routine |
| PXRB$A_MODIFY | Pointer to MODIFY routine |
| PXRB$A_QUERY | Pointer to QUERY routine |
| PXRB$A_MAKE_TLV | Pointer to MAKE_TLV routine |

The following constants are defined in conjunction with PXRB:

| Constant | Value | Use |
|---|---|---|
| DELBK$C_DELEGATE_ BLOCK_SIZE | 40 | Length of DELBK struct |

**pxrb**




(continued on next page)

| | |
|---|---|
| PXRB$AR_NAME (32 bytes) | 28 |
| PXRB$A_CREATE | 60 |
| PXRB$A_CLONE | 64 |
| PXRB$A_DELEGATE | 68 |
| PXRB$A_DELETE | 72 |
| PXRB$A_MODIFY | 76 |
| PXRB$A_QUERY | 80 |
| PXRB$A_MAKE_TLV | 84 |

## C.7 Persona Extension Create Flags (CREATE_FLAGS)

This flag is used by a caller of SYS$PERSONA_CREATE_EXTENSION to indicate that the created extension should be the primary extension. The visible result for your persona extension image will be that the index of this will be in location 0 of the PXB Array.

The following are the contents of the aggregate structure CREATE_FLAGS:

| Field | Use |
|---|---|
| PXB$V_PRIMARY_EXTENSION | This field is 1 bit long, and starts at bit 0. Clone operation |
| PXB$V_FILL_6_ | This field is 31 bits long, and starts at bit 1. |

**create_flags**

## C.8 Persona Delegation Block (DELBK)

Use this structure to pass delegation information between processes.

The following are the contents of the aggregate structure DELBK:

| Field | Use |
|---|---|
| DELBK$L_FLINK | Forward link |
| DELBK$L_BLINK | Backward link |
| DELBK$W_SIZE | size |
| DELBK$B_TYPE | Type code |
| DELBK$B_SUBTYPE | Subtype code |
| DELBK$L_PERSONA_ID | Persona ID |
| DELBK$L_PERSONA_ADDRESS | Persona address |
| DELBK$L_CLIENT_EPID | Client's EPID |
| DELBK$L_CLIENT_TPID | Client's Thread PID |
| DELBK$L_BYTCNT | BYTCNT/BYTLM to be debited upon delegation |
| DELBK$L_IMGCNT | Value of PHD$L_IMGCNT when we started |
| DELBK$L_STATUS | Status |

**delbk**

| | | | |
|---|---|---|---|
| DELBK$L_FLINK | | | 0 |
| DELBK$L_BLINK | | | 4 |
| DELBK$B_SUBTYPE | DELBK$B_TYPE | DELBK$W_SIZE | 8 |
| DELBK$L_PERSONA_ID | | | 12 |
| DELBK$L_PERSONA_ADDRESS | | | 16 |
| DELBK$L_CLIENT_EPID | | | 20 |
| DELBK$L_CLIENT_TPID | | | 24 |
| DELBK$L_BYTCNT | | | 28 |
| DELBK$L_IMGCNT | | | 32 |
| DELBK$L_STATUS | | | 36 |

## C.9 PSB Ring Buffer (PSBRB)

Keeps a history of recent persona activity when the system is booted with the SECURITY_MON.EXE executive image.

The following are the contents of the aggregate structure PSBRB:

| Field | Use |
|---|---|
| PSBRB$L_FLINK | Standard listhead Forward link |
| PSBRB$L_BLINK | Standard listhead Backward link |
| PSBRB$W_SIZE | Standard structure size, in bytes |
| PSBRB$B_TYPE | Standard Type code for PSB (DYN$C_ SECURITY) |
| PSBRB$B_SUBTYPE | Standard Subtype code (DYN$C_ SECURITY_PRB) |
| PSBRB$L_MAX_INDEX | Maximum number of records in ring |
| PSBRB$L_CURRENT_INDEX | Index of nex record to be written in the RingBuffer |
| PSBRB$T_RECORDS | Offset to first record in RingBuffer |
| PSBRB$L_FUNCTION | Define which support routine is reporting this record |
| PSBRB$L_PSB | Address of PSB being acted upon by the function |
| PSBRB$L_REFCNT | Current reference count the above PSB |
| PSBRB$L_FLAGS | Contents of PSB flags |
| PSBRB$L_MODE | Access mode of current PSB |
| PSBRB$L_PC | Address from where the function was called |
| PSBRB$L_PSL | PSL at the time of the operation |
| PSBRB$L_UTHREAD_ID | Current active DECthread (if any) |
| PSBRB$L_KTB | Current KTB |
| PSBRB$L_CPU | CPU ID from current KTB |
| Function specific information | |
| PSBRB$L_AUX_1 | $Assume = previous PSB |
| $Clone = PSB being created | |
| PSBRB$L_AUX_2 | $Assume = previous PSB reference count |

