# OpenVMS Technical Journal

V4, June 2004

# Table of Contents

# Introduction to Developing ACME Agents

Takaaki Shinagawa OpenVMS Security Engineering, HP

## Overview

ACME (Authentication Credential Management Extension) is the new authentication subsystem provided as an EAK (Early Adopter Kit)[1] on OpenVMS Alpha Version 7.3-2. ACME provides a "plug-in" environment in which individual ACME agents for different authentication policies can be loaded independently. In addition, ACME allows application programs to perform authentication directly through the $ACM system service. Currently, HP provides the native VMS, Windows NTLM, and LDAP agents. Third parties can develop ACME agents for new authentication policies. Although the concept of ACME is very similar to PAM (Pluggable Authentication Module) on Unix platforms, ACME has a proprietary architecture and programming interfaces. Developing an ACME agent requires solid understandings of the overall ACME subsystems, interactions between ACME agents, data structures and callout and callback functions in the agent, persona extensions, and $ACM clients. The purpose of this article is to provide an overview of ACME and easy-to-understand instructions on how to develop ACME agents.

The next section, Introduction to ACME, provides an overview of the entire ACME subsystem and introduces concepts that are prerequisites for ACME agent development. Instructions for developing ACME agents and persona extensions are addressed in Section 3, Implement an ACME Agent, and Section 4, Implement a Persona Extension, respectively. Section 5, Configure ACME, shows how to configure an ACME subsystem using all of the components.

## Introduction to ACME

### What is an ACME Agent?

An ACME agent is a module that implements an authentication policy on an OpenVMS system. When we log into an OpenVMS system, the system asks us to enter a username and password, and then it performs authentication. By default, authentication on OpenVMS systems is performed by comparing the user-entered username and password against the SYSUAF file. This is the authentication policy implemented in the VMS ACME agent. In addition to the VMS agent, the NT and LDAP agents are available with OpenVMS Alpha today. The NT ACME agent authenticates users with the NTLM database with the Microsoft LAN Manager authentication policy, and the LDAP ACME agent authenticates users against an LDAP server.

What if we need new authentication policies for our OpenVMS operating environments? This is the motivation of developing new ACME agents. If we want new ways of authenticating users on OpenVMS systems, the answer is to develop a new ACME agent. You can develop new ACME agents for new authentication policies to accommodate your needs.

[1] Because this is an early adopter kit release, ACME should be used only for non-production purposes. The SYS$ACM component, however, has been ready for production since OpenVMS Alpha Version 7.3-1. Currently the platform for ACME agent development is OpenVMS Alpha Version 7.3-2 or higher. As of this writing (May 2004), ACME is not yet available on OpenVMS I64 (Itanium). There is no plan to provide this functionality on OpenVMS VAX.

## What is the ACME subsystem?

The ACME subsystem refers to all of the components for authentication on an OpenVMS system (Figure 1). Its main feature is in its "plug-in" architecture, in which you can load individual ACME agents implementing authentication policies. An OpenVMS system administrator can load an ACME agent for a new authentication policy. For example, in order to authenticate users against an LDAP directory, the system administrator can load the LDAP ACME agent along with the VMS agent. Authentication is invoked by the $ACM system service in the application.

There are two distinct types of ACME agents: DOI and Auxiliary agents. The most significant difference is whether the agent issues credentials. A DOI agent has a capability to issue credentials, but an Auxiliary agent does not. An Auxiliary ACME agent implements specific functions that complement a DOI agent. Extra functions such as additional authentication methods (e.g. token-based and smart card), password filtering and new password checking can be implemented in Auxiliary agents.

An OpenVMS system manager can load multiple ACME agents in addition to the mandatory VMS ACME agent. When multiple DOI agents handle requests (authentication, authorization, credential generation, etc.) in an ACME subsystem, it is called a cooperative model. An independent model refers to a situation where only one DOI ACME agent processes requests and issue credentials along with the VMS ACME agent

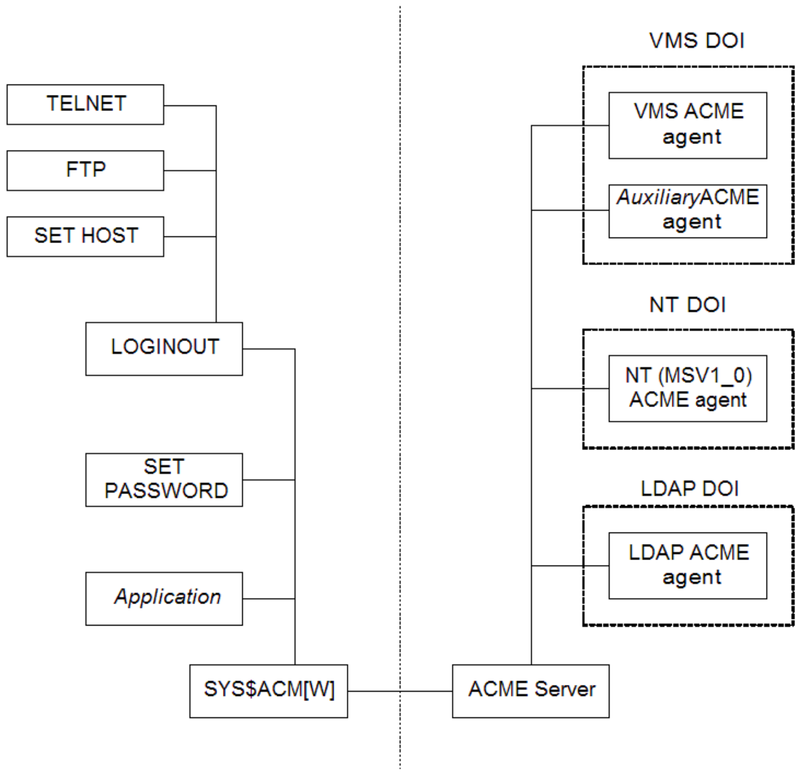


**Figure 1: Overview of the ACME Subsystem**

## How does the ACME subsystem work?

As shown in Figure 1, the ACME subsystem is composed of the ACME server, ACME agents and the $ACM application. Applications send an authentication-related request by calling the SYS$ACM system service. The request is eventually processed by the ACME agent. In order for an OpenVMS system manager to configure the ACME subsystem, the SET SERVER ACME commands are provided on the DCL command line. A persona extension also plays an important role in conjunction with the subsystem—it is responsible for storing credentials issued by the ACME agent.

In the following lines, interactions inside the ACME subsystem are described in the order the ACME subsystem is configured and processes requests.

1. If all the ACME components have been in place, the first step to using the ACME subsystem is to start the ACME server (SET SERVER ACME/START). The ACME server can be considered as an engine of the ACME subsystem— it dispatches requests from a $ACM application to ACME agents.

2. Once the ACME server is started, it becomes possible to load ACME agents. When an ACME agent is being loaded in the ACME subsystem (SET SERVER ACME/CONFIGURE), the ACME$CO_AGENT_INITIALIZE control callout function is executed in the agents (Figure 2).

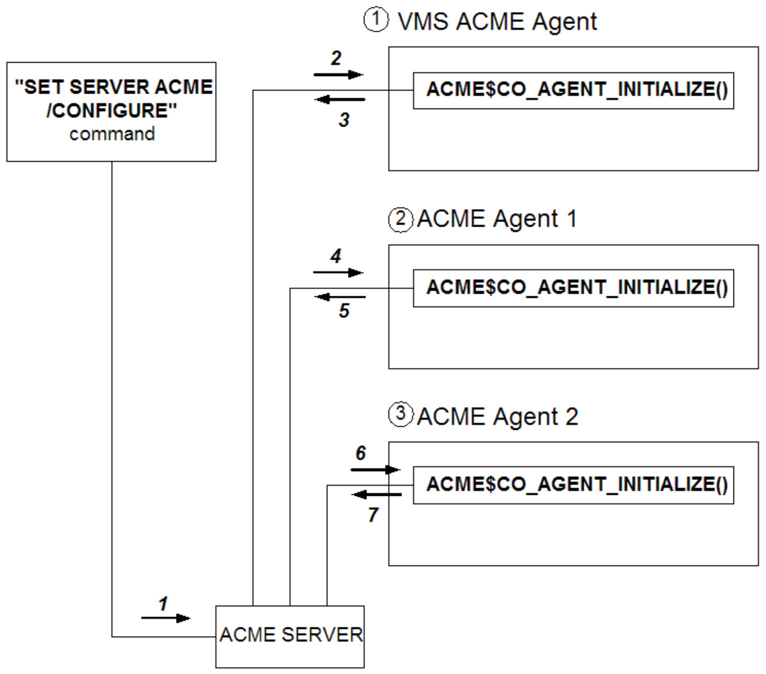

① VMS ACME Agent
ACME$CO_AGENT_INITIALIZE()
② ACME Agent 1
ACME$CO_AGENT_INITIALIZE()
③ ACME Agent 2
ACME$CO_AGENT_INITIALIZE()
"SET SERVER ACME /CONFIGURE" command
ACME SERVER
1
2
3
4
5
6
7

**Figure 2: ACME Control Flow for ACME$CO_AGENT_INITIALIZE**

3. To activate request dispatching, the system manager runs the SET SERVER ACME /ENABLE command. At this time, the ACME$CO_AGENT_STARTUP routine in each ACME agent specified in the command is executed (Figure 3).

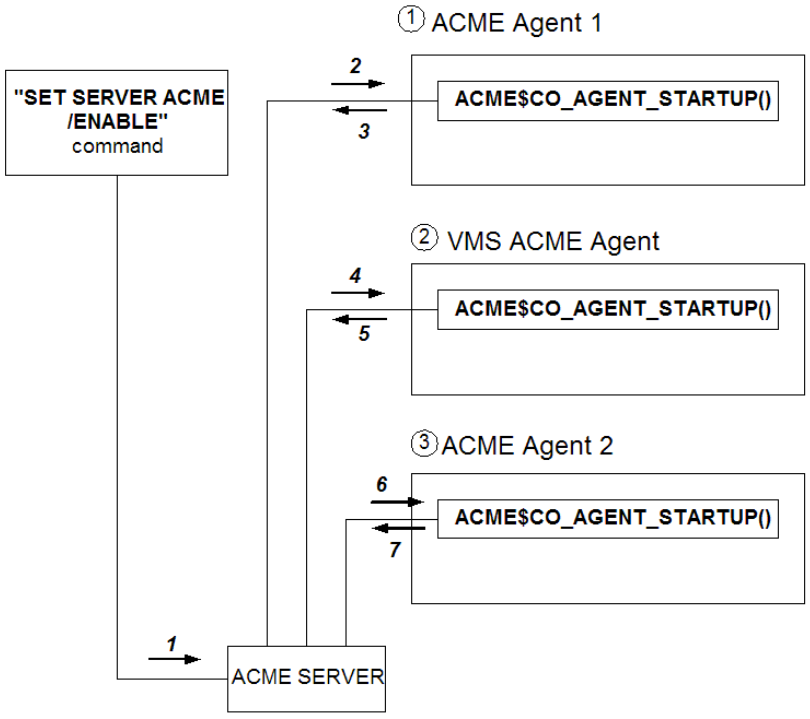


**Figure 3: ACME Control Flow for ACME$CO_AGENT_STARTUP**

4. Once the dispatching is enabled successfully, the ACME server and agents are ready to process the requests from the $ACM application. Every Authenticate Principal or Change Password request is performed using the following steps:

   4-1. The application calls the $ACM system service. The ACME$_FC_AUTHENTICATE_PRINCIPAL function code is specified for the Authenticate Principal request, and ACME$_FC_CHANGE_PASSWORD is specified for the Change Password request.

   4-2. The ACME server receives the request from the $ACM system service and dispatches it to the ACME agents.

   4-3. The requests are processed by the ACME agents in the order the agents are loaded.

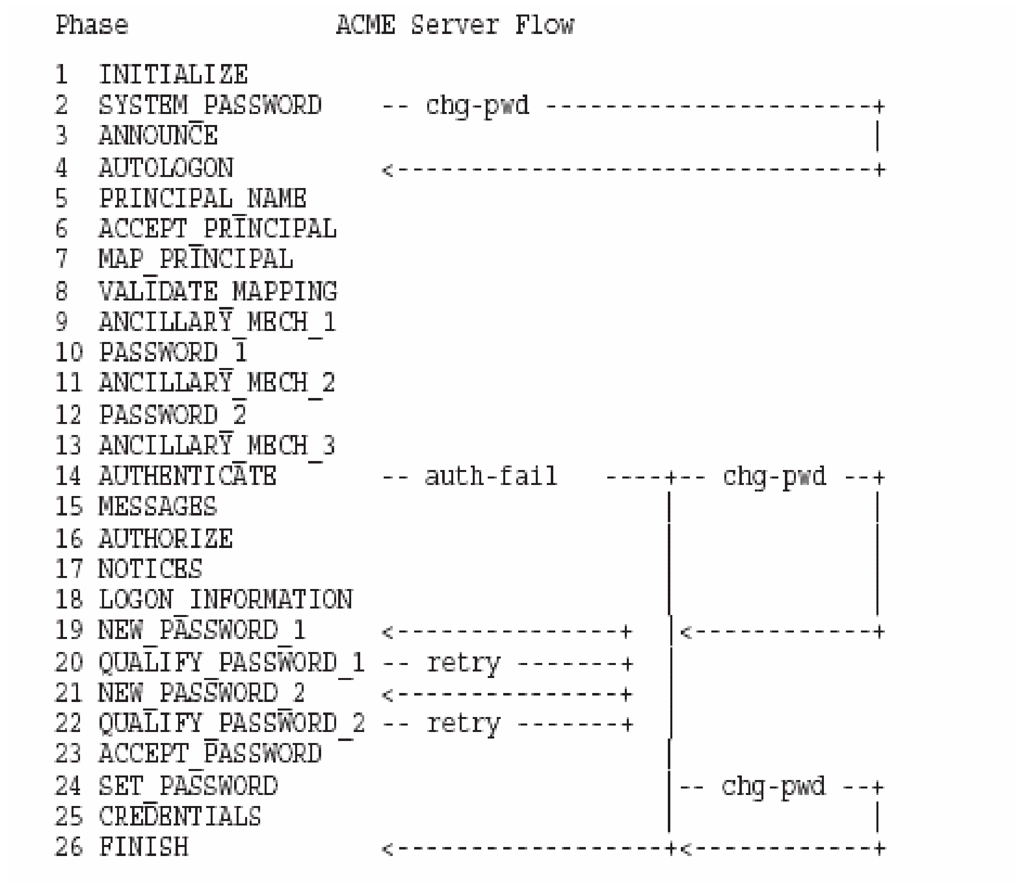
   4-4. Each time the request is processed by a callout routine, a return value from the routine to the ACME server determines the request's flow.

   - If ACME$_CONTINUE is returned, the ACME server dispatches the request to the next callout routine.
   - If the agent returns ACME$_PERFORMDIALOGUE, the ACME server performs a specified input/output dialogue, and then the request is dispatched to the same callout routine again.

- If the agent returns ACME$_FAILURE, the server dispatches the request to the final callout routine, ACME$CO_FINISH.
- If the agent returns ACME$_AUTHFAILURE, the ACME server continues processing through the ACME$CO_AUTHENTICATE callout routine, and then the service will return ACME$_AUTHFAILURE to the $ACM client.

For more return values, refer to Table 1-2 in the *ACME Developer's Guide*.

4-5. Repeat Steps 4-3 through 4-5 until you complete the ACME$CO_FINISH callout routine. Figure 4 shows all phases and request flows for authenticate-principal and change-password requests.



```
Phase                ACME Server Flow

1  INITIALIZE
2  SYSTEM_PASSWORD    -- chg-pwd ----------------------+
3  ANNOUNCE                                            |
4  AUTOLOGON          <--------------------------------+
5  PRINCIPAL_NAME
6  ACCEPT_PRINCIPAL
7  MAP_PRINCIPAL
8  VALIDATE_MAPPING
9  ANCILLARY_MECH_1
10 PASSWORD_1
11 ANCILLARY_MECH_2
12 PASSWORD_2
13 ANCILLARY_MECH_3
14 AUTHENTICATE       -- auth-fail   ----+-- chg-pwd --+
15 MESSAGES                              |             |
16 AUTHORIZE                             |             |
17 NOTICES                               |             |
18 LOGON_INFORMATION                     |             |
19 NEW_PASSWORD_1     <---------------+  |<------------+
20 QUALIFY_PASSWORD_1 -- retry -------+  |
21 NEW_PASSWORD_2     <---------------+  |
22 QUALIFY_PASSWORD_2 -- retry -------+  |
23 ACCEPT_PASSWORD                       |
24 SET_PASSWORD                          |-- chg-pwd --+
25 CREDENTIALS                           |             |
26 FINISH             <------------------+<------------+
```

**Figure 4: Authentication and change-password request phases and flow**

Both Authenticate Principal and Change Password requests follow the procedures described above. There are, however, some differences in callout routines that these two types of requests go through. As shown in Figure 4, the Authenticate Principal and Change Password requests are processed through different series of callout routines. Some callout routines are specific to either type of requests. Callout routines from ACME$CO_NEW_PASSWORD_1 through ACME$CO_SET_PASSWORD (Figure 4) are used to handle one or two new password(s)—these are essential to a change-password request. But an authentication request does not go though these routines unless the password is expired, and a new password must be obtained from the user. For a change-password request, the ACME server doesn't dispatch the request to the ACME$CO_ANNOUNCE, ACME$CO_MESSAGES, ACME$CO_AUTHORIZE, ACME$CO_NOTICES, ACME$CO_LOGON_INFORMATION, and ACME$CO_CREDENTIALS routines. The next section describes more details about callout routines.

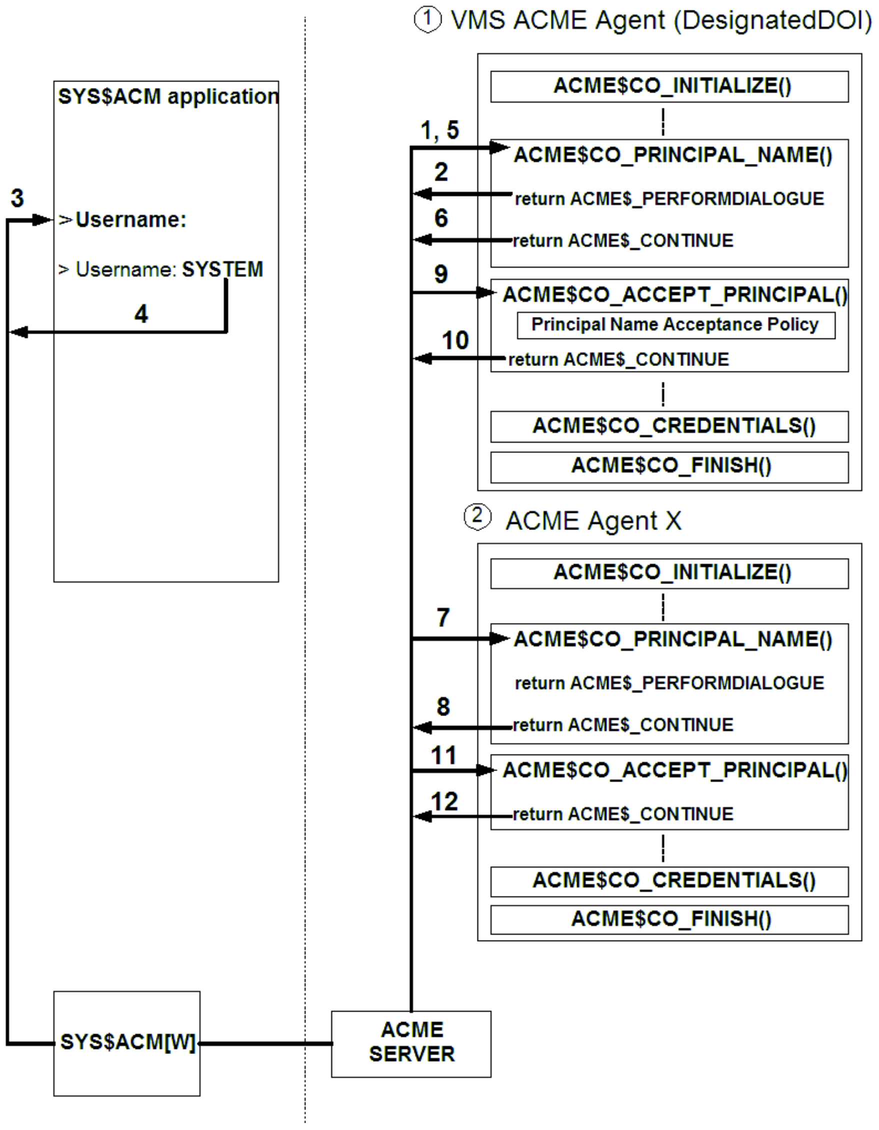


**Figure 5: ACME Control Flow for the PRINCIPAL_NAME and ACCEPT_PRINCIPAL phases**

Figure 5 illustrates ACME control flow in the PRINCIPAL_NAME and ACCEPT_PRINCIPAL phases. Whereas Figure 4 shows sequence and flow only in an ACME agent, Figure 5 shows the ACME control flow when multiple ACME agents are loaded. After completing the INITIALIZE through AUTO_LOGON phases, the ACME dispatches the request to the ACME$CO_PRINCIPAL_NAME callout routine of the ACME agent loaded first (1 in Figure 5). To queue a username prompt (Username: ) to the $ACM client, this callout routine returns ACME$_PERFORMDIALOGUE (2). The ACME server sends the prompt string to the $ACM application (3). After the user enters a username (4), the request is dispatched again to the PRINCIPAL_NAME callout routine (5). If the principal name is received, the callout routine returns ACME$_CONTINUE (6). If only one ACME agent is loaded, the request is dispatched into the ACCEPT_PRINCIPAL routine for the next step. In this example, however, the ACME server dispatches the request to the same phase in the agent loaded in the second place (7). Because this phase has already been completed by the first agent, the second agent returns ACME$_CONTINUE (8). Then, the ACME server dispatches the request to the next phase, ACCEPT_PRINCIPAL, in the first agent (9). If the principal name policy accepts the user-entered principal name, the ACME$CO_ACCEPT_PRINCIPAL routine returns ACME$_CONTINUE (10), and the request is dispatched to the same callout routine in the second agent (11). The second agent is not a Designated DOI agent (the Designated DOI should have been declared by the first one in the ACCEPT_PRINCIPAL phase or before). Thus it simply returns ACME$_CONTINUE (12). The request will be dispatched by the ACME server in the same way through the FINISH phase.

Upon successful authentication, a $ACM client can acquire credentials from the ACME agent that supports issuing credentials in the CREDENTIALS callout routine. The ACME agent issues credentials to a persona extension associated with the $ACM application by calling the ACMEKCV$CB_ISSUE_CREDENTIALS callback function in its ACME$CO_CREDENTIALS callout routine. Once the credentials have been issued into the persona extension, the $ACM application can perform operations with the credentials by calling the PERSONA system services such as SYS$PERSONA_ASSUME and SYS$PERSONA_QUERY. To retrieve and handle the credentials, the application must support the formats of the credentials. For example, an application that supports only the VMS credentials cannot handle those newly defined in another persona extension. It is essential that the $ACM client, ACME agents, and persona extensions agree on the credentials formats. Figure 6 depicts the whole process of issuing and acquiring the credentials.