The following constants are defined in conjunction with PSBRB:

| Constant | Value | Use |
|---|---|---|
| PSB$_TLV_FLAGS | 1 | 1 |
| PSB$_TLV_ARBFLAGS | 2 | 2 CHP$_FLAGS Placeholder to avoid conflict with ARB TLVs |
| PSB$_TLV_ARBPRIV | 3 | 3 CHP$_PRIVS Placeholder to avoid conflict with ARB TLVs |
| PSB$_TLV_MODE | 4 | 4 |
| PSB$_TLV_WORKCLASS | 5 | 5 |
| PSB$_TLV_RIGHTS | 6 | 6 |

| Constant | Value | Use |
|---|---|---|
| PSB$_TLV_USERNAME | 9 | 9 |
| PSB$_TLV_ACCOUNT | 10 | 10 |
| PSB$_TLV_NOAUDIT | 11 | 11 |
| PSB$_TLV_AUTHPRIV | 12 | 12 |
| PSB$_TLV_PERMPRIV | 13 | 13 |
| PSB$_TLV_IMAGE_WORKPRIV | 14 | 14 |
| PSB$_TLV_RIGHTS_ENABLED | 15 | 15 |
| PSB$_TLV_AUTHRIGHTS | 16 | 16 |
| PSB$_TLV_MINCLASS | 17 | 17 |
| PSB$_TLV_MAXCLASS | 18 | 18 |
| PSB$_TLV_UID | 19 | 19 |
| PSB$_TLV_UIC | 22 | 22 |
| PSB$_TLV_WORKPRIV | 23 | 23 |
| PSB$_TLV_MIN_CODE | 1 | |
| PSB$_TLV_MAX_CODE | 23 | |

## PSBRB

| | | | |
|---|---|---|---|
| PSBRB$L_FLINK | | | 0 |
| PSBRB$L_BLINK | | | 4 |
| PSBRB$B_SUBTYPE | PSBRB$B_TYPE | PSBRB$W_SIZE | 8 |
| PSBRB$L_MAX_INDEX | | | 12 |
| PSBRB$L_CURRENT_INDEX | | | 16 |
| PSBRB$L_FUNCTION | | | 20 |
| PSBRB$L_PSB | | | 24 |
| PSBRB$L_REFCNT | | | 28 |
| PSBRB$L_FLAGS | | | 32 |
| PSBRB$L_MODE | | | 36 |
| PSBRB$L_PC | | | 40 |
| PSBRB$L_PSL | | | 44 |
| PSBRB$L_UTHREAD_ID | | | 48 |
| PSBRB$L_KTB | | | 52 |
| PSBRB$L_CPU | | | 56 |

(continued on next page)

| | |
|---|---|
| PSBRB$L_AUX_1 | 60 |
| PSBRB$L_AUX_2 | 64 |
| unused | 68 |

## C.10 Persona Security Block Array (PSB_ARRAY)

Contains the master list of personas held by a process.

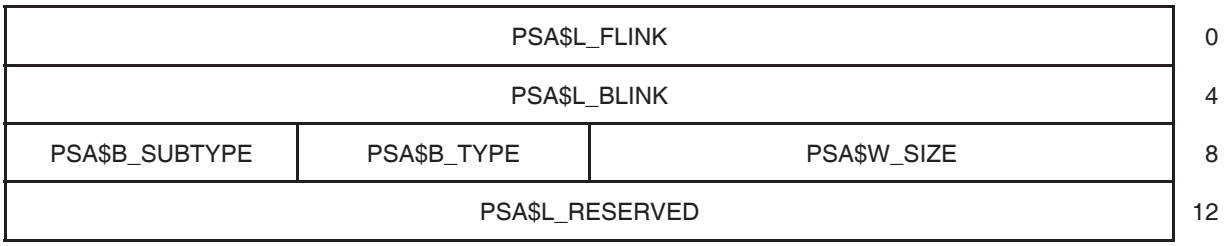
The following are the contents of the aggregate structure PSB_ARRAY:

| Field | Use |
|---|---|
| PSA$L_FLINK | Standard listhead Forward link |
| PSA$L_BLINK | Standard listhead Backward link |
| PSA$W_SIZE | Standard structure size, in bytes |
| PSA$B_TYPE | Standard Type code for PSB (DYN$C_SECURITY) |
| PSA$B_SUBTYPE | Standard Subtype code (DYN$C_SECURITY_PSB) |
| PSA$L_RESERVED | Keep array quad aligned ... |
| PSA$T_ELEMENTS | Offset to first element in array |

The following constants are defined in conjunction with PSB_ARRAY:

| Constant | Value | Use |
|---|---|---|
| PSB$K_LENGTH | 288 | Length of PSB structure |

### psb_array



# D

# ACME Agent and Persona Extension Code Examples

The release notes contain a pointer to code examples for a sample ACME agent and persona extension.

# Glossary

This glossary lists and defines the terms used in this guide.

**ACM**

See *Authentication and Credential Management*.

**ACM client process**

A process that calls the SYS$ACM[W] system service.

**ACM client program**

A program that calls the SYS$ACM[W] system service.

**ACM communications buffer**

An area provided by the SYS$ACM[W] system service via the **ACM context argument**. It contains an itemset to specify required user interaction in dialog mode.

**ACM context argument**

An argument to the SYS$ACM[W] system service that passes a pointer variable. If the SYS$ACM[W] system service requires additional information in dialog mode, it fills in that variable so it points to an ACM communications buffer.

**ACM dispatcher**

That code within the ACME server main image that makes calls to the ACME callout routines provided by the various ACME agent shareable images and reacts to the status codes they return.

**ACME**

See *Authentication and Credential Management Extension*.

**ACME agent**

The abbreviated name for an ACME agent shareable image.

**ACME agent control callout routine**

Supports SET SERVER ACME commands. One of four entry points provided by an ACME agent shareable image to be called by the ACME server main image. For controlling the startup, shutdown, suspension, or resumption of operations for that ACME agent shareable image.

**ACME agent shareable image**

A shareable image used within the ACME server process to implement one or more forms of authentication and optionally provide credentials to the process that called the SYS$ACM[W] system service. The VMS ACME is an example of an ACME agent shareable image that ships with the OpenVMS operating system. Others can be provided with add-on products or be written locally.

**ACME authentication and password callout routine**

Supports the processing of the SYS$ACM[W] system service function codes ACME$_FC_AUTHENTICATE_PRINCIPAL and ACME$_FC_CHANGE_ PASSWORD. One of 29 entry points provided by an ACME agent shareable image to be called by the ACME server main image. For processing function codes for that ACME agent shareable image.

**ACME callback routine**

An entry point provided by the ACME server main image to be called by any ACME agent shareable image for implementation of the authentication policy of one or more DOIs.

**ACME callout routine**

An entrypoint provided by an ACME agent shareable image to be called by the ACME server main image for implementation of the authentication policy of one or more DOIs.

**ACME event and query callout routine**

One of the ACME callout routines ACME$CO_EVENT or ACME$CO_QUERY. Provided by an ACME agent shareable image to be called by the ACME server main image. For processing the SYS$ACM[W] system service function codes ACME$_FC_EVENT or ACME$_FC_QUERY, respectively.

**ACME server log**

A text file written by the ACME server main image containing error and trace information from the ACM dispatcher and the various ACME agents. You can define the ACME$SERVER logical name to send the ACME server log to a specified disk. The default destination is SYS$SPECIFIC:[SYSMGR]ACME$SERVER.LOG.

**ACME server main image**

The main image that runs in an ACME server process and makes calls to each ACME agent shareable image. This image is supplied with the operating system.

**ACME server process**

A detached process that performs back end operations in support of the SYS$ACM[W] system service.

**ACME-specific resource**

An ACME-specific entity that can be cached with the ACM dispatcher and retrieved when needed or when next available in the scheduling cycle.

**ACME status**

The fourth longword returned in the structure to which the ACMSB argument to the SYS$ACM[W] system service points. The symbolic name of this cell is ACMESB$L_ACME_STATUS. The ACME status contains a status encoded in a format specific to a particular ACME agent unless the primary status contains one of the following values:

- SS$_BADITMCOD
- SS$_BADBUFLEN
- SS$_BADPARAM

When the primary status contains one of those values, the ACME status indicates what item code was in error.

**AST context**

A structure acquired by an ACME agent shareable image via an ACME callback routine to allow the use of AST-driven system services without actually running ACME agent shareable image code at AST level.

**Authentication and Credential Management (ACM)**

A set of tools that provide the ability to enhance or customize authentication services.

**Authentication and Credential Management Extension (ACME)**

An ACM program.

**authentication policy**

A set of rules determining how to establish the identity of (authenticate) users when they start to use the system. Most operating systems use passwords as the primary authentication mechanism. A system can have different authentication policies implemented for multiple DOIs at the same time.

**credentials**

A set of items used to validate access for a particular DOI. The SYS$ACM[W] system service returns credentials to the ACM client program as an attachment to a persona in the form of an additional persona extension. Other privileged components specific to the same DOI can use that persona extension in making security decisions. The OpenVMS Registry, for instance, can grant or deny access to particular data elements based on the NT persona extension.

**deferred confirmation**

A pattern of dialog mode operation in which an ACM client program confirms a no-echo prompt (such as for a new password) only after the initial response has been at least partially qualified by an ACME agent. This presents a more hospitable interface to users than immediate confirmation.

**designated DOI**

The Domain of Interpretation (DOI) chosen to prevail in processing a particular Authenticate Principal or Change Password request. Interaction between the various ACME agents on a system, in accordance with policy controls set by the system manager, leads to one of the ACME agents designating itself to provide the designated DOI. Other DOIs may contribute to authentication and may

provide credentials. When the call to the SYS$ACM[W] system service specifies a target DOI, that DOI will also be used as the designated DOI.

**dialog mode**

A form of operation whereby the ACM client program calls the SYS$ACM[W] system service successively to complete a full Authenticate Principal or Change Password operation. You specify dialog mode by providing the **context** argument when calling the SYS$ACM[W] system service.

**DOI**

See *domain of interpretation*.

**domain of interpretation (DOI)**

An authentication policy implemented by an ACME agent shareable image or by several in combination.

**event**

Information an ACM client program transmits to an ACME agent for use in some fashion specific to a particular DOI. It might be recorded in a log or used to trigger some mode of operation. Requirements for sending an event, including any required privilege, are specific to the DOI.

**executive image**

A program image specially constructed without transfer vectors and intended to be included as part of the modular OpenVMS executive. This is sometimes known by the informal name *Execlet*. A persona extension image that you write is an example of an executive image.

**extension ID**

The index number for a particular **persona extension** within persona data structures. The extension ID for persona extensions from a particular persona extension image will remain constant until the next time the system is booted.

**immediate confirmation**

A pattern of dialog mode operation in which an ACM client program confirms a no-echo prompt (such as for a new password) before returning the initial response to the ACME server process (and thus before any qualification of the new password regarding acceptability). This presents a lighter system load than deferred confirmation.

**item list**

A chain of item list segments with each segment terminated by the item ACME$_CHAIN, except for the final segment which is terminated by a zero item. Each ACME$_CHAIN item points to the successor segment.

**item list segment**

An array of standard VMS item_list_3 or item list entry B descriptors.

**itemset**

An array of itemset entries provided by the SYS$ACM[W] system service to specify required user interaction.

**itemset entry**

An element within an itemset describing a single user interaction request from an ACME agent.

**LGI callout**

A mechanism introduced in OpenVMS Version 5.5 for customizing LOGINOUT interactions. This was the predecessor to the ACME mechanism.

**logon type**

Also known as *login class*. One of the five types of authentication supported by the SYS$ACM[W] system service (local, dialup, remote, network, and batch). The type that is chosen (or defaulted, in the case of non-privileged callers) governs the degree of interaction that might be required for provision of passwords and changing of passwords.

**message category**

The code value indicating the purpose of output dialogue text.

**PSB**

See *Persona Security Block*.

**Persona Secutiry Block (PSB)**

A data structure used to implement a persona.

**non-dialogue mode**

A form of operation whereby the ACM client program calls the SYS$ACM[W] system service once with all items required.