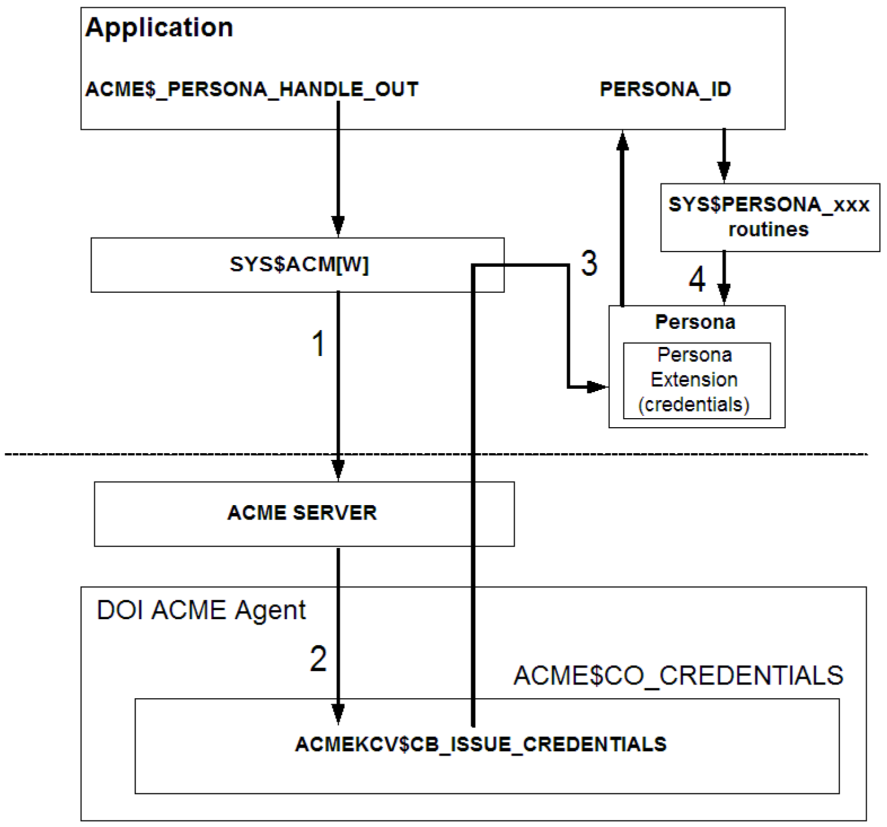


**Figure 6: Credential generation process**

In addition to the authenticate-principal and change-password requests, there are three types of requests from a SYS$ACM system service: QUERY, EVENT, and RELEASE_CREDENTIALS. Their function codes with the SYS$ACM are ACME$_FC_QUERY, ACME$_FC_EVENT and ACME$_FC_RELEASE_CREDENTIALS. The QUERY and EVENT requests are processed by the ACME$CO_QUERY and ACME$CO_EVENT callout routines in an ACME agent, respectively. The RELEASE_CREDENTIALS request simply deletes the credentials in the persona extension. The implementation of these callout routines will be discussed in the next section.

When a system is terminating the ACME subsystem or disabling ACME agents, ACME$CO_AGENT_SHUTDOWN is executed in the same manner as ACME$CO_AGENT_STARTUP (Figure 3).

## What do we have to develop?

As mentioned above, if you are developing an ACME agent that issues credentials, we also need to develop its persona extension as well as an ACME agent. The next section presents instructions for developing an ACME agent, and the following section describes how to implement a persona extension. Also, if the credential type and input requested by the agent are not supported by the existing $ACM application, it is necessary to develop a new one or modify the application.

## Implement an ACME Agent

### Determine the type of the ACME agent

Before you start the implementation, decide on the type of the ACME agent. The ACME agent type is either a DOI or Auxiliary agent. As mentioned in the Introduction to ACME section, a DOI agent issues credentials. An Auxiliary ACME agent does not work alone—it is designed to provide additional functionality such as complementary authentication in conjunction with a DOI agent.

An DOI agent acts as the Designated DOI agent when it is the target of the $ACM call. If you are developing a DOI agent, you also need to decide whether it can function as a Secondary DOI agent in the cooperative model for an untargeted $ACM call. In the cooperative model, each of multiple DOI agents (both Designated and Secondary DOI agents) is expected to be responsible for authentication and issuing credentials. In the independent model, only the DOI agent issues credentials—Secondary DOI agents other than the VMS agent do not perform those operations.

It is desirable to know all other ACME agents with which your agent will be configured in the ACME subsystem. When loading multiple ACME agents, the collection of those rules in all agents defines the operation model (either cooperative or independent model). The ACME server or ACME subsystem doesn't control behaviors of ACME agents to conform the specific operation model. Instead, rules implemented in each agent and occasionally order of agents make differences in their operations. The more details you know about other agents, the easier it becomes to develop an ACME agent that works correctly with those agents.

### Start implementing an ACME agent

Once you decide the ACME agent type, it is time to start implementing the ACME agent. The example below is based on the example code shipped in OpenVMS Alpha (SYS$EXAMPLES:ACME_EXAMPLE_DOI_ACME.C). This example agent is a DOI agent for the independent model and demonstrates the basic implementation necessary for any agent. If you are new to ACME agent development, it is highly recommended that you create a new one based on the example ACME agent.

### Define data structures

In the ACME agent, you define three data structures to store data specific to the agent or each request. The ACME subsystem uses ACME data structures such as WQE (Work Queue Entry) and KCV (Kernel Callback Vector), but they are already predefined in the system header files such as acmedef.h.

The acme_context data structure

Information in the acme_context data structure is kept as long as the ACME agent is active (from startup until shutdown). Thus, data such as authentication success/failure and the number of requests can be defined and stored in this data structure.

The request_context (= wqe_context) data structure

The request_context data structure is also called wqe_context— they are synonyms. Information in the request_context data structure is kept during a single request. Authentication data and user/account specific information associated with an authentication request can be defined and stored in this data structure. For example, a username and password received from the $ACM client are stored.

The credential data structure

If the ACME agent issues credentials, this data structure is essential. All credential fields must be defined in a credential data structure. The definitions of the credential data must be the same as that in the persona extension source code.

### Implement callout routines

The major task of developing an ACME agent is to implement a series of callout routines. They can be classified into the following three types:

- control callout routines
- authenticate-principal/change-password callout routines
- event and query callout routines

The steps that should be implemented in those callout routines are described below. For implementation in C, refer to the example program (ACME_EXAMPLE_DOI_ACME.C) in SYS$EXAMPLES.

This article provides comprehensive procedures for callout functions. For more details such data types of arguments and item/function codes, refer to the *ACME Developer's Guide*.

### Control callout routines

The control routines are ACME$CO_AGENT_INITIALIZE, ACME$CO_AGENT_STARTUP, ACME$CO_AGENT_STANDBY, and ACME$CO_AGENT_SHUTDOWN. As explained in the Introduction to ACME section, these control callout routines are invoked when a system manager loads, starts, suspends, and shuts down the ACME agents/server.

ACME$CO_AGENT_INITIALIZE

This routine is executed when loading the ACME agent (with SET SERVER ACME /CONFIGURE command).

1. Verify the revision levels of the WQE and KCV data structures. The purpose is to check the compatibility of data structure versions of the ACME server and the ACME agent. A structure revision level should be checked in WQE, whereas both ACM kernel and structure revision levels are checked in KCV. In each revision levels, major and minor versions exist. The major versions need to be equal, and the minor version from the ACME server (in WQE) needs to be greater than or equal to the one with the ACME agent. If the

revision levels mismatch, return ACME$_UNSUPREVLVL. For more details in the revision levels, refer to Section 4.1.1 and 4.1.2 in the *ACME Developer's Guide*.

2. Initialize ACME Resource Block (ACMERSRC). Although this initialization is optional for most fields in ACMERSRC, it is required to explicitly set some fields such as privileges depending on the ACME agent's operations. More information about ACME Agent Resource Requirements Block is available in Section B.5 in the *ACME Developer's Guide*.
3. Set the ACME agent's name and report it to the ACME server with ACMEKCV$CB_REPORT_ATTRIBUTES().
4. Set the current status string and report it to the ACME server with ACMEKCV$CB_REPORT_ACTIVITY().
5. Return ACME$_CONTINUE.

ACME$CO_AGENT_STARTUP

This routine is executed when activating the ACME agent (with the SET SERVER ACME /ENABLE command).

1. Allocate the acme_context data structure with ACMEKCV$CB_ALLOCATE_ACME_VM().
2. Initialize the counters in acme_context if they exist.
3. Return ACME$_CONTINUE.

ACME$CO_AGENT_SHUTDOWN

This routine is executed when stopping (with SET SERVER ACME /EXIT) or disabling (with SET SERVER ACME /DISABLE) the ACME agent.

1. Deallocate the acme_context data structure with ACMEKCV$CB_DEALLOCATE_ACME_VM().
2. Return ACME$_NORMAL upon success.

ACME$CO_AGENT_STANDBY

This routine is executed when temporarily disabling (with SET SERVER ACME /SUSPEND) the ACME agent. The purpose of this operation is to close files for a possible system backup operation. Note that there is no corresponding resume callout routine. The request processing resumes when the operator issues the SET SERVER ACME/RESUME command. The example agent does not implement any operation in this callout routine and returns ACME$_CONTINUE.

## Request callout functions

The request routines are invoked by an authentication request from a $ACM client.

ACME$CO_INITIALIZE

This is the first routine every authenticate-principal/change-password request goes through. ACME$CO_INITIALIZE allocates request_context, which is the context data structure for a request, and process common and ACME-specific item lists. If the request is in the dialogue mode, those lists are empty. In the non-dialogue mode, however, items such as a username and password can be obtained in this routine.

1. If another ACME agent is targeted by the $ACM client, return ACME$_CONTINUE (skip the rest of the steps in this ACME agents). This check is done by comparing ACME numbers of ACMEWQE$L_TARGET_ACME_ID and ACMEWQE$L_CURRENT_ACME_ID.
2. If another ACME agent has already declared as a Designated DOI agent, return ACME$_CONTINUE. This check is done by comparing ACME numbers of ACMEWQE$L_DESIGNATED_ACME_ID and ACMEWQE$L_CURRENT_ACME_ID.
3. Allocate and initialize the request-specific data structure (request_context).
4. Set the target status (whether the agent is targeted or not), and update target status value in request_context.
5. Process the common item list (principal name and password may be supplied by the $ACM client).
6. Process the ACME specific item list (ACME-specific items may be supplied by the $ACM client).
7. Return ACME$_CONTINUE.

ACME$CO_SYSTEM_PASSWORD

This callout routine is intended to be implemented only in the VMS agent. Other agents simply return ACME$_CONTINUE. In this routine, the VMS agent obtains a system password for authentication. This phase is associated with the ACME$_PASSWORD_SYSTEM item code with the $ACM system service.

ACME$CO_ANNOUNCE

This routine is invoked by the ACME server only for the Authenticate Principal request. It is implemented to display information to the user prior to the username prompt. The VMS agent displays the message defined with the SYS$ANNOUNCE logical. The example agent doesn't output any message by simply returning ACME$_CONTINUE. To display a message, however, implement the following procedure.

1. If this is the first time to enter this routine:
   - The agent sends the message to the $ACM client through ACMEKCV$CB_QUEUE_DIALOGUE().
   - Return ACME$_PERFORMDIALOGUE.
2. If this is the second time to enter this routine:
   - Return ACME$_CONTINUE.

ACME$CO_AUTOLOGON

If the ACME agent determines a principal name automatically (i.e. without user intervention), the mechanism is implemented in this callout routine. If this capability is necessary, follow the steps below.

1. Check the ACMEWQEFLG$V_PHASE_DONE flag. If this phase has been completed by another agent, return ACME$_CONTINUE.
2. Implement the agent's specific mechanism to determine a principal name.
3. Load the principal name string and length in WQE through ACMEKCV$CB_SET_WQE_PARAMETER.
4. Set the "phase done" flag (ACMEWQEFLG$K_PHASE_DONE) in WQE through ACMEKCV$CB_SET_WQE_FLAG().
5. Return ACME$_CONTINUE.

ACME$CO_PRINCIPAL_NAME

In the dialogue mode, this routine is responsible for prompting and obtaining a principal name string from the $ACM client.

1. Check the ACMEWQEFLG$V_PHASE_DONE flag. If this phase has been completed by another agent, return ACME$_CONTINUE.
2. If another ACME agent has already declared as a Designated DOI agent, return ACME$_CONTINUE (skip the rest of the steps in this ACME agents).
3. If this is the first time to enter this routine:
   - If a principal name has already been provided, return ACME$_CONTINUE.
   - If the principal name is not in the buffer:
     - Queue the principal name prompt to the $ACM client by calling ACMEKCV$CB_QUEUE_DIALOGUE().
     - Set a flag that indicates the request has already entered this routine once. Since returning ACME$_PERFORMDIALOGUE causes the request to be returned to this routine as soon as completing this dialogue, this flag is necessary to know that the next time the request enters this routine is the second time.
     - Return ACME$_PERFORMDIALOGUE.
4. If this is the second time to enter this routine:
   - If the ACME$_PRINCIPAL_NAME_IN item code is not in the common item list, return ACME$_FAILURE.
   - Load the principal name string and length in WQE through ACMEKCV$CB_SET_WQE_PARAMETER.
   - Set the "phase done" flag (ACMEWQEFLG$K_PHASE_DONE) in WQE through ACMEKCV$CB_SET_WQE_FLAG().
   - Return ACME$_CONTINUE.

ACME$CO_ACCEPT_PRINCIPAL

This callout routine examines a principal obtained in the ACME$_CO_PRINCIPAL_NAME routine. If the principal name is accepted by the principal name policy, this routine must declare as a Designated DOI agent. This is the last phase in which an ACME agent can set the Designated DOI status.

1. Check the ACMEWQEFLG$V_PHASE_DONE flag. If this phase has been completed by another agent, return ACME$_CONTINUE.
2. If another ACME agent has already declared as a Designated DOI agent, return ACME$_CONTINUE (skip the rest of the steps in this ACME agents).
3. Load the principal name and length in request_context from WQE. This is necessary when the ACME$CO_ACCEPT_PRINCIPAL routine was completed by another agent. In this situation, the principal and its length fields in the request_context is empty, whereas they are stored in the WQE.
4. Implement a policy to accept a principal name:
   - If the principal is not accepted by the policy, return ACME$_AUTHFAILURE.
5. Set the accepted principal name in WQE thorough ACMEKCV$CB_SET_WQE_PARAMETER().
6. Declare this is the Designated DOI agent.
7. Set the "phase done" flag (ACMEWQEFLG$K_PHASE_DONE) in WQE through ACMEKCV$CB_SET_WQE_FLAG().
8. Return ACME$_CONTINUE.

ACME$CO_MAP_PRINCIPAL

This routine specifies the VMS username in the SYSUAF file which corresponds to the principal name accepted in the previous phases. If the accepted principal name is identical to the VMS username, no mapping is necessary. Because VMS usernames are uppercased, uppercasing is required in the mapping process. For example, an accepted principal name john@openvms may be mapped to JOHN in the UAF record in this routine.

1. Check the ACMEWQEFLG$V_PHASE_DONE flag. If this phase has been completed by another agent, return ACME$_CONTINUE.
2. If this agent is not participating in this request (address of request_context is null), return ACME$_CONTINUE.
3. If this agent is not the Designated DOI agent, return ACME$_CONTINUE.
4. Implement the mapping policy.
5. Convert the principal string to uppercase. This conversion is necessary because the current ACME server doesn't uppercase the principal string. Without uppercasing, a lowercase/mixed-case principal string causes a mismatch against a VMS username, which is always uppercased.
6. Set the uppercased principal in WQE through ACMEKCV$CB_SET_WQE_PARAMETER().
7. Return ACME$_CONTINUE.

ACME$CO_VALIDATE_MAPPING

Optionally, an ACME agent can check the validity of the mapped username in this routine. The VMS agent always checks the mapped VMS username in this phase. If the mapped username doesn't exist in the SYSUAF file, the request is terminated by the VMS agent.

ACME$CO_ANCILLARY_MECH_1

This routine is used when the ACME agent collects additional information from the $ACM client. The example ACME agent doesn't implement this routine because no information other than a username and password is used. However, if an ACME agent uses other types of user credentials such as a cryptographic token, it is expected to obtain them in this callout routine or the other two ancillary mechanism routines (ACME$CO_ANCILLARY_MECH_2 and ACME$CO_ANCILLARY_MECH3).

ACME$CO_PASSWORD_1

In the dialogue mode, this routine is responsible for prompting and obtaining a password from the $ACM client.

1. Check the ACMEWQEFLG$V_PHASE_DONE flag. If this agent is not participating in this request or this phase has been completed by another agent (address of request_context is null), return ACME$_CONTINUE.
2. If the pre-authenticated flag (ACMEWQEFLG$V_PREAUTHENTICATED) has been set:
   - Return ACME$_AUTHFAILURE, or if the agent allows the pre-authenticated mode, return ACME$_CONTINUE.
3. If another ACME agent has already declared as a Designated DOI agent, return ACME$_CONTINUE (skip the rest of the steps in this ACME agents).
4. If this is the first time to enter this routine:
   - If a password has already been provided, return ACME$_CONTINUE.
   - If the password is not in the buffer:
     - The password prompt will be sent to the $ACM client through ACMEKCV$CB_QUEUE_DIALOGUE().
     - Set a flag that indicates the request has already entered this routine once. Since returning ACME$_PERFORMDIALOGUE causes the request to be returned to this routine as soon as completing this dialogue, this flag is necessary to know that the next time the request enters this routine is the second time.
     - Return ACME$_PERFORMDIALOGUE.
5. If this is the second time to enter this routine:
   - If the ACME$_PASSWORD_1 item code is not in the common item list, return ACME$_FAILURE.
   - Load the password string and length in WQE through ACMEKCV$CB_SET_WQE_PARAMETER.
   - Set the "phase done" flag (ACMEWQEFLG$K_PHASE_DONE) in WQE through ACMEKCV$CB_SET_WQE_FLAG().
   - Return ACME$_CONTINUE.

ACME$CO_ANCILLARY_MECH_2

This routine is used when the ACME agent collects any ACME-specific authentication information between the ACME$_PASSWORD_1 and ACME$_PASSWORD_2 phases. The example ACME agent doesn't implement this routine.

ACME$CO_PASSWORD_2

If the ACME agent needs a second password for authentication, this routine should be implemented. The implementation should follow the same procedures as for the ACME$CO_PASSWORD_1 routine.

ACME$CO_ANCILLARY_MECH_3

This routine is used when the ACME agent collects any ACME-specific authentication information between the ACME$_PASSWORD_1 and ACME$_PASSWORD_2 phases. The example ACME agent doesn't implement this routine.

ACME$CO_AUTHENTICATE

An ACME agent-specific authentication policy is implemented in this routine.

1. If this agent is not participating in this request (address of request_context is null), return ACME$_CONTINUE.
2. If this is not a Designated DOI agent, return ACME$_CONTINUE.
3. If the pre-authenticated flag (ACMEWQEFLG$V_PREAUTHENTICATED) has been set, return ACME$_AUTHFAILURE.
4. If the agent allows the pre-authenticated mode, return ACME$_CONTINUE.
5. Implement an authentication policy. The example agent implements a very simple authentication policy. It simply checks the first character of the password. If the password starts with 'a', the request is authenticated. Otherwise, authentication fails. In ACME agents for production use, it would be necessary to communicate with a file or database storing user credentials such as passwords.
    - If authentication succeeds, return ACME$_CONTINUE.
    - If authentication fails, return ACME$_AUTHFAILURE.

ACME$CO_MESSAGES

This routine is invoked by the ACME server only for the Authenticate Principal request. This is used to output any text message to the $ACM client between the ACME$CO_AUTHENTICATE and ACME$CO_AUTHORIZE phases. The implementation will be similar to other messaging callout routines such as ACME$CO_ANNOUNCE.

1. If this is the first time to enter this routine:
    - The message will be sent to the $ACM client through ACMEKCV$CB_QUEUE_DIALOGUE().
    - Return ACME$_PERFORMDIALOGUE.
2. If this is the second time to enter this routine, return ACME$_CONTINUE.

ACME$CO_AUTHORIZE

This routine is invoked by the ACME server only for the Authenticate Principal request. It performs authorization for the authenticated request. The request is authorized based on the ACME agent-specific authorization policy. Access restrictions such as privileges, modes of operation, and account duration are examined to grant the authorization. If there is no authorization policy with the ACME agent, this callout routine is optional.

ACME$CO_NOTICES

This routine is invoked by the ACME server only for the Authenticate Principal request. This is used to output a long text message to the $ACM client after successfully completing the authentication and authorization phases. For example, the VMS agent displays a message defined with SYS$WELCOME. The implementation will be similar to other messaging callout routines such as ACME$CO_ANNOUNCE and ACME$CO_MESSAGES.

1. If this is the first time to enter this routine:
    - The message will be sent to the $ACM client through ACMEKCV$CB_QUEUE_DIALOGUE().
    - Return ACME$_PERFORMDIALOGUE.
2. If this is the second time to enter this routine:
    - Return ACME$_CONTINUE.

ACME$CO_LOGON_INFORMATION

This routine is invoked by the ACME server only for the Authenticate Principal request. This is used to output text information to the $ACM client after the authorization phase. The difference from the ACME$CO_NOTICES callout routine is that this routine displays short, critical logon information. This will be the final output text after authentication and authorization. The VMS agent uses this phase to display last logon time and number of logon failures. The implementation will be similar to other messaging callout routines such as ACME$CO_ANNOUNCE and ACME$CO_MESSAGES.

1. If this agent is not participating in this request (address of request_context is null), return ACME$_CONTINUE.
2. If this is not a Designated DOI agent, return ACME$_CONTINUE.
3. If this is the first time to enter this routine:
    - If dialogue is not possible, skip the rest of this routine (return ACME$_CONTINUE).
    - Send a logon message to the $ACM client through ACMEKCV$CB_QUEUE_DIALOGUE().
    - Return ACME$_PERFORM DIALOGUE.
4. If this is the second time to enter this routine, return ACME$_CONTINUE.

ACME$CO_NEW_PASSWORD_1

This callout routine is mainly for change-password requests. The only situation in which this routine is used for authenticate-principal requests is when the password has expired. It prompts and obtain a new password for the $ACM client. The implementation is similar to the ACME$CO_PASSWORD_1 routine.

1. Check the address of request_context and ACMEWQEFLG$V_PHASE_DONE flag. If this agent is not participating in this request or this phase has been completed by another agent, return ACME$_CONTINUE.
2. If the ACMEWQEFLG$V_SKIP_NEW_PASSWORD flag is set, return ACME$_CONTINUE.
3. If this is not a Designated DOI agent, return ACME$_CONTINUE.
4. If the request in the AUTHENTICATE_PRINCIPAL mode and the PasswordExpired flag is not set, return ACME$_CONTINUE. As mentioned above, unless a password is expired, an authenticate-principal doesn't need this phase.
5. If this is the first time to enter this routine:
    - If a new password has already been provided, return ACME$_CONTINUE.
    - If the new password is not in the buffer:
        - If either input or noecho can't be performed, return ACME$_INSFDIALSUPPORT.
        - The new password prompt will be sent to the $ACM client through ACMEKCV$CB_QUEUE_DIALOGUE().
        - Set a flag that indicates the request has already entered this routine once. Since returning ACME$_PERFORMDIALOGUE causes the request to be returned to this routine as soon as completing this dialogue, this flag is necessary to know that the next time the request enters this routine is the second time.
        - Return ACME$_PERFORMDIALOGUE.
6. If this is the second time to enter this routine:
    - If the ACME$_NEW_PASSWORD_1 item code is not in the common item list, return ACME$_FAILURE.
    - Load the new password string and length in WQE through ACMEKCV$CB_SET_WQE_PARAMETER.
    - Set the "phase done" flag (ACMEWQEFLG$K_PHASE_DONE) in WQE through ACMEKCV$CB_SET_WQE_FLAG().
    - Return ACME$_CONTINUE.

ACME$CO_QUALIFY_PASSWORD_1

This callout routine implements a policy to accept a new password. The length and special/numerical characters in a new password are usually examined.

1. Check the address of wqe_context and ACMEWQEFLG$V_PHASE_DONE flag. If this agent is not participating in this request or this phase has been completed by another agent, return ACME$_CONTINUE.
2. If the ACMEWQEFLG$V_SKIP_NEW_PASSWORD flag is set, return ACME$_CONTINUE.

3. If this is the first time to enter this routine:
    - If this is not a Designated DOI agent, return ACME$_CONTINUE.
    - If the request in the AUTHENTICATE_PRINCIPAL mode and the PasswordExpired flag is not set, return ACME$_CONTINUE.
    - If the new password is not in the buffer, return ACME$_FAILURE.
    - If the new password is in the buffer:
        - Implement a policy to accept or reject a new password.
            - If the new password is accepted:
                - Update the new password in the password database/file.
                - Set the "phase done" flag (ACMEWQEFLG$K_PHASE_DONE) in WQE through ACMEKCV$CB_SET_WQE_FLAG().
                - return ACME$_CONTINUE.
            - If the new password is rejected:
                - A message indicating that the new password was rejected is sent to the $ACM client through ACMEKCV$CB_QUEUE_DIALOGUE().
                - Return ACME$_PERFORMDIALOGUE.
4. If this is the second time to enter this routine (only when the password was rejected):
    - Set the "phase done" flag (ACMEWQEFLG$K_PHASE_DONE) in WQE through ACMEKCV$CB_SET_WQE_FLAG().
    - To terminate the request, return ACME$_FAILURE. However, in most production systems, it is a common practice to ask the user to enter another new password. To implement this way, use ACME$_RETRYPWD. This return value causes the request to re-enter the previous routine, ACME$CO_NEW_PASSWORD_1.

ACME$CO_NEW_PASSWORD_2

If the ACME agent needs a second password for authentication, this routine should be implemented. The implementation should follow the same procedures as for the ACME$CO_NEW_PASSWORD_1 routine.

ACME$CO_QUALIFY_PASSWORD_2

If the ACME agent needs a second password for authentication, this routine should be implemented. The implementation should follow the same procedures as for the ACME$CO_QUALIFY_PASSWORD_1 routine.

ACME$CO_ACCEPT_PASSWORDS

The purpose of this routine is to prepare for the next routine. It checks the availability of the password database and other resources (e.g. network connections to the database) for the operation in the ACME$CO_SET_PASSWORD callout routine. This is to minimize the possibility of failure in the next phase.

ACME$CO_SET_PASSWORD

The role of this routine is to update the password file/database for the ACME agent. Because the example ACME agent doesn't have its password database, it is not implemented. If the password update fails in this phase, this may cause discrepancies of saved passwords in password databases of multiple ACME agents. For example, if ACME agent B fails to update a password after ACME agent A has successfully updated the password in its password file, the passwords saved for agent A and B are inconsistent. To avoid or minimize such a situation, it is recommended to check the availability of the password database/file in the previous routine, ACME$CO_ACCEPT_PASSWORDS.

ACME$CO_CREDENTIALS

This routine is invoked by the ACME server only for the Authenticate Principal request. This must be implemented in any DOI agent, which always issues credentials to the $ACM client. Only the authenticate-principal requests invoke this routine. Since credentials are passed to and stored in the persona extension image, the definition of the credential data structure must be consistent with the one in the persona extension code.

1. Check the address of request_context. If this agent is not participating in this request, return ACME$_CONTINUE.
2. If this is not a Designated DOI agent, return ACME$_CONTINUE.
3. If the credential is not requested by the $ACM client, return ACME$_CONTINUE (skip the rest).
4. Send the credential to the $ACM client through ACMEKCV$CB_ISSUE_CREDENTIALS().
5. Return ACME$_CONTINUE.

ACME$CO_FINISH

This is the last routine in the ACME agent. Cleanup operations such as memory deallocation and file I/O closure are implemented in this routine.

1. If this agent is not participating in this request (address of request_context is null), return ACME$_NORMAL.
2. Deallocate request_context.
3. Return ACME$_NORMAL.

## Event and Query callout routines

The optional routines ACME$CO_EVENT and ACME$CO_QUERY are not executed in the course of the dialogue request processing. They are invoked when an application calls the $ACM system service with function codes specifying those operations. Unlike the request callout functions, these routines are always executed in the targeted mode. The $ACM client targets a specific ACME agent with the ACME$CO_EVENT or ACME$CO_QUERY call, so the ACME server calls no other agents for this callout routine. Therefore, you don’t have to consider configurations and scenarios with other ACME agents. This is the reason no specific and required internal steps exist for these routines. In both routines, item codes from the $ACM client can be found in the item list, and a specific event or information query should be implemented for each supported item code.

ACME$CO_EVENT

This routine, ACME$CO_EVENT, is implemented for event-related operations of an ACME agent. The ACME$_FC_EVENT function code with SYS$ACM in the application invokes this callout routine. Although implementing this routine is optional, it is appropriate to handle discrete events such as auditing and error checking. In this routine, the following input and output item codes are processed.

- ACME$_EVENT_DATA_IN

  The ACME$_EVENT_DATA_IN item code is an input item code. It specifies the buffer containing information applicable to an event operation. The meaning of this data is specific to the domain of interpretation for which it is used.
- ACME$_EVENT_TYPE

  The ACME$_EVENT_TYPE item code is an input item code. It specifies the type of event being reported. The buffer must contain a longword value. Interpretation of the value is specific to the domain of interpretation to which the event is being reported.
- ACME$_SERVER_NAME_IN

  Specifies the Event Server to which an Event should be directed. The meaning of this item is specific to the target domain of interpretation.
- ACME$_EVENT_DATA_OUT

  The ACME$_EVENT_DATA_OUT item code is an output item code. It specifies the buffer to receive information returned from an event operation. The meaning of this data is specific to the domain of interpretation for which it is used.
- ACME$_SERVER_NAME_OUT

  Reports the Event Server to which an Event was directed. The meaning of this item is specific to the target domain of interpretation.

ACME$CO_QUERY

This routine is used to provide the ACME agent's information with a $ACM client for function code ACME$_FC_QUERY. In this routine, it is necessary to parse values in the three input item codes (ACME$_QUERY_TYPE, ACME$_QUERY_KEY_TYPE, and ACME$_QUERY_KEY_VALUE) and return data with the output item code, ACME$_QUERY_DATA.

- ACME$_QUERY_TYPE

  The ACME$_QUERY_TYPE item code is an input item code. It specifies the type of data to be returned in the buffer described by the corresponding ACME$_QUERY_DATA item code. The ACME$_QUERY_TYPE item code requires that an ACME$_QUERY_DATA item code immediately follow it in the item list.
- ACME$_QUERY_DATA

  The ACME$_QUERY_DATA item code is an output item code. It specifies the buffer to receive the data returned from the query operation relating to the corresponding ACME$_QUERY_TYPE item code. The ACME$_QUERY_DATA item code requires that an ACME$_QUERY_TYPE item code immediately precede it in the item list.

- ACME$_QUERY_KEY_TYPE

  The ACME$_QUERY_KEY_TYPE item code is an input item code. It specifies the key type for establishing the context of a query operation. The key format is specific to the ACME agent to which the call is directed. An ACME$_QUERY_KEY_TYPE item requires that an ACME$_QUERY_KEY_VALUE item immediately follow it in the item list.
- ACME$_QUERY_KEY_VALUE

  The ACME$_QUERY_KEY_VALUE item code is an input item code. It specifies the key data for establishing the context of a query operation. An ACME$_QUERY_KEY_VALUE item requires that an ACME$_QUERY_KEY_TYPE item immediately precede it in the item list.

## Implement a Persona Extension

The persona extension's role with the ACME subsystem is to store authentication credentials issued by its corresponding ACME agent. As mentioned earlier, an ACME agent issuing credentials requires its corresponding persona extension (for the mechanism of issuing credentials, refer to Figure 6). If the ACME agent doesn't issue any credentials, it is not necessary to develop a persona extension for the ACME agent. If the ACMEKCV$CB_ISSUE_CREDENTIALS callback function is called in the ACME$CO_CREDENTIALS routine, its persona extension must be installed with the ACME agent. This section provides introductory guidelines for development of a persona extension image for ACME agent developers.

### Introduction to a persona and persona extension

A persona is a group of kernel-based, protected data structures storing a user’s security profile for a process/thread. Every process in the system has at least one persona, called the natural persona. The natural persona is created during process creation. The primary data structure of a persona is the Persona Security Block (PSB). The PSB contains the security profile, including UIC, system rights chains, privileges, account name, user name, auditing flags, and counters. As shown in Appendix C in the *ACME Developer's Guide*, a persona is composed of several data structures:

- Persona Security Block (PSB)
- Persona Extension Block Array (PXB_ARRAY)
- Persona Extension Block (PXB)
- Persona Extension Creation Flags (PXB_Flags)
- Persona Extension Dispatch Vector (PXDV)
- Persona Extension Registration Block (PXRB)
- Persona Extension Create Flags (Create_Flags)
- Persona Delegation Block (DELBK)
- PSB Ring Buffer (PSBRB)
- Persona Security Block Array (PSB_ARRAY)

A persona extension is a data structure called Persona Extension Block (PXB). Figure 7 illustrates how a PXB is linked to the PSB data structure. All persona extensions except the VMS extension are indexed in the PXB_ARRAY structure. Each extension is a PXB data structure that a system manager can load the agent-specific persona extension as an executive image. The Configure ACME section addresses the installation of a persona extension, and Appendix B provides DCL and SYSMAN commands for building and installing a persona extension image. In the following lines in this section, we will focus on development of a persona extension image.

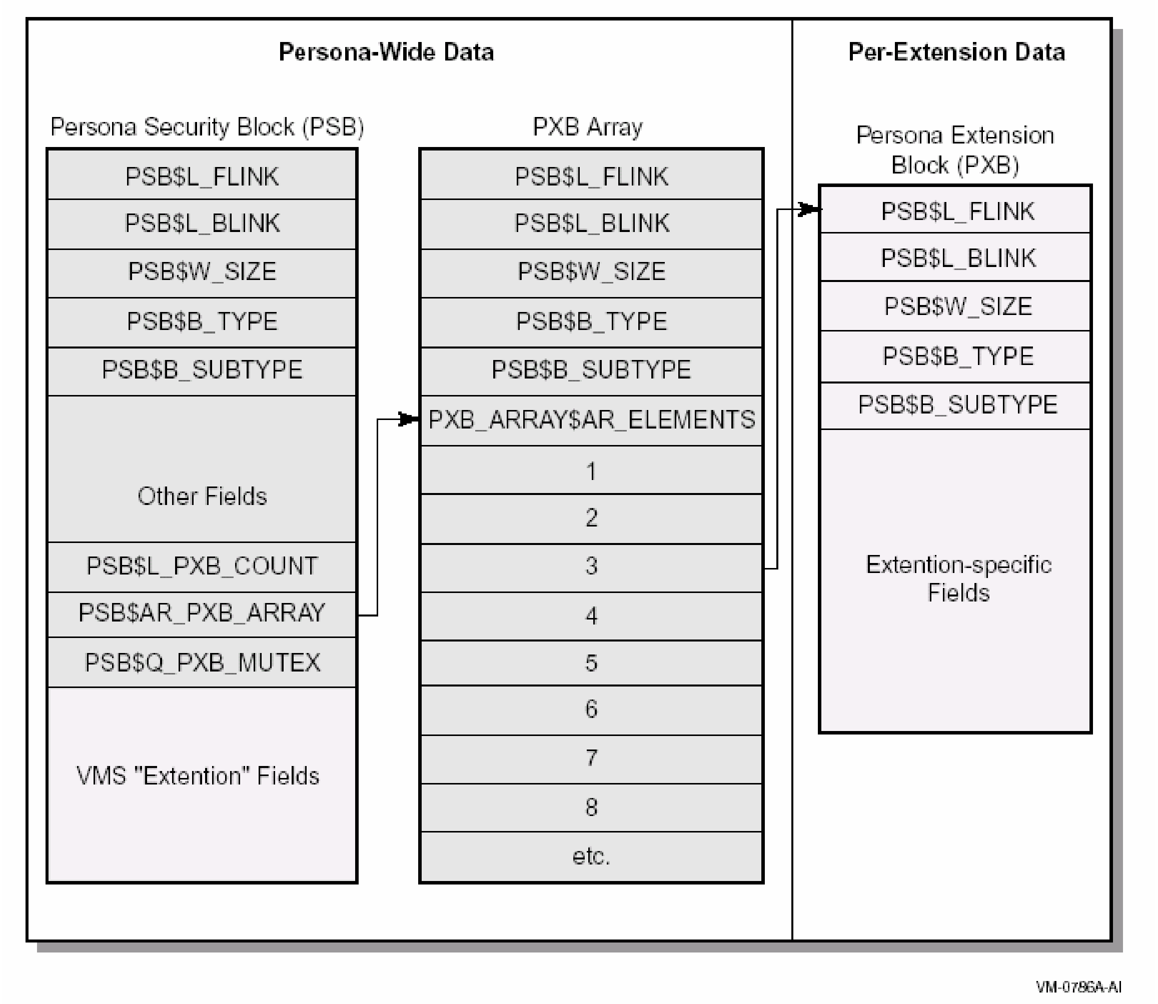


**Figure 7: Some persona data structures (credentials are stored in "Extension-specific Fields")**

## Start implementing a person extension

The developer must follow the programming interface for a persona extension—it is different from the ACM agent's interface. The primary task of persona extension development is to implement persona extension routines. The instructions for those routines will be provided in this section.

The example persona extension code (ACME_PERSONA_EXT.C) can be found in the SYS$EXAMPLES directory. It is strongly recommended that you refer to the file while reading instructions in this section. For building and installing the example persona extension program, refer to Appendix B.

### Define data structures in the persona extension program

As listed above, the entire persona extension system uses several data structures. In the persona extension program, only the credential and Persona Extension Block (PXB) data structures must be defined — both are specific to the persona extension.

#### The credential data structure

The definition of this credential data structure must be exactly the same as the definition in the ACME agent. Any differences between the data structure definitions cause problems, including a difference as small as a buffer size.

#### Persona Extension Block (PXB)

PXB is the data structure to store the persona extension data. The header fields are pre-defined and must always be defined in a PXB. If the PXB is defined as a struct pxb_p1 as in the example code, its header fields are defined as follows:

- struct pxb_p1 *pxb_p1$l_flink;
- struct pxb_p1 *pxb_p1$l_blink;
- unsigned short int pxb_p1$w_size;
- unsigned char pxb_p1$b_type;
- unsigned char pxb_p1$b_subtype;

The remaining fields in the PXB are the credentials. All fields in the credential data structure should be appended after the header part.

For more information about PXB, refer to Section C.3 in the *ACME Developer's Guide*.

### Implement persona extension routines

#### Initialization routine (mandatory)

int persona_ext_initialize ();

The role of this routine is to declare the addresses of other person extension routines. All routines in the persona extension image are registered during a system boot. NSA$REGISTER_PSB_EXTENSION() in this routine performs this registration. Two arguments are passed to this initialization function. The first argument is a descriptor containing a name of the extension image. The second argument is a Persona Extension Dispatch Vector (PXDV). Addresses of persona extension routines must be set to their corresponding fields (PXDV$A_CREATE, PXDV$A_CLONE, PXDV$A_DELEGATE, PXDV$A_DELETE, PXDV$A_MODIFY, PXDV$A_QUERY, and PXDV$A_MAKE_TLV) in PXDV. The details about NSA$REGISTER_PSB_EXTENSION() are available in Section 12.5 in the *ACME Developer's Guide*.

Assuming that the persona extension routines are named as persona_ext_create() and so forth, and pxdv is the PXDV declared in this routine, we can register the routines as follows in this initialize routine.

```
pxdv.pxdv$a_create   = (void *) persona_ext_create;     /* required */
pxdv.pxdv$a_clone    = (void *) persona_ext_clone;      /* optional */
pxdv.pxdv$a_delegate = (void *) persona_ext_delegate;   /* optional */
pxdv.pxdv$a_delete   = (void *) persona_ext_delete;     /* required */
pxdv.pxdv$a_modify   = (void *) persona_ext_modify;     /* required */
pxdv.pxdv$a_query    = (void *) persona_ext_query;      /* required */
pxdv.pxdv$a_make_tlv = (void *) persona_ext_make_tlv;   /* required */

status = nsa$register_psb_extension(&p1_desc, &p1_pxdv);
```

Create routine (mandatory)

```
int persona_ext_create (
    PSB *psb,
      PXB **pxb,
      P1_CREDENT *credential,
      unsigned int credential_size
    );
```

This routine creates a new persona extension. Specifically, this routine allocates a PXB in the non-paged pool and sets credential values in the PXB. For PXB allocation in the non-paged pool, EXE_STD$ALONONPAGED() is used. Credentials are copied into the credential data structure defined in this program.

Clone routine (optional)

```
int persona_ext_clone (
      PSB *psb,
      PXB *pxb,
      PXB *new_pxb
      );
```

This routine copies an existing persona extension in the context of the current process. An OpenVMS application can request this operation by calling the $PERSONA_CLONE system service. In the Clone routine, another extension-specific PXB is allocated with EXE_STD$ALONONPAGED(), and the system PXB is copied to the newly allocated PXB. Before returning SS$_NORMAL, the address of the new PXB must be passed to the new PXB provided as the third argument of this routine. The implementation of this routine is optional. If it is not implemented, no persona extension is created in the new persona during the $PERSONA_CLONE operation.

Delegate routine (optional)

```
int persona_ext_delegate (
        PSB *psb,
        PXB *pxb,
        int unused,
        PXB *new_pxb,
        PSB *new_psb
        );
```

This routine copies an existing persona extension into a different process. This routine is invoked by the $PERSONA_DELEGATE system service in an application program. The implementation is similar to the clone routine. A new extension-specific PXB is allocated with EXE_STD$ALONGPAGED(). After the original PXB is copied to the new PXB data structure, the address of the new PXB is given to the routine's forth argument.

Delete routine (mandatory)

```
int persona_ext_delete (
        PSB *psb,
        PXB *pxb
      );
```

In this routine, the PXB data structure is deleted with EXE_STD$DEANONPAGED(). In general, the implementation of this routine is simple. Unless the PSB or PBX is empty, the whole PXB is deallocated with EXE_STD_DEANONPAGED(). But the way of deletion depends on the clone and delegate implementation.

Modify routine (mandatory)

```
int persona_ext_modify (
        PSB *psb,
        PXB *pxb,
        int itemcode,
        char *buf_addr,
        int buf_len
        );
```

This routine modifies data in the persona extension when the application calls the $PERSONA_MODIFY system service. In this routine, the item code provided as a third argument is modified to the value stored in the buffer in the forth argument. If the input item code is not supported, SS$_BADITMCOD must be returned. If the modification operation is successful, SS$_NORMAL will be the return value.

Query routine (mandatory)

```
int persona_ext_query (
        PSB *psb,
        PXB *pxb,
        int itemcode,
        char *buf_addr,
        int buf_len,
        int *ret_len,
        int queryflg,
        struct dsc$descriptor_s *dsc
        );
```

This routine retrieves a credential field requested by the application calling the $PERSONA_QUERY system service. The following item codes must be supported:

- ISS$_COMMON_FLAGS
- ISS$_DOI
- ISS$_COMMON_USERNAME
- ISS$_DOMAIN
- ISS$_COMMON_PRINCIPAL
- ISS$_COMMON_ACCOUNT
- ISS$_EXTENSION

If the input item code is not supported in this routine, SS$_BADITMCOD will be returned.

When queryflag is turned on, this routine compares the input data to the one in PXB. For every item code, the flag is checked first, and then both "compare" and "retrieve" modes should be implemented. The following lines are an example.

```
case ISS$_DOI:
    if (queryflg == COMPARE) {
                /* compare buffer */
                if (strcmp(buf_addr,  pxb_p1_p->pxb_p1$domain) != 0)
                        status = FALSE;
                else
                        status = TRUE;
    }else{
                if (buf_len >= sizeof(pxb_p1_p->pxb_p1$domain)){
                        strncpy(buf_addr, pxb_p1_p->pxb_p1$doi,
                                    sizeof(pxb_p1_p->pxb_p1$doi));
                        ret_len = sizeof(pxb_p1_p->pxb_p1$doi);
                }else
                        status = SS$_BADBUFLEN;
    }
    break;
```

Make_TLV routine (mandatory)

```
int persona_ext_make_tlv (
    PSB *psb,
    PXB *pxb,
    int itemcode,
    char *buf_addr,
    int buf_len,
    int *ret_len,
    int flags
    );
```

This routine is available with the intention to package credentials into a position-independent string for batch jobs. However, implementation of this routine is rarely required at this point. Thus, it is sufficient to simply return SS$_UNSUPPORTED in most cases.

## Configure ACME — Put all the components together

Now we have all of the pieces for authentication with the ACME subsystem. For ACME agents to work properly, careful configuration of the ACME components, which are explained in the previous sections, is necessary. Follow the steps below.

### Installing the persona extension image

The installation of a persona extension requires more than just copying the image into SYS$LOADABLE_IMAGES. The Alpha image should be tested with CHECK_SECTIONS.COM, a utility to check that the executive image is loadable. In addition, run a SYSMAN command to load the persona extension image, and then execute VMS$SYSTEM_IMAGES.COM to generate a new system image data file. After all this is done, reboot the system. The DCL and SYSMAN commands for these operations can be found in Appendix B.

### Configuring the SYSUAF flags and security policy bits

If your ACME agent performs authentication, you must set either the EXTAUTH flag in the SYSUAF record for each VMS account to use external authentication or the IGNORE_EXTAUTH security policy bit. If either flag is not set in the user account and you enable the VMS agent first, the VMS agent handles authentication requests.

For an untargeted $ACM call, an ACME agent including the VMS agent becomes the Designated DOI agent if it declares DOI through the ACMEKCV$CB_SET_DESIGNATED_DOI() callback function.

When an ACME agent (other than the VMS agent) becomes the Designated DOI agent and performs authentication, the VMS agent synchronizes the password with the mapped user account in the SYSUAF file. For example, after a user, Mike, is authenticated by ACME agent X in which Mike's password is "password_x," Mike's password in SYSUAF will be updated to password_x. To disable this automatic password synchronization, the DISPWDSYNCH flag can be set in Mike's account in SYSUAF.

To enforce the same effects as the EXTAUTH and DISPWDSYNCH flags for all user accounts in the entire system, the IGNORE_EXTAUTH and GUARDS_PASSWORD bits in the SECURITY_POLICY system parameter bitmask can be used. The SECURITY_POLICY bit can be modified by the SET SECURITY command with the SYSGEN utility. After changing a value in the SECURITY_POLICY bit, the system must be rebooted to enforce the new value.

For more information about those flags and bits, refer to Section 1.10 in the *ACME Developer's Guide*. Values of the SECURITY Policy bits can be found in Chapter 7 in the *OpenVMS Guide to System Security*.

### Configuring the ACME subsystem

The ACME agent images must be copied to SYS$LIBRARY, and then the ACME subsystem can be configured with SET SERVER ACME commands shown below. There are several states of the ACME subsystem, and the SHOW SERVER ACME command displays the current state.

Figure 5 shows the ACME subsystem's state transitions with SET SERVER ACME commands.