**OpenVMS executive**

The combination of SYS$PUBLIC_VECTORS.EXE, SYS$BASE_IMAGE.EXE, and executive images that together form the core of the operating system.

**OpenVMS user name**

The name used to identify a user in non-authentication system service calls after a user is logged in. It is case-blind and limited to 12 alphanumeric characters making it considerably less flexible than the principal name. Note that the input prompt *Username:* is actually requesting a principal name.

**persona**

A kernel data structure (internal code PSB) associated with a process forming the basis for identity within the operating system.

**persona extension**

A kernel data structure (internal code PXB) attached to a persona associated with a process for the purpose of holding credentials for a particular DOI.

**Persona Extension Block (PSB)**

A data structure used for any persona extension other than the OpenVMS persona extension.

**persona extension image**

An executive image containing support routines for a particular persona extension.

**persona extension routine**

A routine contained within a persona extension image that is invoked in kernel mode by the OpenVMS executive to support operations on a particular persona extension.

**persona ID**

A longword value representing a persona held by a particular process. Translation of that value into a reference to the corresponding **PSB** is handled entirely by persona system services.

**primary status**

The first longword returned in the structure to which the ACMSB argument to the SYS$ACM[W] system service points. The symbolic name of this cell is ACMESB$L_STATUS. It indicates the overall status of the request.

**principal name**

The initial name used to claim an identity, expressed in a syntax appropriate for a particular DOI. Note that the traditional input prompt *Username:* is actually requesting a principal name. In simple cases the spelling of the principal name is the same as the spelling of the OpenVMS user name to which it maps.

**principal name mapping**

The transformation performed by an ACME agent that determines what OpenVMS user name is associated with a particular principal name.

**PXB**

See *Persona Extension Block*.

**request**

The collection of data within the ACME server process pertaining to a particular call or related set of calls to the SYS$ACM[W] system service by a client process. The physical manifestation of a request is centered in a work queue entry.

**return status**

The value returned by the SYS$ACM[W] system service. Success indicates only that the request was sent to the ACME server process. Success does not indicate the final result of processing.

**secondary status**

The second longword returned in the structure to which the ACMSB argument to the SYS$ACM[W] system service points. The symbolic name of this cell is ACMESB$L_SECONDARY_STATUS. It indicates a more detailed explanation of the primary status.

**status ACME ID**

The third longword returned in the structure to which the ACMSB argument to the SYS$ACM[W] system service points. The symbolic name of this cell is ACMESB$L_ACME_ID. It indicates the identity of the ACME agent that provided status information.

**SYS$ACM[W] system service**

The Authentication and Credential Management system service.

**target DOI**

The DOI specified on the initial call to the SYS$ACM[W] system service to be the one to handle the request.

**targeted request**

A request where the caller of the SYS$ACM[W] system service specifies item code ACME$_TARGET_DOI_ID or item code ACME$_TARGET_DOI_NAME to indicate which DOI should handle the request.

**TCB**

See *trusted computing base*.

**trusted computing base (TCB)**

A combinatin of computer hardware and operating system software that enforces a security policy. In OpenVMS systems, the TCB includes the entire executive and file system, all other system components that do not execute in user mode (such as device drivers, RMS, and DCL), most system programs installed with privilege, and a variety of other utilities used by system managers to maintain data relevant to the TCB.

**UCS encoding**

Unicode Character Set encoding. This uses the character set under which characters are represented in 16 bits. OpenVMS uses UCS2-4, in which each 16-bit character is actually stored in a 32-bit cell (4 bytes).

**VMS ACME**

The ACME agent that implements the traditional pre-ACME OpenVMS authentication policy.

**well-known item**

The seven common input text items that might be requested by any ACME agent: ACME$_PASSWORD_SYSTEM, ACME$_PRINCIPAL_NAME, ACME$_ PASSWORD_1, ACME$_PASSWORD_2, ACME$_NEW_PASSWORD_SYSTEM, ACME$_NEW_PASSWORD_1 or ACME$_NEW_PASSWORD_2.

**WQE**

See *work queue entry*.

**work queue entry (WQE)**

A data structure provided by the ACME server main image that tracks the progress of a particular request from a client process as it is handled by the ACME server main image and various ACME agent shareable images.

# Index

## A

## B



## C

## D



## E



## F



## G



## I



## M

## T



## U



## V



## W