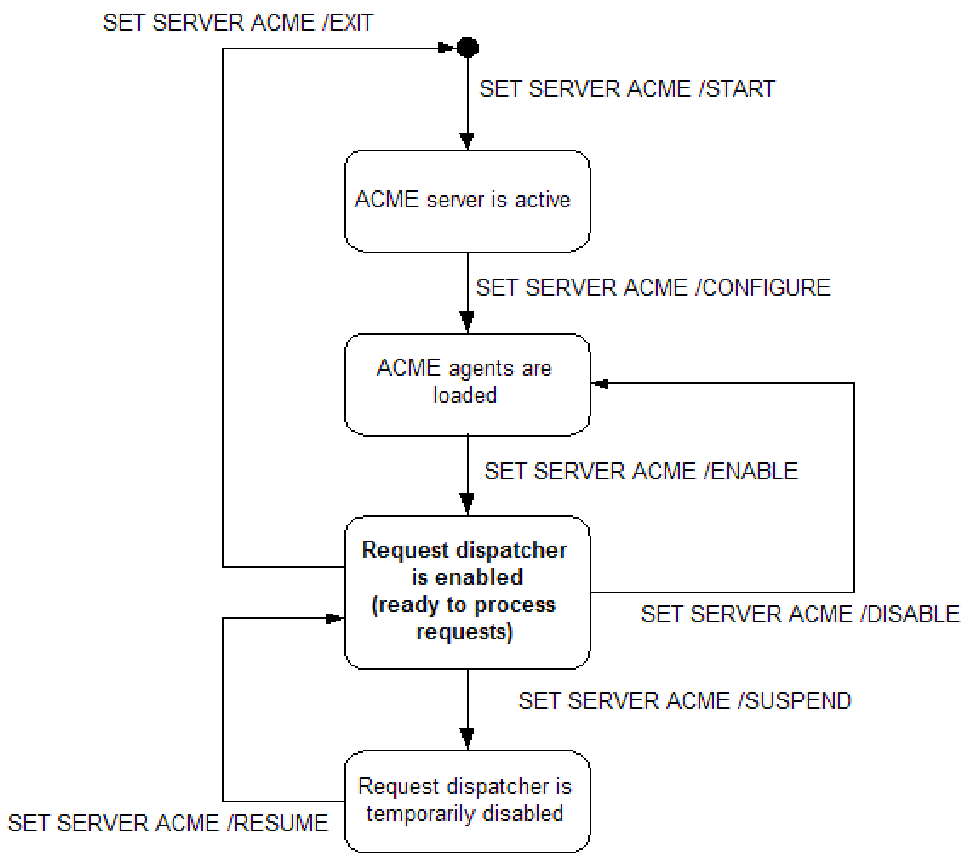


**Figure 5: SET SERVER ACME commands and states of the ACME subsystem**

$ SET SERVER ACME/START

This command is the first step—it starts the ACME server.

$ SET SERVER ACME/CONFIGURE=(NAME=*agent_name*, CREDENTIAL=*credential_name*)

After the ACME server gets started, this command can be run to load an ACME agent.

To load the VMS ACME agent:

$ SET SERVER ACME/CONFIGURE=(NAME=VMS, CREDENTIAL=VMS)

To load the example ACME agent (ACME_EXAMPLE_DOI) with the example persona extension (P1):

$ SET SERVER ACME/CONFIGURE=(NAME=ACME_EXAMPLE_DOI, CREDENTIAL=P1)

```
$ SET SERVER ACME/ENABLE
```

This command activates ACME agents that have already been loaded in the ACME subsystem. The order of ACME agents is specified by this command. The following commands demonstrate the order of the VMS and example ACME agents.

The VMS agent is the first agent.

```
$ SET SERVER ACME/ENABLE=NAME=(VMS, ACME_EXAMPLE_DOI)
```

The example agent is the first agent.

```
$ SET SERVER ACME/ENABLE=NAME=(ACME_EXAMPLE_DOI, VMS)
```

```
$ SET SERVER ACME/DISABLE
```

This command disables the ACME server and agents.

```
$ SET SERVER ACME/EXIT
```

Run this command to stop the ACME server. Once this command is executed, all of the ACME agents are unloaded because the ACME subsystem stops.

## Perform a request from the $ACM application

Once the ACME subsystem has been configured, it is essential to send requests from the $ACM application that will be used in the production environment. If the ACME agent issues credentials, the $ACM application must be capable of handling them. Ensure that the proper $ACM application is used for testing. The ACMEUTIL example program is provided to test the example ACME agent, which issues a simple username and principal as credentials,. Note that Telnet and ftp can be used to send authentication requests, but they are not directly calling the $ACM system service. The authentication with Telnet, ftp, and SET HOST is invoked by the $ACM service in the ACME LOGINOUT image.

Sending an untargeted request

```
$ ACMEUTIL AUTH /PERSONA/DIALOGUE=(INPUT,NOECHO)
```

Sending a request targeting the ACME_EXAMPLE_DOI agent

```
$ ACMEUTIL AUTH /PERSONA /DIALOGUE=(INPUT,NOECHO) /DOMAIN=
ACME_EXAMPLE_DOI
```

Sending a request targeting the VMS agent

```
$ ACMEUTIL AUTH /PERSONA/DIALOGUE=(INPUT,NOECHO)/DOMAIN=VMS
```

## Summary

The ACME subsystem provides a new authentication environment on OpenVMS. To enforce new authentication policies, we can load ACME agents in a "plug-in" manner. Many OpenVMS system managers will benefit from the flexibility of this new capability. As OpenVMS developers, we can create new ACME agents for new authentication policies.

The major task for developing an ACME agent is to implement every callout routine for its authentication policy. By referring to the steps in every callout routine in the Implement an ACME Agent section as well as the example ACME agent's source, ACME agent developers will have clearer ideas about how to implement ACME agents.

If the ACME agent issues credentials, it is also necessary to develop its persona extension. A persona extension is an executive image that securely stores credentials for the $ACM application process/thread. The Implement a Persona Extension section provides comprehensive steps for developing a persona extension. It is recommended that you read this part with the source code of the sample persona extension.

Finally, after the ACME agent and its persona extension become available, ACME developers and OpenVMS system managers have to know how to install and configure all the components for testing and setting up production environments. Comprehensive steps for installing and configuring an ACME agent and persona extension are available in the Configure ACME section.

## For more information

- *ACME Developer's Guide* (SYS$HELP:ACME_DEV_GUIDE.PDF)
- Chapter 33. Authentication and Credential Management (ACM) System Service, *OpenVMS Programming Concept Manual* (http://h71000.www7.hp.com/doc/731FINAL/5841/5841pro_contents_010.html#toc_chapter_33)
- *OpenVMS Guide to System Security* (http://h71000.www7.hp.com/doc/732FINAL/aa-q2hlg-te/aa-q2hlg-te.PDF)
- *HP OpenVMS System Services Reference Manual*: GETUTC–Z (http://h71000.www7.hp.com/doc/732FINAL/DOCUMENTATION/PDF/aa-qsbnf-te.PDF)

## Acknowledgements

While I was writing this article, I received tremendous support from many people in the OpenVMS Engineering Group. I spent lots of time with the members of the ACME project, Rick Barry, John Harney, Doug Fyfe, and Barbara Thomson, about the details of the ACME and persona extension architectures. The feedback from Rick, Barbara and John for the drafts was invaluable. I also thank the editor of this article, Kathleen Johnson, for her extensive review of this article.

## Appendix A: How to Build and Set Up the Example ACME Agent (ACME_EXAMPLE_DOI_ACME.C)

1. Compile the message file (if it hasn't been done)

```
$ MESSAGE ACME_EXAMPLE_DOI_MSG.MSG
```

   - This command creates ACME_EXAMPLE_DOI_MSG.OBJ, which will be linked later

2. Build (compile and link) the example ACME agent
   (This command creates VMS$ACME_EXAMPLE_DOI_ACMESHR.EXE)

```
$ @ACME_EXAMPLE_DOI_BUILD.COM
```

3. Copy the example ACME agent image to SYS$LIBRARY

```
$ COPY VMS$ACME_EXAMPLE_DOI_ACMESHR.EXE SYS$LIBRARY
```

## Appendix B: How to Build and Set Up the Persona Extension Example (ACME_PERSONA_EXT.C)

1. Build (compile and link) the example persona extension image
   (This command creates P1_EXT.EXE)

```
$ @ACME_PERSONA_BUILD.COM
```

2. Test the example persona extension image with SYS$ETC:CHECK_SECTIONS.COM (on OpenVMS Alpha only)

```
$ @SYS$ETC:CHECK_SECTIONS.COM P1_EXT.EXE
```

3. Copy the example persona extension image to SYS$LOADABLE_IMAGES

```
$ COPY P1_EXT.EXE SYS$LOADABLE_IMAGES
```

4. Install the example persona extension image
   (the image is P1_EXT.EXE, and the product name is ACMETEST)

```
$ MCR SYSMAN
SYSMAN> SYS_LOADABLE ADD/LOG ACMETEST P1_EXT

$ @SYS$UPDATE:VMS$SYSTEM_IMAGES.COM
```

5. Reboot the system

```
$ @SYS$SYSTEM:SHUTDOWN
```

During reboot, an error message appears if the persona extension Image is not loaded. If you don't see the error message, the image should be loaded properly.

To verify:

```
$ ANALYZE /SYSTEM
SDA> SHOW EXECUTIVE P1_EXT
```

## Appendix C: How to Set Up the ACME Agent and Persona Extension After Reboot

1. Start the ACME server

```
$ SET SERVER ACME/START/LOG
```

2. Load the VMS ACME agent (this agent must be always loaded)

```
$ SET SERVER ACME/CONFIGURE=(NAME=VMS,CREDENTIAL=VMS)
```

3. Load the example agent with the example persona extension (P1)

```
$ SET SERVER ACME/CONFIGURE=(NAME=ACME_EXAMPLE_DOI,CREDENTIAL=P1)
```

4. Enable the agents (the order is the example agent is first)

```
$ SET SERVER ACME/ENABLE=NAME=(ACME_EXAMPLE_DOI,VMS)
```

## Appendix D: How to Test the Example Agent from the ACMEUTIL Client Program (in SYS$EXAMPLES)

ACMEUTIL is a DCL utility program executing the $ACM[W] system service for authentication and change-password requests.

Authenticate Principal request (targeted call, credential is requested)

```
$ acmeutil auth /persona/dialogue=(input,noecho)/domain=acme_example_doi
```

Change Password request (Untargeted call)

```
$ acmeutil change dialogue=(input,noecho)
```

For more information about this program, see ACMEUTIL_SETUP.COM in SYS$EXAMPLES.

## Appendix E: How to Install the ACME LOGINOUT Image

The ACMELOGIN kit is provided to install versions of LOGINOUT.EXE and SETP0.EXE that are modified to use the SYS$ACM system service. Since these images use SYS$ACM, they will use the authentication policies provided by the ACME agents that have been configured on your system including user-defined agents.

Note: It is recommended that you first test your ACME agent using the ACMEUTIL utility described earlier in this document before installing the ACMELOGIN kit.

Three PCSI kits are contained in the BACKUP saveset SYS$UPDATE:ACME_DEV_KITS.BCK. Restore the PCSI kits to your default directory using BACKUP:

```
$ BACKUP SYS$UPDATE:ACME_DEV_KITS.BCK/SAVE *.*
```

This will create three PCSI kits:

```
DEC-AXPVMS-V732_ACMELOGIN-V0100--4.PCSI      (ACMELOGIN V1.0 patch kit)
```

The ACMELOGIN kit contains modified versions of LOGINOUT.EXE and SETP0.EXE that use the SYS$ACM system service to perform authentication and password changes.

```
DEC-AXPVMS-V732_LOGIN-V0100--4.PCSI      (LOGIN V1.0 patch kit)
```

The LOGIN kit contains the original LOGINOUT.EXE and SETP0.EXE images that were shipped with this release. You can install this kit to restore the original versions of these files if you've previously installed the ACMELOGIN kit for development and testing.

Note: If you previously installed any ECO kits that modified LOGINOUT.EXE or SETP0.EXE, you will need to re-apply those ECO kits after restoring the original images using the LOGIN kit.

```
DEC-AXPVMS-V732_ACMELDAP-V0100--4.PCSI  (ACMELDAP V1.0 patch kit)
```

The ACMELDAP kit contains the LDAP ACME sharable image, management tools, a startup file, CLD file, and initialization template, as well as the .PS, .TXT and .HTML documentation.

Install the kit using the Polycenter Software Installation Utility from a privileged account.

To install the ACME LOGINOUT image:

```
  $ PRODUCT INSTALL V732_ACMELOGIN
```

To install the traditional LOGINOUT image:

```
  $ PRODUCT INSTALL V732_LOGIN
```

## Appendix F: ACME Terminology

### 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 cannot issue credentials. It cannot be the target of an $ACM call. An auxiliary agent logically works in conjunction with the designated DOI agent.

### 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:

ACME$_LOGON_TYPE

ACME$_AUTH_MECHANISM

ACME$_NEW_PASSWORD_FLAGS

ACME$_PRINCIPAL_NAME_IN

ACME$_PASSWORD_1

### Cooperative model

For untargeted $ACM calls, an DOI agent in the cooperative model enforces authentication and issue credentials using a single principal-name and password scheme as seen from the perspective of the $ACM application.

### Credential

Information containing the user's identity, privileges, and roles within a given security environment.

### Designated DOI agent

For targeted $ACM calls, the Designated DOI agent is the DOI agent specified in the $ACM call. For untargeted $ACM calls, the Designated DOI agent is generally the first DOI agent in the order of execution that locates the principal-name in its principal-name database. This is the only

DOI agent allowed to prompt for passwords. It always issues credentials if the authentication is successful and is responsible for the ultimate success or failure of the request unless the VMS agent cannot map the principal name.

### 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 principalnamespace, 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.

### Independent model

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.

### LOGINOUT

Two different LOGINOUT images are shipped with the OpenVMS Alpha operating system. To use the ACME subsystem, the ACME LOGINTOUT image must be installed. The other image, LOGIN82, is used for the traditional $LGI authentication. The procedures to install ACME LOGINOUT will be described in Appendix E.

### 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 may perform authentication and issue credentials.

## 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:

ACME$_PASSWORD_SYSTEM

ACME$_PRINCIPAL_NAME_IN

ACME$_PASSWORD_1

# Parallelism and Performance in the OpenVMS TCP/IP Kernel

Robert Rappaport, HP Software Engineer

Yanick Pouffary, HP Software Technical Director

Steve Lieman, HP Software Engineer

Mary J. Marotta, HP Information Developer

## Introduction

In October 2003, TCP/IP Services for OpenVMS introduced into OpenVMS production environments a radically modified and improved Scalable Kernel. The Scalable Kernel enables parallelism in TCP/IP by taking advantage of available CPU capacity in a multiCPU configuration. It allows network performance to scale almost linearly as CPUs are added to the configuration. The Scalable Kernel was designed to enhance network application performance without jeopardizing the integrity of the basic UNIX code.

## The SMP Challenge

The TCP/IP Kernel maintains a large in-memory database. Access to this database is synchronized by the use of several spin locks, all of which are associated with interrupt priority level (IPL) 8. On single CPU systems, only one active IPL 8 thread executes at a time. Therefore, there is no possibility for contention for the TCP/IP-specific spin locks on single CPU systems. On multiCPU systems, however, the potential for such contention increases as the number of CPUs in the configuration increases. The Scalable Kernel eliminates this contention.

When customers add CPUs to symmetric multiprocessing (SMP) systems, they expect the extra processing power to boost network performance, but the classic TCP/IP kernel does not take advantage of the extra processing power of the added CPUs. The number of users may actually increase, but almost all network I/O interactions are handled while holding the TCP/IP global spin lock (I/O lock 8). Contention for this lock makes it difficult to increase network throughput under these circumstances.

## The Architecture of the TCP/IP Kernel

The OpenVMS TCP/IP kernel, the heart of the OpenVMS TCP/IP architecture, was ported from BSD UNIX. It was intentionally designed to operate like the UNIX- TCP/IP kernel and to interoperate with the OpenVMS operating system with a minimum of programming changes.

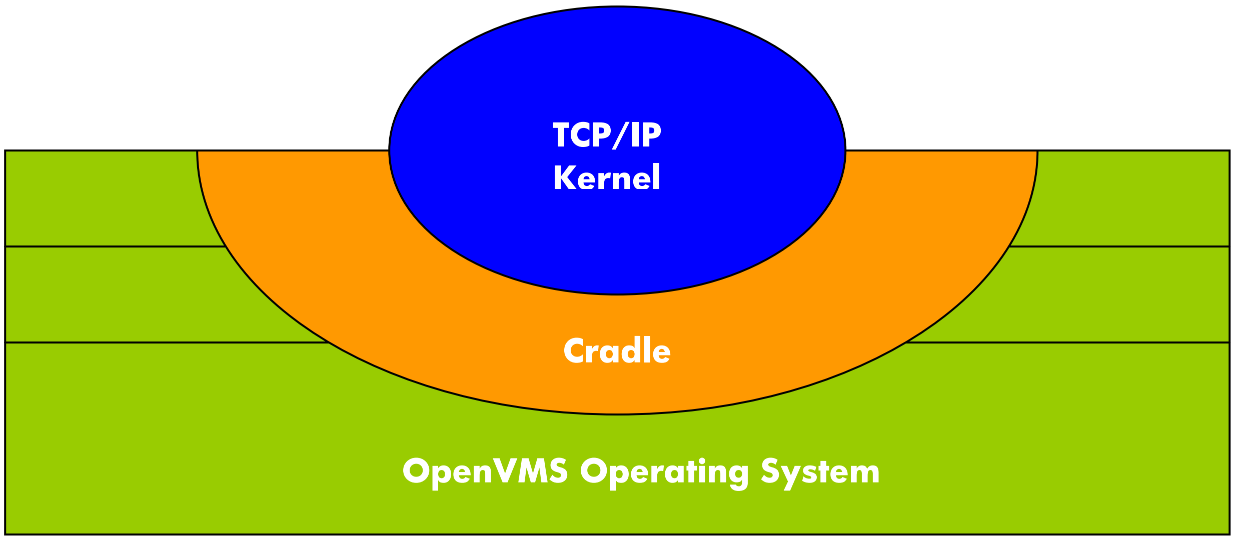


**Figure 1 The Architecture of the TCP/IP Kernel**

As illustrated in Figure 1, the OpenVMS TCP/IP kernel consists of two distinct parts:

- The **TCP/IP kernel** -- code that is ported from UNIX.
- The **cradle** -- OpenVMS code that supports and nurtures the UNIX code.

The cradle surrounds the UNIX code, creating an environment in which only a small percentage of the UNIX code has to be made aware that it is not operating in a UNIX system. The cradle provides transparent UNIX-like interfaces that serve the ported UNIX code in three general areas:

- User-level I/O requests are preprocessed in the cradle and fed into the UNIX code at the appropriate point.
- I/O terminations from the UNIX code are intercepted by the cradle transparently, as are all UNIX interactions with the LAN drivers.
- All interactions from the UNIX code with the OpenVMS operating system, such as the dynamic allocation and deallocation of memory, are handled transparently.

## TCP/IP Thread Contexts

Code executing in the TCP/IP kernel is either in **process context** or **kernel context** mode. A thread running in process context mode has access to the user address space (for example, the user's buffers). Threads running in process context are almost always executing code in the cradle. Threads running in kernel context run at IPL 8 holding the TCP/IP global spin lock.

In the classic TCP/IP kernel, when a thread changes mode to kernel context, it has to wait for the TCP/IP global spin lock. In the Scalable Kernel environment, kernel context threads are created as IPL fork threads, which then acquire the TCP/IP global spin lock. Kernel context threads are almost always executing in the UNIX-ported portion of the code.

Figure 2 illustrates how process context threads running in the classic Kernel environment contend for I/O lock 8 (the TCP/IP global spin lock of that environment) in order to change their mode to kernel. Once a thread acquires this spin lock it can then proceed to carry out its TCP/IP kernel work while all process context threads contending for this spin lock must wait, spinning and wasting valuable CPU cycles.

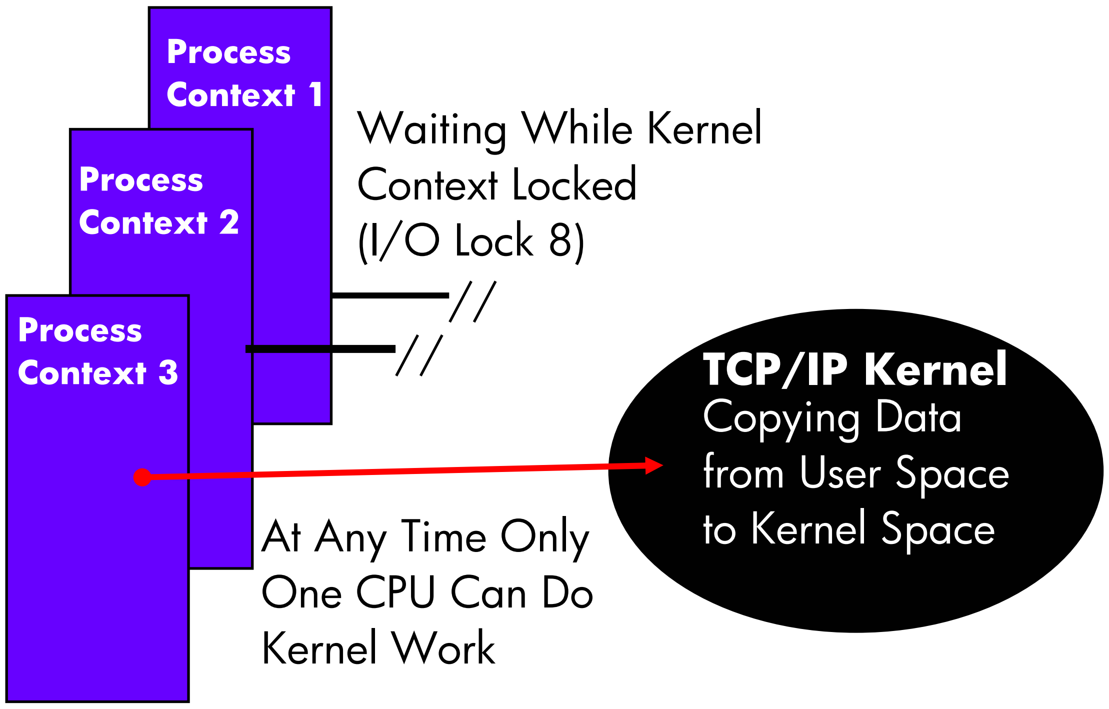

Process Context 1
Process Context 2
Process Context 3
Waiting While Kernel Context Locked (I/O Lock 8)
TCP/IP Kernel
Copying Data from User Space to Kernel Space
At Any Time Only One CPU Can Do Kernel Work

**Figure 2 Process Context in the Traditional Kernel**

As the number of network links per system gets larger and as the links get faster and faster, the potential number of network users requesting service can expand rapidly. As the demand increases, more and process context threads end up spinning in a loop, waiting for service while other threads are processed

## Introducing Parallelism into the TCP/IP Kernel

To address the problem of wasted CPU cycles spent spinning and to allow more work to get done on SMP systems, **parallelism** was introduced into the TCP/IP kernel code. Analysis of the classic kernel showed that only a small part of the processing of network operations had to be done while the TCP/IP internal database was locked. It was possible to change the order of the code flow in the two most frequently invoked network operations (read and write) so that:

- The kernel context portion of each read or write could run in an IPL 8 fork thread.
- The completion of read and write operations would not depend on these IPL 8 fork threads being completed.

In other words, read and write operations could be designed so that the process context portion of the work queues an IPL 8 fork thread to complete the kernel context portion of the work. Once this fork thread is queued, the user I/O request can then be completed. This is how the TCP/IP Scalable Kernel works.

In the Scalable Kernel, read and write operations are processed at IPL 2 in process context and queue IPL 8 fork threads to complete the kernel context work. Because each read or write operation does not have to wait until the fork thread has completed, the operation can be marked as completed (I/O Posted) immediately after queueing the fork thread.

The IPL 8 fork threads that are operating in kernel context need to acquire the TCP/IP global spin lock in order to access the in-memory database. Allowing these fork threads to run on any available CPU would lead to contention for the spin lock. Therefore, all of these TCP/IP kernel context fork threads are directed to a queue that is processed on one specific CPU in the configuration (the designated TCP/IP CPU) on a first-come, first-served order. Because all the threads in the system that need to acquire the TCP/IP global spin lock run on one single CPU, contention for this spin lock is eliminated. And because this spin lock is no longer I/O lock 8, no other OpenVMS code will attempt to use it.

The Scalable Kernel introduces a new mechanism for code to request the creation of a kernel context thread. The mechanism involves allocating a newly-defined data structure (the TCPIP_KRP), filling in the TCPIP_KRP, and then queuing this data structure to a global work queue. If the queue is empty at the time, an IPL 8 fork thread is created, which will run on the designated TCP/IP CPU and which will process every TCPIP_KRP in the queue.

## Tracking a Write Operation

The object of any TCP/IP write operation is to take data from a user buffer and place this data into a socket. The operation is performed in two distinct steps:

1. Copy the user data from the user buffer into a system buffer (MBUF) or a chain of system buffers (does not require holding the TCP/IP global spin lock)
2. Append this chain of system buffers into the socket (requires holding the TCP/IP global spin lock)

In the Scalable Kernel, the processing of a write operation is straight-forward. One or more MBUFs are allocated to accommodate the user data, and then the data is copied from user space into the new MBUF chain. A TCPIP_KRP is allocated and initialized so that it requests that this new MBUF chain be appended to the data in a particular socket. The initialization of the TCPIP_KRP includes passing the address of the MBUF chain, the address of the socket, and so forth. After the TCPIP_KRP is initialized, it is queued to the global work queue and the write request is completed.

At the same time that the write operation is being processed on one CPU, another write operation can be processed on another CPU in the system. Presumably, the other write operation is writing to a different socket. Because neither of these operations needs to acquire the global spin lock to complete, both operations run to completion without any interference. Similarly, they can run in parallel with ongoing read operations as well.

The power of the design of the Scalable Kernel becomes obvious. In a large multiCPU system, user programs running in parallel on the various CPUs of the system constantly call TCP/IP operations such as read and write. They run to completion, in parallel, without interfering with each other. Each of these requests leaves behind a TCPIP_KRP that is queued to be processed on the designated TCP/IP CPU; the processing of these TCPIP_KRP requests also runs in parallel with all the other operations.

Each process context operation leads to an associated kernel context operation. The amount of work entailed in each kernel context operation adds to the load of work on the designated TCP/IP CPU, but as long as this designated CPU is not completely saturated with work, the Scalable Kernel is able to scale close to linearly as more CPUs are added to the configuration.

The Scalable Kernel takes advantage of multiple CPUs by separating the user processes from the kernel process. Rather than blocking the CPU, it queues new user I/O requests. The flow of the send and receive logic in the cradle runs from start to finish without any interference from other TCP/IP threads. When they are successful, operations leave a pending asynchronous kernel-context thread to complete their requests. The user application does not have to wait for the kernel context thread to complete. When it queues the kernel context thread, the user request is completed. Network operations become more like transaction-oriented operations, where the parallel threads prepare transactions to be processed by the designated TCP/IP CPU.

As illustrated in Figure 3, applications no longer compete with one other to acquire locks in order to proceed.

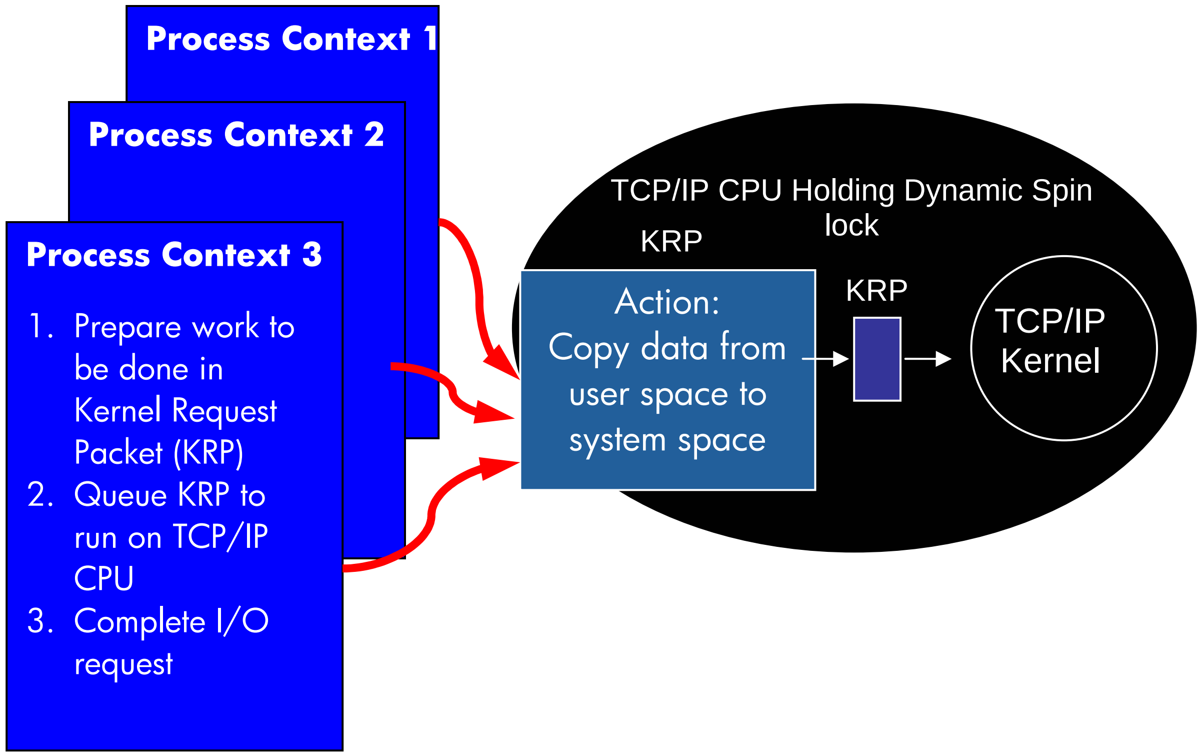

Process Context 1
Process Context 2
Process Context 3
1. Prepare work to be done in Kernel Request Packet (KRP)
2. Queue KRP to run on TCP/IP CPU
3. Complete I/O request
TCP/IP CPU Holding Dynamic Spin lock
KRP
Action: Copy data from user space to system space
KRP
TCP/IP Kernel

**Figure 3 Process Context Threads in the Scalable Kernel**

## Types of Kernel Request Packets (KRPs)

The TCPIP_KRP describes the request to perform an operation in kernel context, including a pointer to the action routine to be called, and a series of parameters that the routine will need to complete the request. There are many different types of requests for kernel context work in the Scalable Kernel.

In total, there are over 50 different types of KRPs in the Scalable Kernel. The type of KRP created depends on the work:

- A thread executing in process context that wishes to **write** data to a socket packages up all the data to be written to the socket inside a KRP and then creates a kernel context thread to process the KRP. The processing of this KRP includes extracting the information from the KRP and calling the UNIX routines that insert new data into the transmit channel of a socket.
- A thread receiving a call from the OpenVMS LAN driver must pass **received** data from the network. This thread packages the received network data in a KRP and then creates a kernel context thread to process this KRP. To process this KRP, the kernel has to parse the received network data (IP header, TCP or UDP header, and so forth), place the parsed

data into the receive channel of a socket, and possibly wake up a user thread waiting for data to arrive on this socket.

- When a thread running a **TCP/IP timer** goes off, the information about the timer is packaged in a KRP and a kernel context thread is created to process it, executing the appropriate code to deal with the specific timer that expired.

### Kernel Context Threads

The processing of kernel context threads is invisible to the TCP/IP application program. All the kernel threads access the same shared in-memory database, which holds information that cannot be accessed concurrently by more than one thread at a time. Processing in kernel context is ensured by the fact that the threads that execute in kernel context are all directed to a single, designated CPU in the configuration, where they execute one by one, at high priority and at high speed with no interference from other threads.

Instead of I/O lock 8, the Scalable Kernel uses several new dynamic spin locks, like the TCP/IP global Spin lock, which is held for relatively long periods of time, and several mini-spin locks, which are never held for very long. Each TCPIP_KRP is processed in an IPL 8 fork thread on the designated TCP/IP CPU, while holding the TCP/IP global Spin lock. Since all of the threads that need the TCP/IP global spin lock run on the TCP/IP CPU, there is never any contention for the spin lock.

Executing all the kernel threads on the same CPU also optimizes CPU cache utilization because the same objects in the shared database are usually referenced from the same CPU.

## The Scalable Kernel

TCP/IP Services Version 5.4 introduces the Scalable Kernel as an optional new feature for the specific purpose of validating and quantifying the performance gains for those systems with the heaviest TCP/IP loads on SMP systems. The Scalable Kernel significantly improves the potential for performance gains depending on the applications and configuration.

The Scalable Kernel will be the default TCP/IP kernel in the next major release of TCP/IP Services for OpenVMS beyond V5.4. Tests have shown equivalent or better operational performance on single-CPU systems, and indisputable benefits for multiCPU systems under the heaviest TCP/IP loads. To obtain the benefits of the Scalable Kernel, you must upgrade your system to TCP/IP Services Version 5.4 or higher.

## Measuring Throughput

The maximum gain to expect in system throughput by using the Scalable Kernel is a direct function of the amount of MPSYNCH that is attributable to TCP/IP, based on measurements using the classic TCP/IP kernel. System throughput gain is always highly application-dependent.

In a given configuration running the Scalable Kernel, the amount of remaining capacity (headroom) can be estimated by measuring the amount of time that the TCP/IP global spin lock is held by the designated TCP/IP CPU under heavy TCP/IP load. For example, in a multiCPU configuration, if the TCP/IP global spin lock is held for 40% of the time, the number of CPUs in the configuration can be doubled before causing TCP/IP bottlenecks.

## Scalable Kernel Performance Tests

The following graphs show real-life data confirming the success of parallelism in the field. Note that although performance tests may show higher I/O operations per second, better utilization of system resources, and so forth, the customer is only interested in getting more of his work done in a specific unit of time.

## Performance Summary

On a16-CPU GS160 Wildfire system running the classic TCP/IP kernel, the testbed application resulted in an MPSYNCH backup averaging four or more CPUs. That is, at any given time, four or more CPUs were spinning and doing nothing constructive. (This represents 25% to 35% of the potential productivity of a 16-CPU machine!)

The Scalable Kernel restores that 25% to 35% gain in throughput, virtually eliminating the waste of the previously spinning CPUs. In other words, the Scalable Kernel allows greatly expanded parallelism to make use of previously lost CPU cycles.

## Comparing the Traditional Kernel to the Scalable Kernel

These test results show the overall pattern of improvement when the scalable kernel is running (in green), and when it is not running (in red).

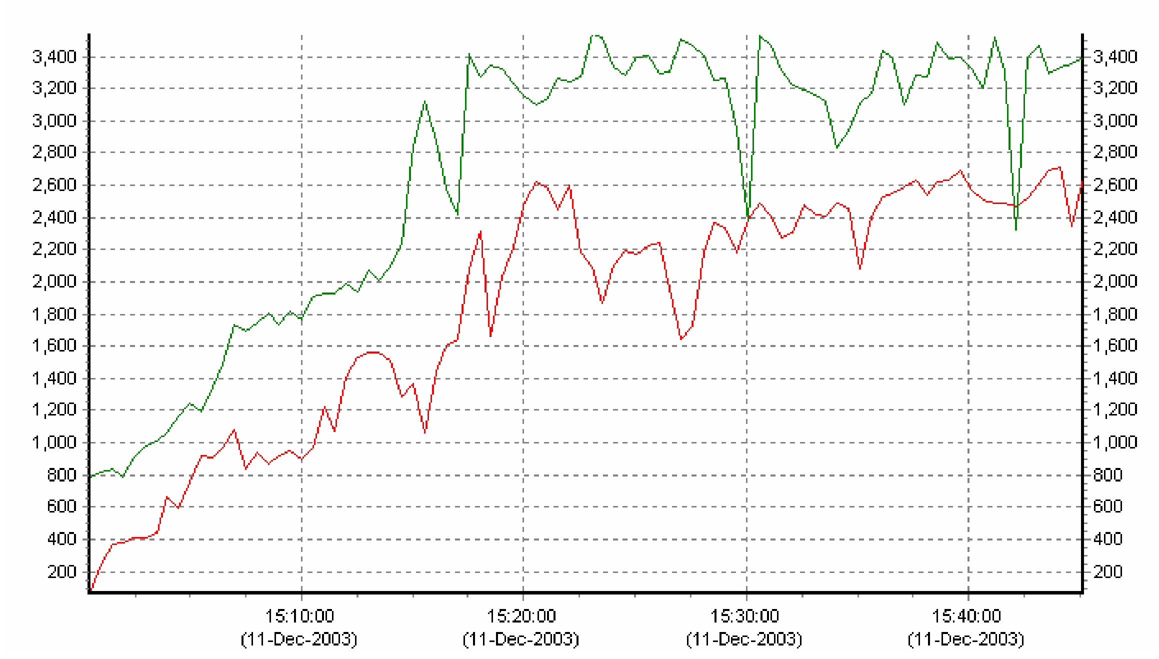


**Figure 4 - TCP/IP Transmit Packets per Second**

The purpose of the scalable kernel is to increase the amount of network I/O that an OpenVMS system can process. As Figure 4 shows, the rate of TCP/IP traffic can potentially increase by 30% or more when the Scalable Kernel enabled.

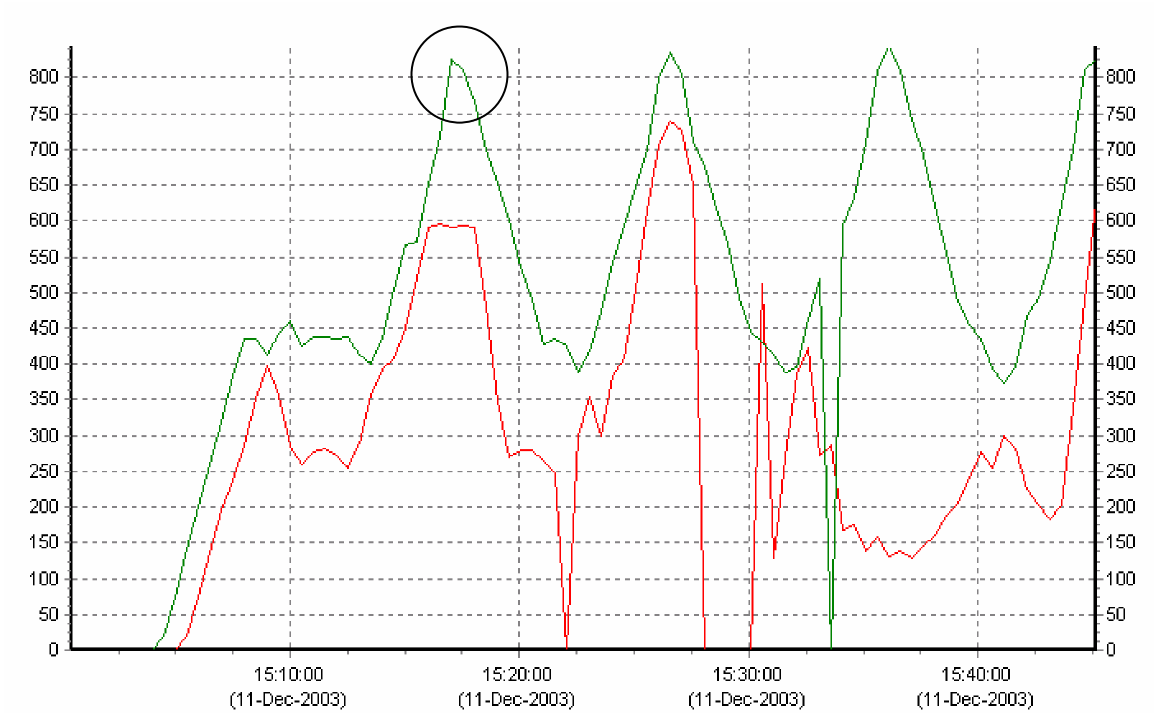


**Figure 5 – Customer Orders per Minute**

As shown in Figure 5, the Scalable Kernel allows the system to format more customer orders in a given time than the classic kernel. When the load gets heavy, the Scalable Kernel is able to respond and complete more real work per minute than previously possible.

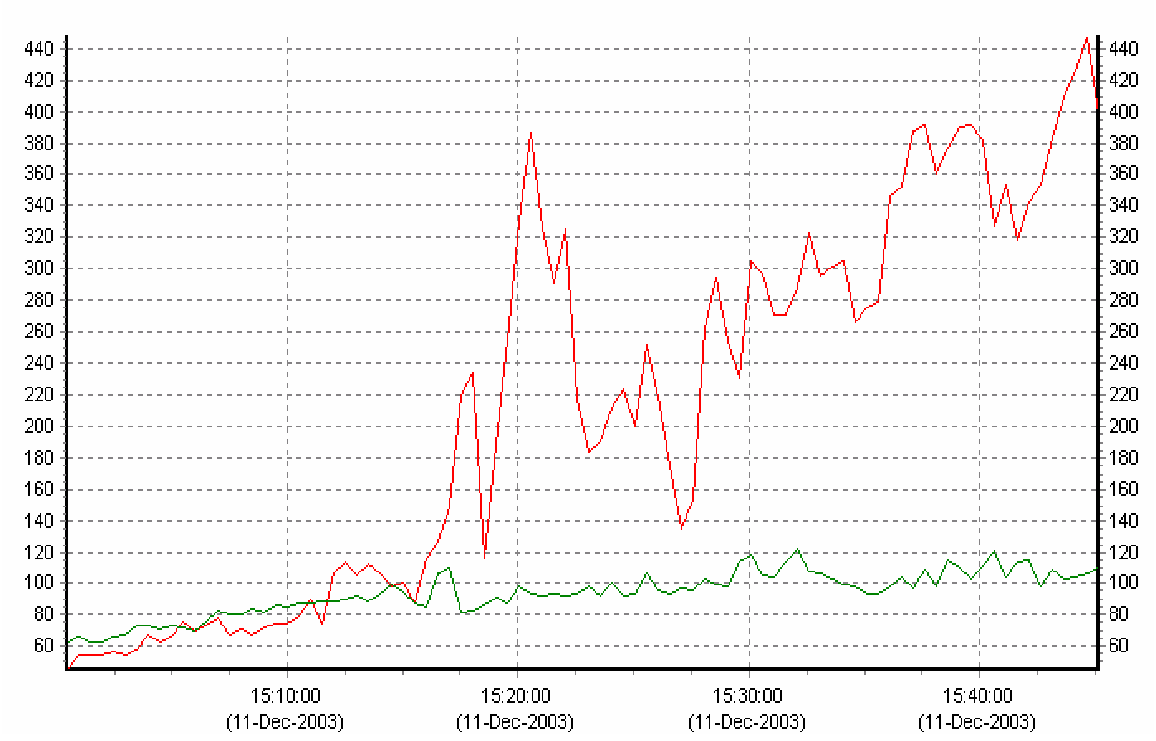


**Figure 6 – MPSYNCH Percentage**

When the Scalable Kernel is running, multiprocessor synchronization contention (MPSYNCH) is dramatically reduced, as shown in Figure 6.

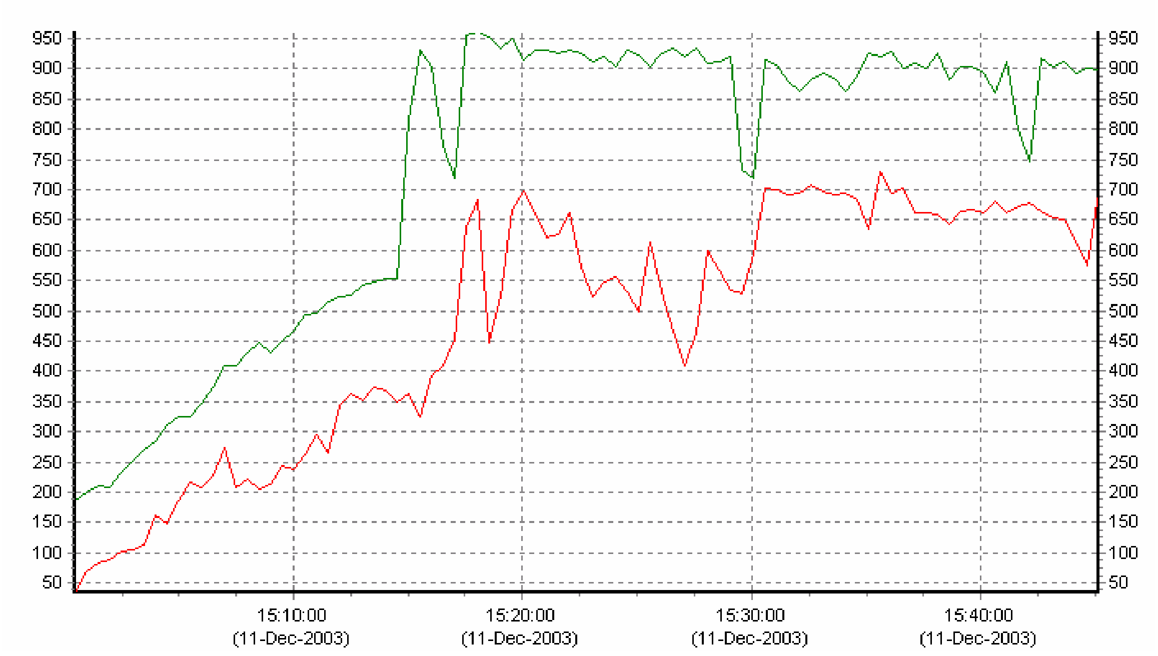


**Figure 7 – Percentage of CPU Busy**

As Figure 7 shows, a greater percentage of CPU time is spent in user-mode, which means that more application work is getting done when the Scalable Kernel is running.

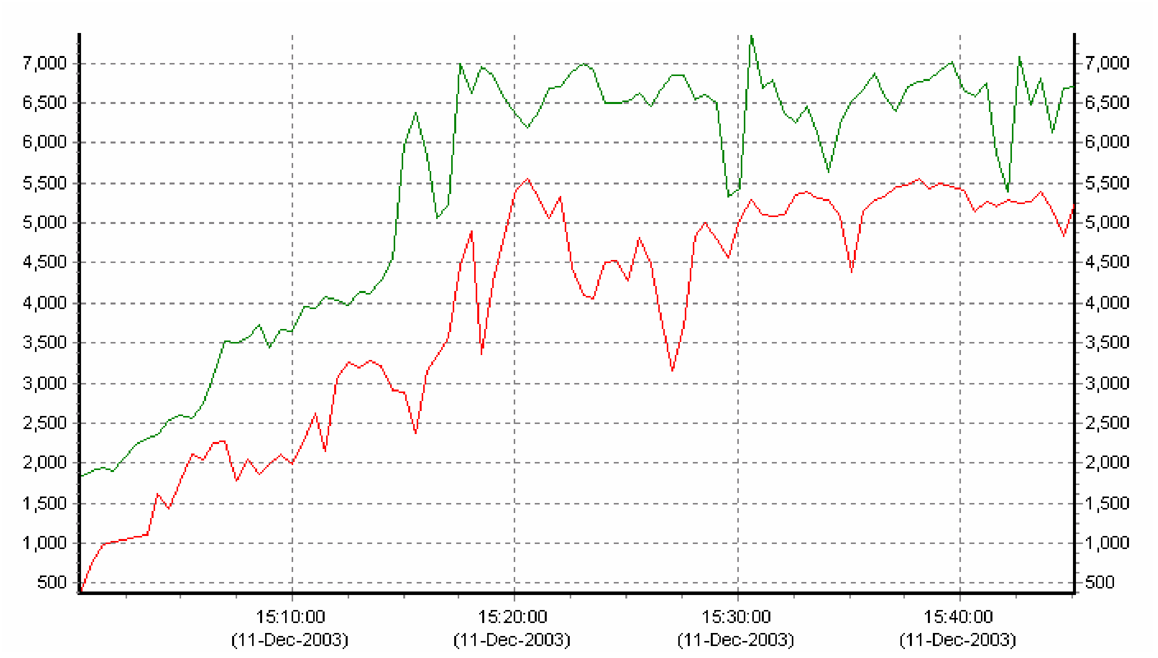


**Figure 8 - Buffered I/O Rate**

On OpenVMS, TCP/IP I/O activity is expressed as buffered I/O. Figure 8 shows how the rate of buffered I/O increases when the Scalable Kernel is running.

## Comparing the Traditional Kernel to the Scalable Kernel

| | **Traditional Kernel** | **Scalable Kernel** |
|---|---|---|
| **Hold % All Locks** | 54.8% | 35.2% (while completing more work) |
| **Spin % All Locks** | 24.1% | 4.4% (very good) |
| **Locks Per Second** | 204,153 | 162,033 (doing more work with far fewer locks per unit of work) |
| **I/O Lock 8 Hold Time** | 31.4% | <5% (very good. Now much more of I/O lock 8 is available for handling heavy disk I/O.) |

## The Importance of Maintainability

Over half the OpenVMS TCP/IP kernel code is ported from UNIX and the TCP/IP code base is under constant development. In order for the OpenVMS TCP/IP kernel to remain up-to-date with leading-edge functionality, frequent infusions of new UNIX code are required. The OpenVMS TCP/IP engineering team must repeat the port of the UNIX code periodically. The amount of OpenVMS modifications introduced into the ported UNIX code must be restricted, so that re-porting operations remain a manageable task that can be accomplished in a time period of weeks, not years.

Limits on the amount of changes that could be introduced into the UNIX code dictated the approach to the challenge of achieving parallelism in the TCP/IP Services for OpenVMS product. Complicated locking schemes that would require that lock domains span both the ported UNIX code and the OpenVMS cradle would have greatly increased the complexity of the solution, introducing issues of quality as well as increased maintenance.

The Scalable Kernel is the ideal solution because it is customized to the OpenVMS SMP environment, it operates just as well in a single-CPU configuration as an SMP system, and it imposes the least amount of overhead for future maintenance.

## Future Kernel Enhancements

In the future, the Scalable Kernel may be enhanced to handle even larger CPU configurations. To accomplish this, the current single, shared in-memory database could be divided into two or more databases, each of which would be serviced by its own designated kernel context CPU. This would ensure that the designated TCP/IP CPU does not become a limit to system throughput.

Additional performance gains can be realized by optimizing the processing of the transactions, so that the designated TCP/IP CPU takes less time and effort to process each individual transaction, thereby supporting a greater number of parallel threads without becoming overloaded.

## For More Information about TCP/IP Services Performance

The Scalable Kernel allows greater parallelism in the processing of TCP/IP requests and is not a generalized performance panacea that solves everything. Processing tens of thousands of TCP/IP packets and distributing them over thousands of sockets requires CPU cycles, which will inevitably take its toll. The Scalable Kernel allows you to deal efficiently with this necessary use of CPU resources by adding CPUs to the configuration and then allowing these additional CPUs to be effectively used by the system instead of merely spinning, doing nothing and getting in the way.

There are other steps, independent of the Scalable Kernel, which you can take to improve the performance of individual TCP/IP operations, including TCP/IP tuning, adjusting window sizes for sockets, and so forth. For more information about these performance enhancement techniques, consult the *TCP/IP Services for OpenVMS Tuning and Troubleshooting* guide.

# OpenVMS Technical Journal

## Examining Web Services:

**Protecting Your OpenVMS Investment**

David J. Sullivan, Expert Member Technical Staff

### What are web services?

Web services are a set of technologies designed to aid in the development of heterogeneous and platform-neutral solutions. Since their introduction, web services have received unprecedented acceptance among enterprise software vendors. They provide businesses with a strategic advantage because of their relative simplicity, lower cost, and wide cross-platform availability.

This article examines web services from both a business perspective and a technical perspective. It explains the business needs driving web services as well as the core web services technologies. It also explains the OpenVMS strategy for supporting web services and provides example applications that illustrate the integration of an OpenVMS application with other platforms, such as Microsoft® .NET and Linux®. These topics may be read independently. Use the links below to go to the sections of most interest to you:

### Why should you care about web services?

Traditional integration technologies have many flaws. The traditional approach to integration relies on the use of middleware products. These products are often costly, proprietary, and difficult to use. Middleware is rarely available across all platforms and the costs associated with the use of middleware are significant. Perhaps of most concern is that a user is forced to bet their business on the success of a particular vendor's middleware product.

Web services provide a technology that is capable of tying together any applications written in any language on any operating system. As compared with traditional middleware, web services are less expensive, open, and simpler to use. Your business is safe because implementations of the web service standards are widely available from many different sources.

A common and costly business problem occurs when technology limits the availability of information that flows throughout the corporation. Often, applications located in different divisions of a corporation cannot share data because they use different operating systems, middleware, and programming languages. These islands of technology are often isolated from each other or use gateways that attempt to patch together the various technologies. The potential combinations of integration points are significant.

For example, in Figure 1, the lettered boxes represent separate applications, and each arrow represents a technology used to integrate them. The boxes represent the following types of applications:

**A:** OpenVMS applications

**B:** Microsoft C++ application using DCOM

**C:** IBM C application using MQSeries

**D:** HP-UX COBOL application using CORBA

**E:** This system does not access an OpenVMS system

**F:** Microsoft Windows® Java™ application using JMS

**G:** Sun Java application using J2EE

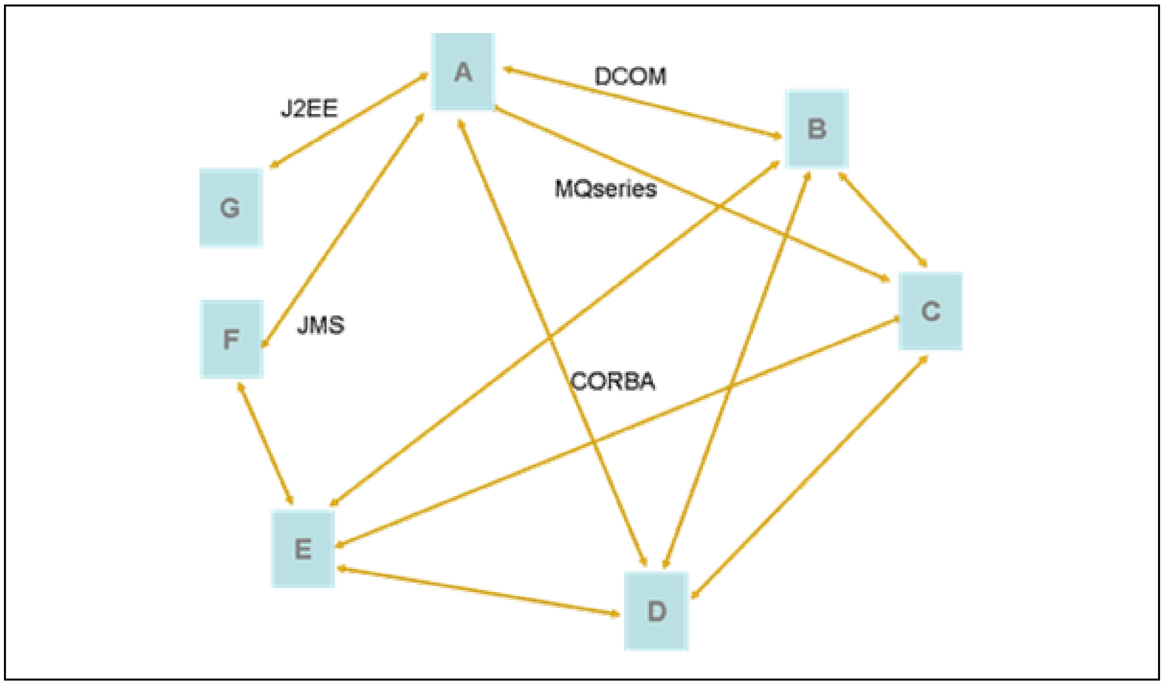


**Figure 1: High-cost, complex IT environment**

It is easy to see the complexity and overhead that is required to develop and maintain this infrastructure. Each arrow represents a large investment of resources in hardware, software, management and training. Each hard-coded connection has unique security concerns that increase complexity. Also, this outdated technology does not work well across the Internet, thereby eliminating the ability to perform effective business-to-business interactions. Integrating applications becomes very difficult because each pair of applications must be addressed separately.

With web services, solutions architects have a common integration technology on all platforms. By exposing an OpenVMS application as a web service, it can be called by any web service client, regardless of the platform or programming language used by the client (Figure 2). Likewise, the web service client never knows the platform or programming language of the web service being called.

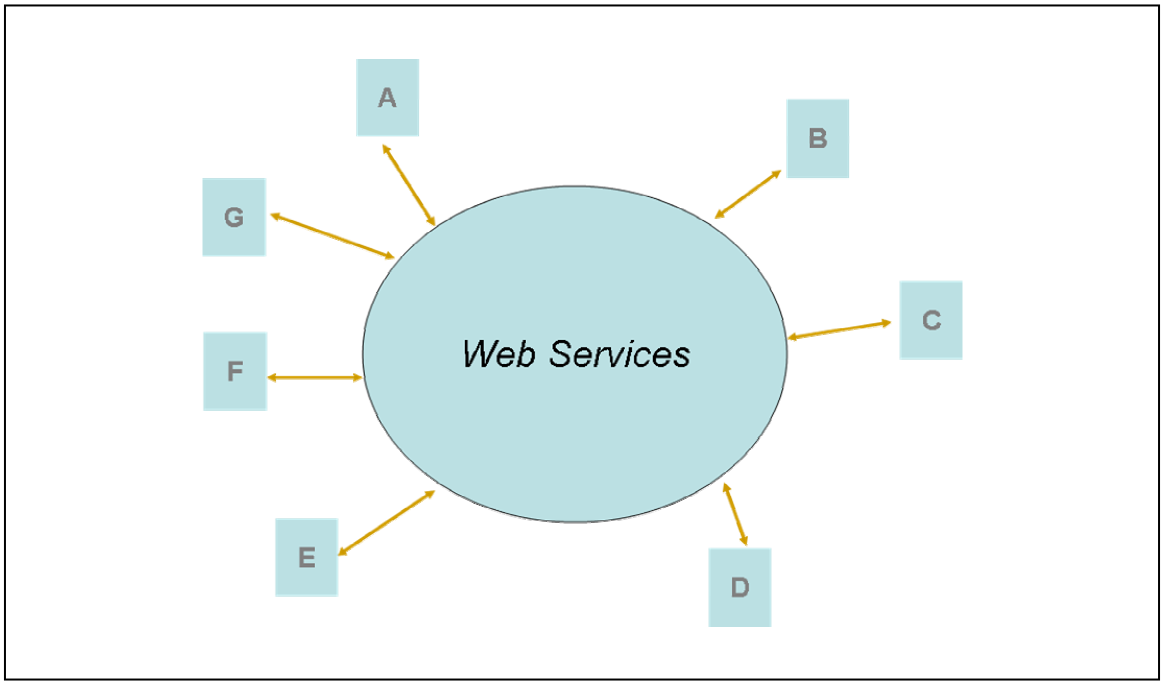


**Figure 2: Simpler, less expensive IT environment**

Integration issues with OpenVMS are now much easier and less costly with web services. Architects can make use of the unique strengths of OpenVMS systems without having to make the undesirable tradeoffs listed earlier. Legacy applications can be integrated with other solutions that are based on web services. The original investment in design, development, testing, and maintenance of OpenVMS applications continues to pay dividends to the business while freeing money and resources for investments in other areas.

Table 1 lists the advantages of using web services over traditional middleware integration products.

**Table 1: Advantages of web services over traditional middleware**

| **Traditional middleware** | **Web services** |
|---|---|
| Communication styles are dictated by middleware. Usually RPC or MOM. | Supports many styles of communication RPC, Messaging, 1-way, 2-way, conversational. |
| Lack of cross-platform availability creates islands of separate technology. | Allows single integration backbone across all platforms. |
| Lacks standards:<br>• Many ways to do common jobs<br>• Proprietary solutions from individual vendors | Standards based:<br>• One way to do common jobs<br>• Open solutions from many vendors |
| Hard to use (design, program, manage). | Simpler to use. |
| Expensive. | Less expensive. |
| Does not encourage reuse of code. | Encourages reuse of code. |
| Customized for certain operating systems and programming languages. | Operating system and programming language neutral. |
| Does not scale across devices. | Can be used from the smallest handheld device up to a large server. |

| | |
|---|---|
| Does not work over Internet. | Designed to operate over Internet. |
| Brittle and rigid. | Flexible and adaptable. |
| Tightly coupled. | Loosely coupled. |
| Proprietary data formats. | Leverages XML. |
| Undocumented binary protocols. | Uses open and standard text protocols. |
| Invasive. | Less invasive. |

## How do OpenVMS customers benefit?

There are many ways in which OpenVMS customers can benefit from using web services. The advantages over traditional middleware are significant.

### Investment protection

An existing application can be exposed as one or more web services to extend its use and prolong its life.

### Utilization of OpenVMS strengths

The strengths of OpenVMS can be used throughout a corporation to host critical, bet-your-business applications. These OpenVMS applications can be exposed as web services and can be used by non-OpenVMS platforms. The client code never knows that the service being used is deployed on OpenVMS.

### Availability of more applications

OpenVMS applications can make use of applications that are not natively available on OpenVMS. In these cases, the OpenVMS system acts as a consumer of web services that are deployed on other platforms, such as Microsoft Windows .NET or HP-UX.

### Phasing out expensive middleware

Customers have the option of phasing out costly, unnecessary middleware over time.

### New opportunities for legacy applications

Web services provide a new opportunity for applications that previously could not be integrated with traditional middleware. Web services support more design models than traditional middleware and are neutral as to the programming model used by the application.

### Cross-language programming

Web services can be used on OpenVMS systems as a simpler and more extensible mechanism for communication between different programming languages.

### Availability over the Internet

OpenVMS applications can be programmatically accessed from the Internet. Web services are designed to operate over the Internet without any extra design or coding.

### Cost savings

Standardizing on web services reduces the cost of development, testing, maintenance, and management. Web services platforms are widely available from many sources including free, high-quality, open source offerings.

**Compatibility with newer technologies**

Because of the wide acceptance and availability of web services, new technologies such as the Adaptive Enterprise, Virtualization, and the GRID are all built on a web services IT backbone.

**Use of XML**

XML is the preferred data storage and description mechanism. It is highly likely that a business already uses XML documents to represent valuable data. Web services are based on XML and naturally support the exchange of XML documents.

**Common business languages**

Industries are using XML to define industry-standard definitions of common business entities.

## Success Stories

Many interesting case studies show how web services have had a significant and positive impact for Fortune 500 businesses. *Web Services Case Studies*

## Technology

The technologies behind web services are designed by standards organizations. There are a staggering number of drafts, recommendations, and standards dealing with web services. Fortunately, most developers do not need to master, or even understand, the majority of these specifications.

Web services development platforms hide many of the details of these technologies from the developer. In fact, some development platforms allow engineers to use wizards to generate all the source code for a web service. However, the more developers understand the concepts behind the core technologies, the more effectively they can design and develop robust solutions based on web services.

Each web service platform presents these same concepts in different ways and with different levels of exposure. After reading this section, you should have a much better understanding of how the unique development tools are able to generate highly interoperable applications.

### WS-I

The Web Services Interoperability (WS-I) organization is responsible for delivering clear and consistent recommendations for ensuring interoperability between web services. This is perhaps the most important standards organization in the web services world. In order for web services to continue to spread, there must be a single definition of web services compliance.

In August 2003, WS-I announced the approval of the Basic Profile 1.0 (BP 1.0). BP 1.0 consists of implementation guidelines for how a set of core web services specifications should be used together to develop interoperable web services.

This article describes only those features that are included in the Basic Profile.

### XML

XML stands for Extensible Markup Language. It was adopted by the W3C in 1998 and has since become the preferred way to store business data.

XML was designed to describe and store data. It is similar to a markup language (such as HTML) in that it has begin and end tags. Unlike HTML, it has no predefined tags. Rather, with XML the author first defines the tags and then uses them to describe the data. When XML tags are defined, they often have a hierarchy to convey a relationship, such as parent-child.

XML does not determine how the data is displayed. Separating the data from how it is displayed allows XML to remain simple and flexible. XML gets its power and popularity from its simplicity. An XML document is:

- A plain text file
- Human readable
- Understood by all platforms and programming languages

Because all operating systems understand text files, XML documents can be created and modified using a simple text editor. It is this common understanding of text files that allows XML to be used as a common data format across all platforms and languages.

As an example, the XML file (called employee.xml) in Figure 3 should be easy to understand even if you have never before seen XML.

```
<?xml version="1.0"  encoding=”UTF-8”?>
<employee>
      <name>
        <first>David</first>
        <last>Sullivan</last>
      </name>
      <email>davidjsullivan@hp.com</email>
</employee>
```

**Figure 3: employee.xml**

The employee.xml file in Figure 3 contains five tags: `<employee>, <name>, <first>, <last>,` and `<email>.` Three pieces of data are stored in the file: `<David>, <Sullivan>,` and `<davidjsullivan@hp.com>`. This is a complete, well-formed XML file. It has a hierarchy of tags describing that an employee has a name and an email address. Also, the <name> tag contains a first and last name. It's that simple!

It would be simple for an application to parse this XML file and do something interesting with the data. For instance, a browser might display the data as part of an employee list, or an application might parse the file to send out automated email notifications.

## XML schema

The XML document in Figure 3 is well formed. A well-formed document is syntactically correct. For example, all begin and end tags are properly nested. This XML document is useful in its current state. However, the document is not considered valid because we have no way of knowing whether the tags and associated data are semantically correct. For instance, if we remove the line `<first>David</first>`, is the <name> tag still useful (valid)? The answer is maybe. It depends on the intent of the person who defined the tags. This is where an XML schema comes in.

An XML schema is a language used to describe XML tags. When an XML document has a schema, it becomes much easier for an application to understand the layout of an XML document. Also, a schema allows an application to determine whether the contents of the XML file are semantically valid before processing the document.

Each tag in an XML document is defined with an element declaration. When writing a schema file, the author has the option of specifying different properties for the elements, including:

- An associated data type (for instance, string, number, or user-defined type).
- Constrain the valid values for a data type (for instance, allow only integers that are even).
- Require or prohibit child elements and attributes (for instance, the element name could require a last name but make the first name optional).

There are many more options in XML schemas. Refer to a good book on XML for details.

Figure 4 shows the schema file (called employee.xsd) for the XML document in the Figure 3. At first sight, the schema language appears complex (and it is), but after a day or two, writing schema files becomes second nature.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="x-schema/employeeData"
            xmlns="x-schema/employeeData"
            elementFormDefault="qualified">

<xsd:element name="employee">
    <xsd:complexType>
        <xsd:sequence>

            <xsd:element name="name">
                <xsd:complexType>
                   <xsd:sequence>
                        <xsd:element name="first"/>
                        </xsd:sequence>
                        <xsd:element name="last">
                        </xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>

            <xsd:element name="email">
            </xsd:element>

        </xsd:sequence>
   </xsd:complexType>
</xsd:element>

</xsd:schema>
```

**Figure 4: employee.xsd**

In the employee.xsd schema file, we see the definitions of the elements *employee*, *name*, *first*, *last*, and *email*. Notice the nesting of the elements. Any XML file using this schema must include all of the elements defined in the schema, because the author didn't mark any of them as optional. For this reason, the answer to the question in the previous section is no, the <name> tag is only valid if it includes both a first and last name.

Modern development environments, such as the OpenVMS IDE NetBeans, provide rich tools to create and modify XML and XML schemas.

*Note: A DTD is an alternative to a schema. DTDs are not addressed here because they are out of date and not used in web services.*

## Introducing SOAP, WSDL, and UDDI

There are three standards-based technologies that, along with XML, comprise the core of web services.

- SOAP (Simple Object Access Protocol)

  The function of SOAP is to transmit XML messages between two applications. SOAP is itself an XML language and is defined using an XML schema.

- WSDL (Web Services Description Language)

  WSDL is a language based on XML. It is used to precisely define the details of a web service. Web service consumers use this information to build SOAP messages for the service the WSDL describes.

- UDDI (Universal Description, Discover, and Integration)

  UDDI is a specification that defines a directory for web services. Often called the "Yellow Pages for Web Services," UDDI directories are used to store services that are available to consumers.

Figure 5 provides a high-level overview of how the SOAP, WSDL, and UDDI technologies interact. These three technologies were defined specifically for use with web services. In Figure 5, a web services client performs the following steps to obtain a stock quote from an OpenVMS application that has been exposed as a web service.

**Step 1:** The client looks in a UDDI directory to find a web service that can supply stock quotes. It discovers the stockQuote service can supply the desired features. The UDDI registry provides an address, in the form of a URL, to the client. (You can skip this step if the client already knows the address of the web service.)

**Step 2:** The client sends a lookup request to the address obtained in step1, asking for a description of the service. This description is returned to the client in an XML schema language called WSDL.

**Step 3:** The consumer uses the WSDL description obtained in step 2 to identify the details of the service's functions and associated arguments. The client makes a call to the stockQuote service by sending a SOAP protocol request.

**Step 4:** The web services platform receives the SOAP protocol request and returns the quote to the client via a SOAP protocol response.

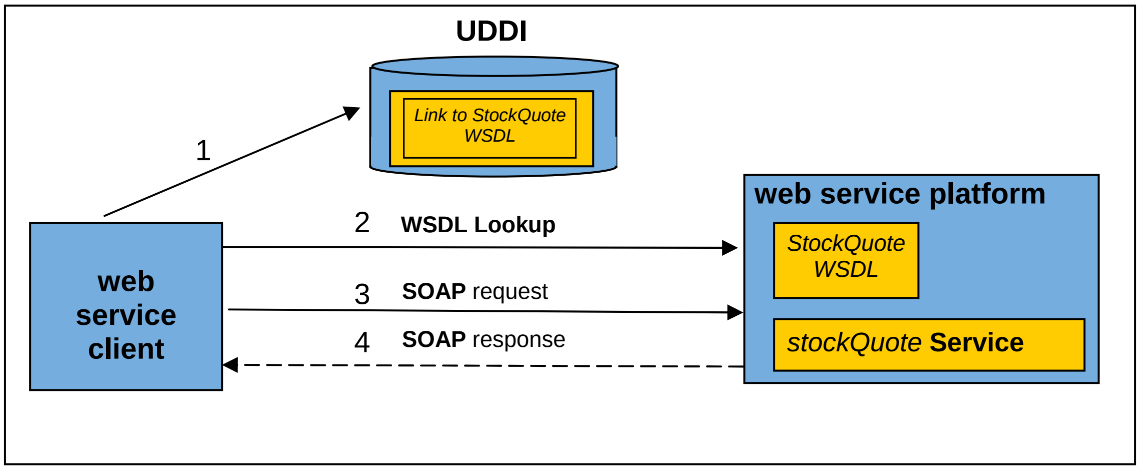


**Figure 5: Overview of WSDL, SOAP, and UDDI**

## SOAP

SOAP stands for Simple Object Access Protocol. The function of SOAP is to transmit XML messages between two applications. SOAP is itself an XML language and is defined using an XML schema. The schema defines a SOAP root element called Envelope. You can look at the SOAP V1.1 XML schema definition at the following URL:

http://schemas.xmlsoap.org/soap/envelope/

Before SOAP can transmit XML to a destination, it must first wrap the XML in a SOAP Envelope. An Envelope defines SOAP-specific elements and has a specific structure. The envelope contains two sections: a Header (optional) and a Body (required). Figure 6 shows the structure of the SOAP Envelope.

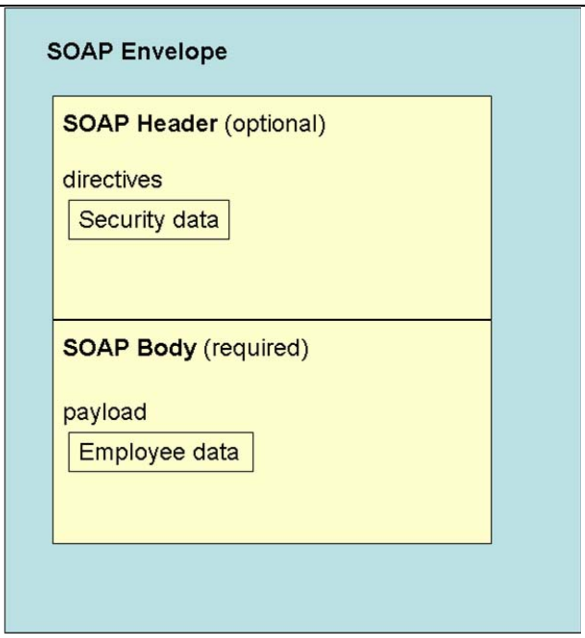


**Figure 6: Structure of the SOAP Envelope**

Web services platforms use SOAP as the protocol for transmission of XML messages. In Figure 6, the XML employee data is sent within the SOAP Body.

### *SOAP Messaging Styles*

The data of the SOAP Body, also called the payload, is defined by the application. The SOAP schema defines two styles for structuring the payload: RPC and document.

With the RPC messaging style, the structure of the payload is defined by SOAP. SOAP defines its own representation of an RPC call. This structure specifies the method name and arguments for a call to the service.

With the document messaging style, the structure of the payload is defined by the application's XML. SOAP does not impose any structure on the contents of the SOAP Body.

Figure 7 illustrates the two SOAP messaging styles.

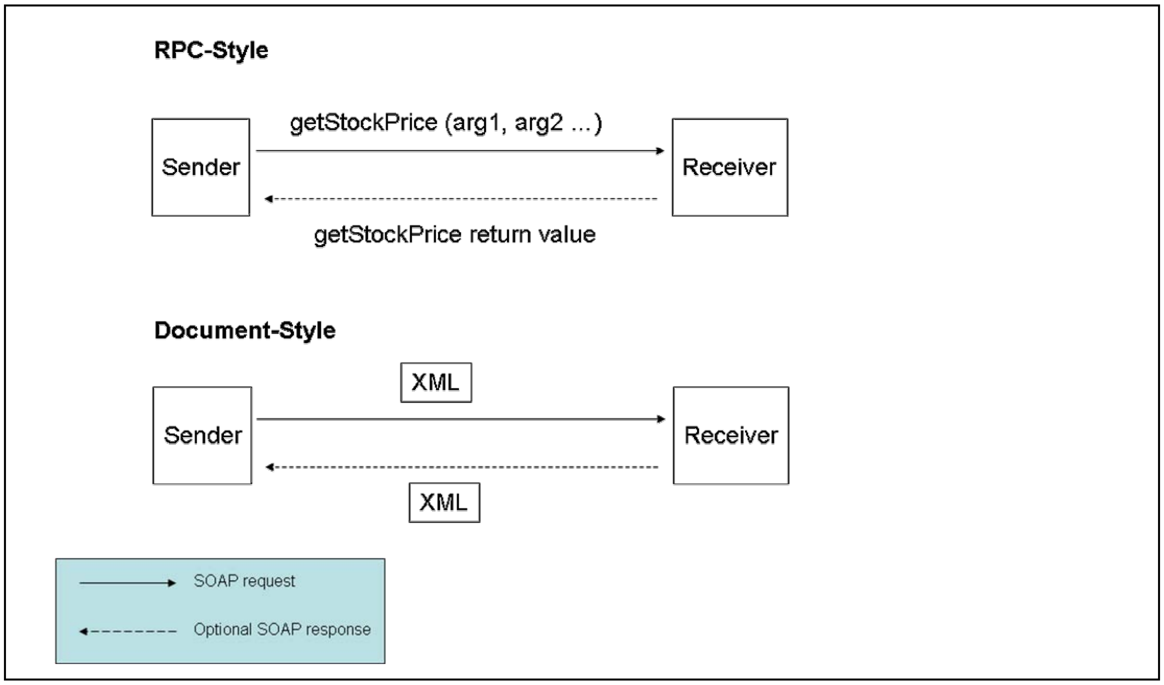


**Figure 7: SOAP messaging styles**

The following example shows a SOAP request using the RPC messaging style. In this example, the method getStockPrice is being called and requires one argument, which is a ticker symbol. The web services client application does not specify the element names or hierarchy. SOAP automatically uses this structure within the SOAP Body for RPC messaging.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
xmlns:abc="http://abc.com/stock">
<soap:Body>
   <abc:getStockPrice>
      <symbol>HPQ</symbol>
   </abc:getStockPrice>
</soap:Body>
</soap:Envelope>
```

The following example shows a SOAP request using the document messaging style. In this example, the contents of the SOAP Body are defined by the application. The employee XML is sent to the web service without any intervention from SOAP.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
xmlns:abc="http://abc.com/employee">
<soap:Body>
<employee>
      <name>
        <first>David</first>
        <last>Sullivan</last>
      </name>
      <email>davidjsullivan@hp.com</email>
</employee>
</soap:Body>
</soap:Envelope>
```

### *SOAP Transport*

Although SOAP is responsible for transmitting the XML message, it is not designed to use its own network transport to send messages. Instead, SOAP uses the HTTP protocol for sending messages. By using HTTP as the transport vehicle, web services are available across the Internet.

*Note: SOAP can use other transports, but the WS-I Basic Profile supports only HTTP.*

### *SOAP extensibility*

SOAP is extensible. The header section of the SOAP Envelope is used to add more features. For instance, security, reliability, and transactions are important features for enterprise-level corporations. These features can be added by specifying XML headers in the SOAP Envelope request. The headers can be either standard or propriety extensions to the SOAP protocol.

When a SOAP message is sent to a receiver, the message can be transmitted through one or more intermediaries. An intermediary can examine, modify, and remove headers, depending on its role in the transmission of the message. Figure 8 shows the interaction of SOAP intermediaries.

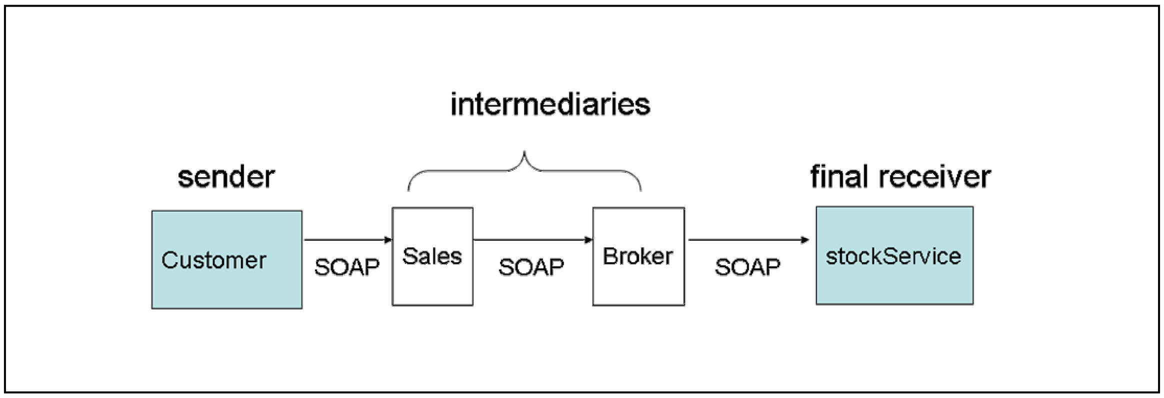


**Figure 8: SOAP intermediaries**

## WSDL

WSDL stands for Web Services Description Language. WSDL is an XML language that is defined using a schema and is used to define precisely the details of a web service. Developers typically do not need to write in WSDL, but they need to understand the organization and purpose of WSDL documents.

### WSDL elements

The WSDL schema language provides a number of elements to describe a web service. These elements fall into three groups.

**Group 1:** These elements describe the interface exposed for use by clients. The interface is defined in abstract terms.

**Group 2:** These elements "bind" the abstract interface to a transport chosen by the service, such as SOAP. This binding maps the abstract interface descriptions to concrete mechanisms of the chosen transport.

**Group 3:** These elements name the web service and assign an address that the client uses to identify the location of the service.

#### *Group 1: Describing the interface*

The group 1 WSDL elements describe the web service in abstract terms. Table 2 describes these elements.

**Table 2: WSDL elements that describe the web service interface (group 1)**

| Element name | Description |
|---|---|
| *portType* | Provides a name for an interface. |
| *operation* | Provides a name for a method in a *portType*. |
| *input, output* | Describe the *operation* element and are used to specify one of the following interaction models:<br>• 1-way interaction: The client calls the service and the service does not send a response back to the client. In this case. only the *input* element appears in the operation.<br>• Request/reply interaction: The client calls the service and the service sends a response back to the client. In this case, only the *input* and *output* elements appear in the operation. |
| *part* | Specifies a piece of data that must appear in the message element. There may be zero or more part elements in a *message* element. There are two attributes for describing the data: **type** and **element**.<br>With the **type** attribute, the *part* element is given an associated data type. The type information is provided to the receiver of the message.<br>With the **element** attribute, the *part* element is an XML document. The document's schema provides the type information for use by the receiver. |

WSDL concepts often take some time to comprehend. Let's look at the same concepts from the client's point of view. Imagine that you are a client who wants to call a web service. You might you want the WSDL description to answer the following questions:

- What is the name of the interface for the client to call?
- Within that interface, what is the name of the method to call?
- For the method, what data must the client send, if any?
- Does the service return a response?
- If there is a response, what data will the response contain, if any?

Table 3 describes the WSDL elements associated with this information.

**Table 3: Information associated with WSDL elements**

| Question | WSDL elements | Comments |
|---|---|---|
| What is the name of the interface? | *portType* | A servicecan have multiple interfaces, each with a unique name. |
| What is the name of the method? | *operation* | Each method in an interface has a unique name. |
| What data must the client send? | *input*, *message*; *part* (optional) | By definition, there is an incoming message; therefore, we need *input* and *message*.<br>The service might not require data from the client, so the message element might not have any *part* elements. |
| Does the service return a response? | *output* (optional) | Some services don't return a message to the client. |
| What data will the response contain? | *message*; *part* (optional) | The part dictates what will be returned. |

Let's look at fragments of a WSDL document from the ABC stock exchange (Figure 9). They have a web service with an interface named StockQuote and a method named getStockPrice. The method receives an input message that contains a piece of data named symbol, which is a string. The method also returns a message that contains a piece of data named price, which is a float.

```
<!-- message elements describe the data passed as input and output -->
<message name="getStockPriceRequest">
    <part name="symbol" type="xsd:string"> </part>
</message>

<message name="getStockPriceResponse">
    <part name="price" type="xsd:float"> </part>
</message>

<!-- portType and operation element describe the interface -->
<portType name="StockQuote">
  <operation name="getStockprice">
   <input name="symbol" message="abc:getStockPriceRequest"> </input>
   <output name="price" message="abc:getStockPriceResponse"></output>
  </operation>
</portType>
```

**Figure 9: Sample WSDL document**

### *Group 2: Binding the interface to a transport*

The abstract interface described by the WSDL elements in group 1 must be bound to a transport. The WSDL binding element uses the schema of the specified transport to define a mapping between the abstract interface and the specific transport.

WSDL defines three separate transport bindings, SOAP, HTTP, and MIME. Each transport has its own schema.

*Note: This article addresses only the SOAP binding because it is the only binding supported by the WS-I Basic Profile.*

The SOAP protocol has a schema for binding to a WSDL interface. The schema defines elements that are used within the WSDL document. These elements define the contents of the SOAP Envelope, Header, and Body.

The SOAP schema for binding defines a number of elements that can influence the way a SOAP message is formatted. This article focuses on the most common elements (and their attributes).

In Figure 10, the elements in bold are defined by the WSDL schema. The elements in italic are defined by the SOAP schema. Note that the WSDL input element, which represents the request from the client, is composed of a SOAP Header and a SOAP Body. Similarly, the WSDL output element, which is the response from the service, is bound to the SOAP Body.

```
<binding>
      <soap:binding/>
        <operation>
           <soap:operation/>
                <input>
                   <soap:header/>
                   <soap:body/>
                </input>

                <output>
                   <soap:body/>
                </output>
         </operation>
</binding>
```

**Figure 10: Binding hierarchy for WSDL and SOAP**

The *header* element allows the application to specify a SOAP header in the header section of the SOAP Envelope. As noted earlier, applications can specify their own headers to extend the SOAP protocol. For example, an application might add a unique header to monitor the activity of a stock broker.

The *binding* element has two attributes of interest: **transport** and **style**. The **transport** attribute specifies the transport that SOAP uses. *Note: HTTP is used in almost all cases and is the only transport supported by the WS-I Basic Profile.*

The **style** attribute defines the default SOAP messaging style to use for the entire interface (portType). The style can have one of two values: RPC or document.

## RPC-style request

If RPC is specified, the SOAP Body will contain the information required to call a method on an interface. The required information is obtained from the WSDL elements. The following table indicates which WSDL elements are to be used when writing the SOAP Body for an HTTP request.

| Element used | Where the data comes from in WSDL |
|---|---|
| Method name | The **name** attribute of the *operation* element |
| Method argument name | The **name** attribute of the *input* element |

Using the previous stock quote example, the WSDL document in Figure 11 highlights the operation and input elements that will be used to build the payload of the SOAP Body. Figure 12 shows the associated SOAP message for this request.

```
<!-- message elements describe the data passed as input and output -->
<message name="getStockPriceRequest">
    <part name="symbol" type="xsd:string"> </part>
</message>

<message name="getStockPriceResponse">
    <part name="price" type="xsd:float"> </part>
</message>

<!-- portType and operation element describe the interface -->
<portType name=”StockQuote”>
  <operation name=”getStockprice”>
   <input name="symbol" message="abc:getStockPriceRequest"> </input>
   <output name="price" message="abc:getStockPriceResponse"></output>
  </operation>
</portType>
```
**Figure 11: WSDL for an RPC-style request**

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
xmlns:abc="http://abc.com/stock">
<soap:Body>
   <abc:getStockPrice>
      <symbol>HPQ</symbol>
   </abc:getStockPrice>
</soap:Body>
</soap:Envelope>
```
**Figure 12: SOAP message for the RPC-style request**

## Document-style request
If document is specified, the SOAP Body will contain raw XML. The XML content is defined by the application. The WSDL shown in Figure 13 is slightly different when using document style. Notice that the part element has an attribute named **element**, which is used to include the employee XML schema that we defined earlier in this article.

```
<!-- include application defined employee schema -->
<types>
  <xsd:schema targetNamespace=http://abc.com/employee>
      <xsd:import namespace=”http://abc.com/employee”
         schemaLocation=”http://abc.com/employee.xsd”/>
  </xsd:schema>
</types>

<!-- message element describes the data passed as input -->
<message name="updateEmployee ">
    <part name="employeeRec" element="abc:employee"> </part>
</message>


<!-- portType and operation element describe the interface -->
<portType name=”HumanResources”>
  <operation name=”EmployeeRecord”>
   <input name="employeeRec" message="updateEmployee"> </input>
</portType>
```
**Figure 13: WSDL for a document-style request**

Figure 14 shows the associated document-style request. Note that the XML is inserted into the SOAP Body without any extra definition. With document-style messaging, the application can send any XML elements without SOAP dictating a structure.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
xmlns:abc="http://abc.com/employee">
<soap:Body>
<employee>
      <name>
        <first>David</first>
        <last>Sullivan</last>
      </name>
      <email>davidjsullivan@hp.com</email>
</employee>
</soap:Body>
</soap:Envelope>
```
**Figure 14: Associated document-style SOAP request**

### *Group 3: Define a service and give it a name*

Group 3 elements create a web service for use by clients. The service element contains one or more port elements. A port element has an associated binding and an address. The address is a unique URI that is used by clients to call this web service. In Figure 15, the web service StockQuoteService is given the unique address `http://abc.com/employee/stockQuote`.

```
<service name=”StockQuoteService”>
  <port name=”StockQuoteService_port”  binding=”StockQuote_RPCbinding”>
    <soap:address location=”http://abc.com/employee/stockQuote” />
  </port>
</service>
```

**Figure 15: WSDL defining the service to be called by clients**

## UDDI

UDDI stands for Universal Description Discovery and Implementation. UDDI registries act as repositories for web services. Web service providers can publish their services for others to use. Web service consumers can “shop” a UDDI registry to find the service that best fits its particular needs.

Web service providers register their business in a UDDI registry. They describe their business along with the services that they provide. A common analogy for describing the properties of a single registration is based on the United States phone book. For each registration there are three different levels of detail available. The first level is like the white pages: it provides basic information about a company, such as the name, address and phone number. The second level is like the yellow pages: it provides a categorization of a service provider and their services. A single service can appear in different categories. For instance,  a category might be based on the type of service or on the service’s geographical location. The third level is like the green pages: it provides the technical details of how to use the service.

A web service consumer uses the UDDI registry to locate services. Typically,  the consumer can find a service in one of two ways: using a web browser and programmatically. The web interface allows a person to browse the many categories in the registry and read descriptions of the business and services. The programmatic API allows applications the ability to browse and discover businesses and their services.

UDDI registries can be public or private. Public registries are used by businesses to advertise their company and the services that they support. A private registry can be used within a company or within a division of a company to provide services to their employees.

Applications use UDDI registries to publish or discover web services. When a web service is developed, a publishing API is required to place the web service description in the registry. When a client wants to look up a web service, a discovery API is required. Some web service platforms provide implementations for both the publishing and discovery programming interfaces. Some other supply only the discovery interface and still others  supply neither.

The UDDI registry is typically used at development time. If the developer already knows which web service they want to use in the application, then a UDDI registry is not needed.

## OpenVMS web service products and tools

The OpenVMS integration strategy encourages partners to provide best-in-breed products whenever available. OpenVMS will continue to port key open source products to provide a range of choices, enabling users to pick the product that best fits their particular situation. Where appropriate, OpenVMS will provide OpenVMS-aware tools to aid in the development and deployment of web services. A number of web service related products have either been ported to OpenVMS or are known to work on OpenVMS. Table 4 lists these products.

**Table 4: Web service related products for OpenVMS**

*Open source*

| **Product** | **Description** |
|---|---|
| SOAP Toolkit V1 | Web services development and deployment toolkit. Based on the older Apache SOAP Toolkit. |
| SOAP Toolkit V2 | Web services development and deployment toolkit. Based on Apache Axis. |
| XML-J | Java XML parser supporting DOM and SAX interfaces. Based on Apache Xerces.<br>Java transformation engine for converting XML document to other HTTP and other XML documents. Based on Apache Xalan. |
| XML-C | C++ XML parser supporting DOM and SAX interfaces. Based on Apache Xerces.<br>C++ transformation engine for converting XML document to other HTTP and other XML documents. Based on Apache Xalan. |
| Apache Ant | Build environment supplied as part of Apache Tomcat. |
| Apache Log4j | Java logging facility used for testing and debugging. |
| NetBeans | Integrated Java development environment with XML support. |

*Partner products*

| **Product** | **Description** |
|---|---|
| BEA WebLogic Server (WLS) | Enterprise grade J2EE application server. |

*HP products*

| **Product** | **Description** |
|---|---|
| HP BridgeWorks | OpenVMS aware application integration middleware. |

## Providing web services for OpenVMS applications

The majority of products listed in Table 4 allow OpenVMS applications written in Java to easily create and consume web services. However, most OpenVMS applications are not implemented in Java. For these applications, more development time is required to wrap their application and expose them as web service. OpenVMS engineering understands the importance of web services for the long-term viability of non-Java applications and intends to provide customers with tools that will simplify the creation of web services for non-Java applications.

In the same way that vendors provide tools to generate web services wrappers from Java applications, OpenVMS intends to provide a web services toolkit to generate web services wrappers from non-Java applications.

In order to understand which components OpenVMS will supply, let's examine the different components of a web services platform. The delivery of web services features can be separated into three distinct categories. Each of these categories is needed to support web services on OpenVMS. Figure 16 illustrates the OpenVMS web services strategy.

- **A web services client to call a web service.** A client is capable of locating and calling a web service. Clients allow OpenVMS applications to reuse services on other platforms. OpenVMS relies on partners and open source projects for supporting web service clients

- **A web services broker to deploy services.** A broker takes an object (usually Java) and makes it available as a web service. OpenVMS relies on partners and open source projects for supporting web service brokers.

- **A bridge from the broker (usually Java) to the legacy application (never Java).** This is a key component in supporting existing applications. It provides the glue between the web service and the existing 3GL application. Without the bridge, only applications written in Java can be easily exposed as services. No partners or open source projects provide this component on OpenVMS. Therefore, OpenVMS will provide both development and run-time components for wrapping 3GL applications. These components will be provided as a toolkit.

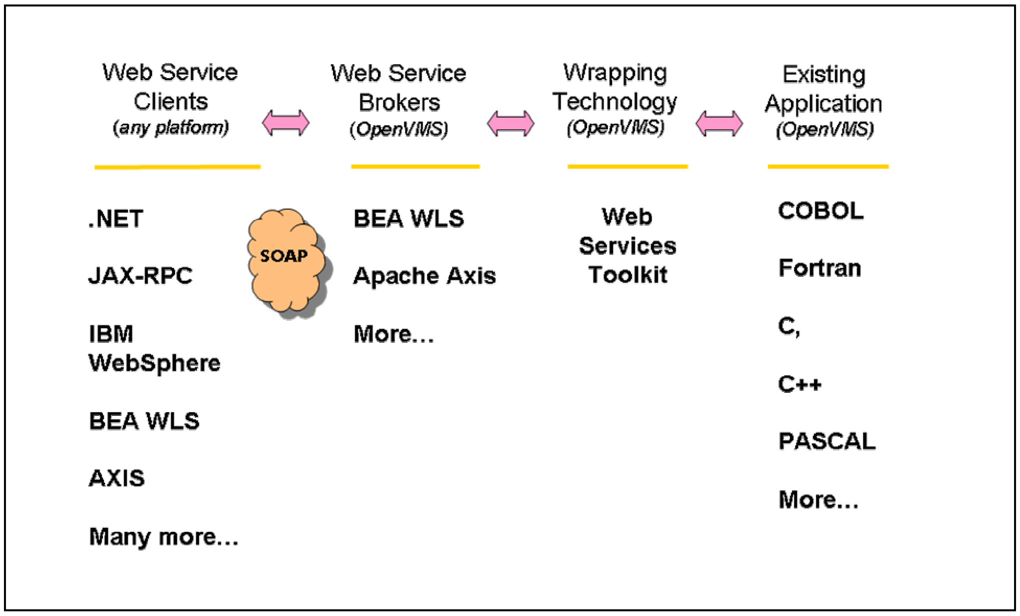


**Figure 16: OpenVMS web services strategy**

## Recommendations

OpenVMS provides a number of products and tools for developing web services. If you have not already begun to evaluate web services, this is the time to do it. Start evaluating the RPC-style and document-style web services designs. Identify the areas in your organization where web services would provide a value.

It is important to remember that web service development platforms can generate code on your behalf. These tools handle the details of WSDL and SOAP. Now that you have an understanding of these technologies, you can alter them as needed to address your unique situation.

Most of the details of the web service technologies can be learned as needed. Most of the key concepts of web services are not new. They have been used in traditional middleware products for decades. You likely already understand more about web services concepts than you think.

Finally, look at the examples in the next section to see how easy it can be to create a web service on OpenVMS and call it from client implementations such as .NET and JAX-RPC.

## Web services examples

As mentioned earlier, developing a web service is not difficult. To illustrate how easy using web services can be, the examples do the following:

- Create a very simple OpenVMS web service.
- Test the services from a web browser.
- Review Java code that uses the JAX-RPC interface to call the OpenVMS service.
- Review C# code that uses Microsoft .NET to call the OpenVMS service.

After you see how easy this process can be, play with the service to make it more interesting. The examples source code is available at the URI:

http://h71000.www7.hp.com/openvms/products/ips/soap/webservices.jar

To extract the contents of the jar file, issue the following command:

```
jar xvf webservices.jar
```

### Install the required software

First, install the products listed in Table 5.

**Table 5: Required software for OpenVMS development environment**

| | |
|---|---|
| Target platform | OpenVMS V7.2-2 or higher |
| Target languag e | Java |
| Required software | Java V1.3.1 or higher<br>http://h18012.www1.hp.com/java/download/index.html |
| | CSWS_JAVA (Apache Tomcat)<br>http://h71000.www7.hp.com/openvms/products/ips/apache/csws_java.html |
| | SOAP_Toolkit V2.0 (Apache Axis)<br>http://h71000.www7.hp.com/openvms/products/ips/soap/soap.html |

### Configure the required software

Verify that each product is installed properly and that all logicals are defined. Run the product's installation verification procedure (IVP), if available.

#### *Step 1:*

After you install the SOAP Toolkit V2.0, run the configuration utility located in the AXIS$ROOT:[AXIS-1_1.OPENVMS.COMS]SOAP_TOOLKIT-2_0_UTIL.COM directory, and select "Copy supplemental jar files to AXIS$ROOT:[AXIS-1_1.LIB]". For example:

```
$ axis$root:[axis-1_1.openvms.coms]soap_toolkit-2_0_util

           SOAP Toolkit 2.0 (based on Apache AXIS) Utility
           -----------------------------------------------
        1 - Show current configuration

        2 - Run SOAP Toolkit 2.0 Validation Procedure

        3 - Copy supplemental jar files to AXIS$ROOT:[AXIS-1_1.LIB]

        E. Exit SOAP Toolkit 2.0 Utility
           ------------------------------------------------
Please choose a task: 3

issuing command: copy/log AXIS$ROOT:[AXIS-1_1.LIBJARS] AXIS$ROOT:[AXIS-1_1.LIB]
/exclude=(xercesImpl.jar,xml-apis.jar)

%COPY-S-COPIED, AXIS$ROOT:[AXIS-1_1.openvms.libjars]activation.jar;1 copied to
AXIS$ROOT:[AXIS-1_1.LIB]activation.jar;1 (107 blocks)
%COPY-S-COPIED, AXIS$ROOT:[AXIS-1_1.openvms.libjars]ant.jar;1 copied to
AXIS$ROOT:[AXIS-1_1.LIB]ant.jar;1 (1440 blocks)
%COPY-S-COPIED, AXIS$ROOT:[AXIS-1_1.openvms.libjars]junit.jar;1 copied to
AXIS$ROOT:[AXIS-1_1.LIB]junit.jar;1 (237 blocks)
%COPY-S-COPIED, AXIS$ROOT:[AXIS-1_1.openvms.libjars]optional.jar;1 copied to
AXIS$ROOT:[AXIS-1_1.LIB]optional.jar;1 (1315 blocks)
%COPY-S-NEWFILES, 4 files created

Choice  (R- Return to Menu)  (E- Exit):
```

#### *Step 2:*

Configure Axis to run under Tomcat by copying the Axis webapps subdirectory under the Tomcat webapps subdirectory. To determine the Tomcat home directory, run the SYS$STARTUP:APACHE$JAKARTA.COM command procedure.

For example, if the Tomcat home directory is DISK$:[CSWS_JAVA.APACHE.JAKARTA.TOMCAT], use the following backup command:

```
$ BACKUP/LOG/IGNORE=INTERLOCK AXIS$ROOT:[AXIS-1_1.WEBAPPS...]*.* -
_$ DISK$:[CSWS_JAVA.APACHE.JAKARTA.TOMCAT.WEBAPPS]
```

#### *Step 3:*

Start Tomcat by running the SYS$STARTUP:APACHE$JAKARTA.COM command procedure.

### Create a simple web service

We will keep the web service as simple as possible. The service simply returns the IP address of the OpenVMS system. From any directory, type or copy the following code into a file named HELLOOPENVMS.JAVA.

```
import java.net.*;

public class helloOpenVMS {

    /** Creates a new instance of helloOpenVMS */
    public helloOpenVMS() {
    }

    public String getAddress() {

        try {

            return InetAddress.getLocalHost().getHostAddress();

        } catch (java.net.UnknownHostException e) {
            return "could not retrieve host IP address";
        }
    }
}
```

### Deploy the web service

***Step 1:***
Verify that JAVA$CLASSPATH is defined and that Tomcat is started.

***Step 2:***
Rename the file a file extension of .JWS (Java web service).

```
$ RENAME HELLOOPENVMS.JAVA HELLOOPENVMS.JWS
```

***Step 3:***
Copy the HELLOOPENVMS.JWS file to the Axis webapps deployment directory under the Tomcat home directory.

For example, if the Tomcat home directory is DISK$:[CSWS_JAVA.APACHE.JAKARTA.TOMCAT], use the following command:

```
$ COPY HELLOOPENVMS.JWS -
_$ DISK$:[CSWS_JAVA.APACHE.JAKARTA.TOMCAT.WEBAPPS.AXIS]
```

Congratulations! Your web service is ready to be tested.

### Test the web service from a browser

Since we have not yet written a web services client, we will use a web browser to test our new service. The browser can be located on any computer that has access to the computer where the web service is deployed.

Type or copy the following URL into the address window of your favorite browser. Make sure you replace *your-computer-name* with the name of the OpenVMS computer that deployed your web service.

`http://`*`your-computer-name`*`:8080/axis/helloOpenVMS.jws?method=getAddress`

This URL calls the web service broker located on *your-computer-name*. The broker is listening on port 8080 for incoming SOAP requests, such as this one. The web service is located in the Axis directory. The name of the web service is helloOpenVMS.jws. We want to call the method getAddress, so we specify this by using the syntax ?method=getAddress.

**Look at the SOAP Response**

After you send the request to the web service, you should see something like the following text returned in your browser. This is the SOAP response from your web service.

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
<soapenv:Body>
  <getAddressResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <getAddressReturn xsi:type="xsd:string">16.32.128.78</getAddressReturn>
  </getAddressResponse>
</soapenv:Body>
</soapenv:Envelope>
```

It is very easy to understand the SOAP response from the web service. Notice that the SOAP Body contains our method getAddressResponse and specifies that the method returned the address 16.32.128.78. (This address will be different for your system.)

**Look at the WSDL**

Let's look at the WSDL generated by Axis for our web service. Type or copy the following URL into the address window of your favorite browser. Make sure you replace *your-computer-name* with the name of the OpenVMS computer that deployed your web service.

`http://`*`your-computer-name`*`:8080/axis/helloOpenVMS.jws?wsdl`

This is the WSDL generated for the preceding SOAP response:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://16.32.128.78:8080/helloOpenVMS.jws"
  xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-
  soap" xmlns:impl="http://16.32.128.78:8080/helloOpenVMS.jws"
  xmlns:intf="http://16.32.128.78:8080/helloOpenVMS.jws"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:message name="getAddressResponse">
    <wsdl:part name="getAddressReturn" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="getAddressRequest">
</wsdl:message>

<wsdl:portType name="helloOpenVMS">
    <wsdl:operation name="getAddress">
        <wsdl:input message="impl:getAddressRequest"   name="getAddressRequest"/>
        <wsdl:output message="impl:getAddressResponse" name="getAddressResponse"/>
    </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="helloOpenVMSSoapBinding" type="impl:helloOpenVMS">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>

    <wsdl:operation name="getAddress">
      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="getAddressRequest">
               <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="http://DefaultNamespace" use="encoded"/>
      </wsdl:input>

      <wsdl:output name="getAddressResponse">
               <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="http://16.32.128.78:8080/helloOpenVMS.jws" use="encoded"/>
      </wsdl:output>
    </wsdl:operation>

</wsdl:binding>


  <wsdl:service name="helloOpenVMSService">
    <wsdl:port binding="impl:helloOpenVMSSoapBinding" name="helloOpenVMS">
      <wsdlsoap:address location="http://16.32.128.78:8080/helloOpenVMS.jws"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

In this WSDL output, the *message* element is named getAddressResponse. It has a *part* element named getAddressReturn, which is a string returned from the service -- in this case, an IP address.

### Writing a simple client using JAX-RPC

Now let's look at the client-side Java code that uses the JAX-RPC interface to call our service.  If you want to create this client yourself, refer to the detailed instructions in the samples download. For this client:

- Target platform: any platform that supports Java (Linux, UNIX, Windows, OpenVMS)
- Target language: Java

The following Java code uses the JAX-RPC interface to call the OpenVMS web service The preceding few lines of code are an entire web services client. In this example, the client code establishes two JAX-RPC objects, Service and Call. The address of the helloOpenVMS web service is set and the method getAddress is invoked. It's all very simple.

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import javax.xml.namespace.QName;

public class getAddress {

    /** Creates a new instance of getAddress */
    public getAddress() {
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        printAddress();
    }

    static void printAddress(){
       try {

            String endpoint =
                 "http://yourcomputername:8080/axis/helloOpenVMS.jws";

            Service service = new Service();

            Call call = (Call)service.createCall();
            call.setTargetEndpointAddress( new java.net.URL(endpoint) );

            String ret = (String) call.invoke( "getAddress" , null);

            System.out.println("got IP address: " + ret);

           } catch (Exception e) {
                 System.err.println(e.toString());
           }
    }
}
```

*Note: JAX-RPC is one of a number of Java interfaces for web services.*

**Writing a simple client using Microsoft .NET**

Now let's look at the client-side C# (C Sharp) code that uses the .NET environment to call our service. If you want to create this client yourself, refer to the detailed instructions in the samples download. For this client:

- Target platform: any Windows platform that supports .NET
- Target language: C#

The following C# code uses the .NET environment to call the OpenVMS web service. This code is also very simple. A .NET web reference is generated using a simple wizard. This web reference generates client-side proxy code that hides details of the invocation of the OpenVMS web service.

For a description of how to create a web reference, refer to the detailed instructions in the samples download.

```
using System;
using webreference.proxy;
namespace addressPicker
{

  class Class1
  {

     /// The main entry point for the application.
     [STAThread]
      static void Main(string[] args)
      {
         helloOpenVMSService myserv = new helloOpenVMSService();

         String ret = myserv.getAddress();

         System.Console.WriteLine("the Service returned the string " + ret);
      }
   }
}
```

## Summary

Web services provide both a tactical and strategic advantage to businesses. Simplicity, low cost, and high cross-platform availability have led to an unprecedented acceptance of web services by enterprise software vendors. OpenVMS customers can use web services to reduce costs while increasing the speed and quality of development projects.

OpenVMS provides best-in-breed products to its customers. With the Web Services Toolkit, applications have new and unique opportunities to leverage OpenVMS strengths from other platforms, such as Microsoft .NET and J2EE.

## For more information

The examples used in this paper are available from the following location:

http://h71000.www7.hp.com/openvms/products/ips/soap/webservices.jar

Feel free to contact the author of this paper by sending email to **davdjsullivan@hp.com**.

# OpenVMS Technical Journal

## Adding a Friend to T4 and Friends

## Incorporating BEA WebLogic Server 8.1 Performance Data

Pat McConnell

Performance Group, OpenVMS Engineering

Pat.McConnell@hp.com

### Overview

This OpenVMS Technical Journal article illustrates an addition to the T4 and Friends family originally introduced in Steve Lieman's "*TimeLine-Driven Collaboration with T4 and Friends: A Timesaving Approach to OpenVMS Performance*".[1] This particular addition to the family is concerned with gathering key BEA WebLogic Server 8.1 performance information and correlating that information with the OpenVMS performance information gathered by T4. The primary purpose of this article is to provide a template for integrating a new monitoring application into the default implementation of T4. The sample BEA WebLogic Server 8.1 monitoring T4 application outlined in this article is available by request from the author.

### Introduction

This work was undertaken during the chaos of an internal benchmarking exercise within OpenVMS Engineering, so there were several important influences that should be brought out to frame this sample extension to the default T4 implementation.

First, the internal benchmarking exercise was focused on the effect of a BEA WebLogic Server 8.1 based workload on OpenVMS. Therefore, it had already been decided to utilize T4 to monitor the OpenVMS systems involved in the benchmark, and that obtaining all possible BEA WebLogic Server performance statistics was not a goal of the benchmark. Second, like all efforts of this type, time was at a premium, meaning that the programming involved with this effort had to be expeditious rather than complete and comprehensive. That said, the resulting code and modifications to the DCL procedures comprising T4 could be valuable to anyone interested in correlating OpenVMS and BEA WebLogic Server 8.1 performance metrics.

### BEA WebLogic Server 8.1 Performance Metrics

Examination of the BEA WebLogic Server 8.1 documentation indicated that a large, comprehensive set of application performance metrics were available for retrieval via JMX. For this particular set of constraints, it was considered impractical to develop a complete monitoring application that obtained all possible performance metrics. It was important to determine the key WebLogic Server performance metrics that captured the essence of the server's performance. A search of the BEA WebLogic Server 8.1 documentation set revealed the document, *Programming WebLogic Management Services with JMX*, which contained a section entitled, **Best Practices: Commonly Monitored Attributes**. This section detailed a number of BEA WebLogic Server 8.1 attributes that provide a general overview of the performance of a WebLogic 8.1 server. Table 1 briefly names and describes these attributes, but an examination of the aforementioned section within the *Programming WebLogic Management Services with JMX* document is necessary for a thorough understanding of the performance attributes and their access mechanisms.

[1] VTJ Volume Three, January 2004

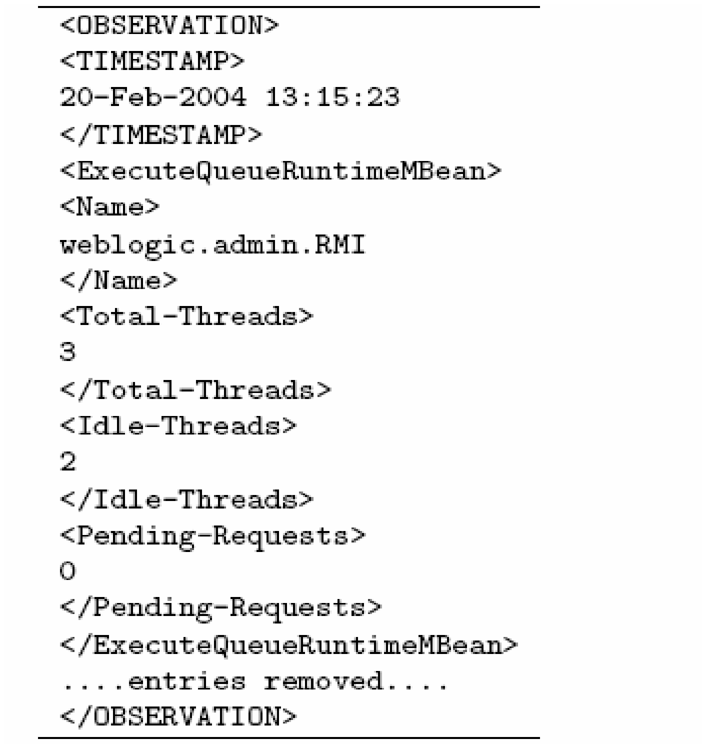
**Table 1**

| Attribute Name | Description |
|---|---|
| `State` | State of Server |
| `HeapSizeCurrent` | Server's JVM heap size |
| `ActiveConnectionsCurrentCount` | Active JDBC connections |
| `ConnectionsHighCount` | JDBC connection pool high water mark |
| `LeakedConnectionCount` | Total number of leaked connections |
| `OpenSocketsCurrentCount` | Current number of open sockets |
| `AcceptBacklog` | Number of requests that can be backlogged |
| `ExecuteThreadCurrentIdleCount` | Idle threads in default execute queue |
| `PendingRequestCurrentCount` | Waiting requests in Server's default execute queue |
| `ActiveConnectionsCurrentCount` | Active connections to a JDBC connection pool |
| `ConnectionDelayTime` | Average time to connect to a JDBC connection pool |
| `FailuresToReconnect` | Connect failures for a JDBC connection pool |

## Collaborating with T4

The next item of business was to determine how to obtain these metrics within the context of the T4 and Friends framework. The ultimate goal of the T4 and Friends framework is to develop a regularly spaced timeline of all performance metrics. The current implementation of T4 is implemented with the assumption that all metrics of interest are available at each interval of the measurement period. However, in reviewing the key BEA WebLogic Server 8.1 performance attributes mentioned previously, it was clear that there might be a varying number of entries under some of the attributes from one interval to another. Therefore, it was decided that an intermediate XML file would be written by the BEA WebLogic Server 8.1 monitoring program.

The intermediate XML file written by the BEA WebLogic Server 8.1 monitoring program is a minimal one whose main purpose is to capture the run parameters for the monitoring session, and then capture the value of the BEA WebLogic Server 8.1 performance attributes at each measurement interval. A later processing step will convert this XML file to a file format appropriate for T4 analysis. A partial example from a measured system can be found in Figure 1.



```
<OBSERVATION>
<TIMESTAMP>
20-Feb-2004 13:15:23
</TIMESTAMP>
<ExecuteQueueRuntimeMBean>
<Name>
weblogic.admin.RMI
</Name>
<Total-Threads>
3
</Total-Threads>
<Idle-Threads>
2
</Idle-Threads>
<Pending-Requests>
0
</Pending-Requests>
</ExecuteQueueRuntimeMBean>
....entries removed....
</OBSERVATION>
```

**Figure 1: Intermediate Partial XML Contents**

The BEA WebLogic Server 8.1 monitoring program, `ServerExQInfo`, was designed to accept inputs compatible with those used within the key collection procedure of T4, to regularly sample the BEA WebLogic Server 8.1 performance attributes, and to write to the intermediate XML file. The processing logic of the program is straightforward and consists of the following steps.

- Accept input parameters, validate, instantiate `ServerExQInfo`
- Create monitoring thread, hold execution until start time
- Monitor BEA WebLogic Server 8.1 via JMX MBean accessor functions at each interval
- Write observations to XML output file
- End monitoring and close XML file at the provided end time

## Extending T4 Default Implementation

Adding a friend to T4 and Friends often requires modifying the DCL procedures that implement T4, so a quick review of T4 processing logic follows.

- `T4$CONFIG` is run to gather parameters
- `T4$COLLECT_V33` is submitted to a batch queue
- `T4$COLLECT_V33` executes on a batch queue to spawn collectors
- `T4$COLLECT_V33` then does post-processing at the end of the monitoring period
- A composite T4 CSV file is then available for analysis

## Changes to T4$COLLECT.COM

The first modification made to the T4 DCL procedures to integrate the BEA WebLogic Server 8.1 performance monitoring functionality was made to the T4$COLLECT_V33 procedure. Within this procedure, the first change was to determine whether to initiate BEA WebLogic Server 8.1 monitoring. Figure 2 shows where in the procedure the change was made, as well as the code that determines whether monitoring is desired. Figure 3 illustrates the contents of the file that controls which BEA WebLogic Server 8.1 instances are monitored.

```
$! NOTE THAT MANY LINES HAVE BEEN DELETED TO THIS POINT
$! TO ILLUSTRATE WHERE TO PLACE WebLogic Server HOOK
$! INTO T4$COLLECT_V33.COM
$!
$! First of all, spawn the MONITOR data collection ...
$!
$ Mem_Size = "("+F$String(f$GetSyi("MemSize")/128)+"Mb"
$ Avail_Cpus = " with ''F$GetSyi("AvailCpu_Cnt")' cpu(s))"
$ Spawn/NoSymbols/NoLogicals/NoWait/Process="''This_Pid'_MON" -
 Monitor/Record=T4_'This_Node'_'Today'_'St_Et'_Mon.Dat -
 /Interval='P6 -
 /Flush_Interval='P6 -
 /Begin="''Start_Time'" -
 /End="''End_Time'" -
 /Comment="''F$GetSyi("Hw_Name")' ''Mem_Size'''Avail_Cpus'" -
 /NoDisplay           All_Classes
$
$
$!  WebLogic Server hook - if present, pass exactly the same driving parameters
$!  to WebLogic Server code.
$!
$ if F$Search("t4$sys:wlstltTab.wls") .Nes. "" Then -
    @t4$sys:wlstlt.Com "''P3'" "''Start_Time'"     "''End_Time'" -
                    "''P6'" "''Today'_''St_Et'" "''P7'" "''This_Pid'"
$!
$!
$! MANY LINES DELETED FROM THIS POINT IN T4$COLLECT_V33.COM
$! TO ILLUSTRATE WHERE TO PLACE WebLogic Server HOOK
$!
$ ENDSUBROUTINE
```

**Figure 2: Changes to T4$COLLECT Procedure**

```
$ WLStlt.wls - file that controls T4 initiated monitoring of
$ WebLogic Server instances.
$ WLSName,WLSURL,WLSPort,WLSAdmin,WLSAdminPassword,Monitor This Instance?,BEAHome
mydomain,t3://monitor.this.system,7001,weblogic,weblogic,Y,dkd400:[bea]
```

**Figure 3: Control of Monitoring File Contents**

## WLSTLT.COM

This modification to `T4$COLLECT_V33` invokes the DCL procedure, `WLSTLT.COM`, which creates a DCL procedure for each BEA WebLogic Server 8.1 instance that the control file indicates should be monitored. Each of these created DCL procedures will be submitted to the batch queue specified in `T4$CONFIG.COM`. These created procedures are the workhorses of the BEA WebLogic Server 8.1 instance monitoring, using `ServerExQInfo` to access the BEA WebLogic Server 8.1 performance metrics, and recording the observations in an XML file.

### Creating a Composite T4 CSV File

At the end of the monitoring session, the T4 procedure `T4$COLLECT_V33` creates a T4 CSV file that aggregates all of the observations made by the various T4 monitors. In this sample T4 and Friends application, the XML file output is not available to T4 to do automatic post processing, so a different approach is required. At the end of each `WLSTLT.COM` execution, the XML file output is converted to a T4 CSV style file, and then each of these files are appended to the T4 CSV file produced via `T4$COLLECT_V33` with the T4 utility, `T4$APRC`. Once this work is done, the resulting composite T4 CSV file contains not only OpenVMS system performance metrics in a timeline format, but the corresponding BEA WebLogic Server 8.1 performance metrics as well. Figure 4 illustrates this situation by showing the output of a sample visualization program where both an OpenVMS and BEA WebLogic Server 8.1 metric are plotted.

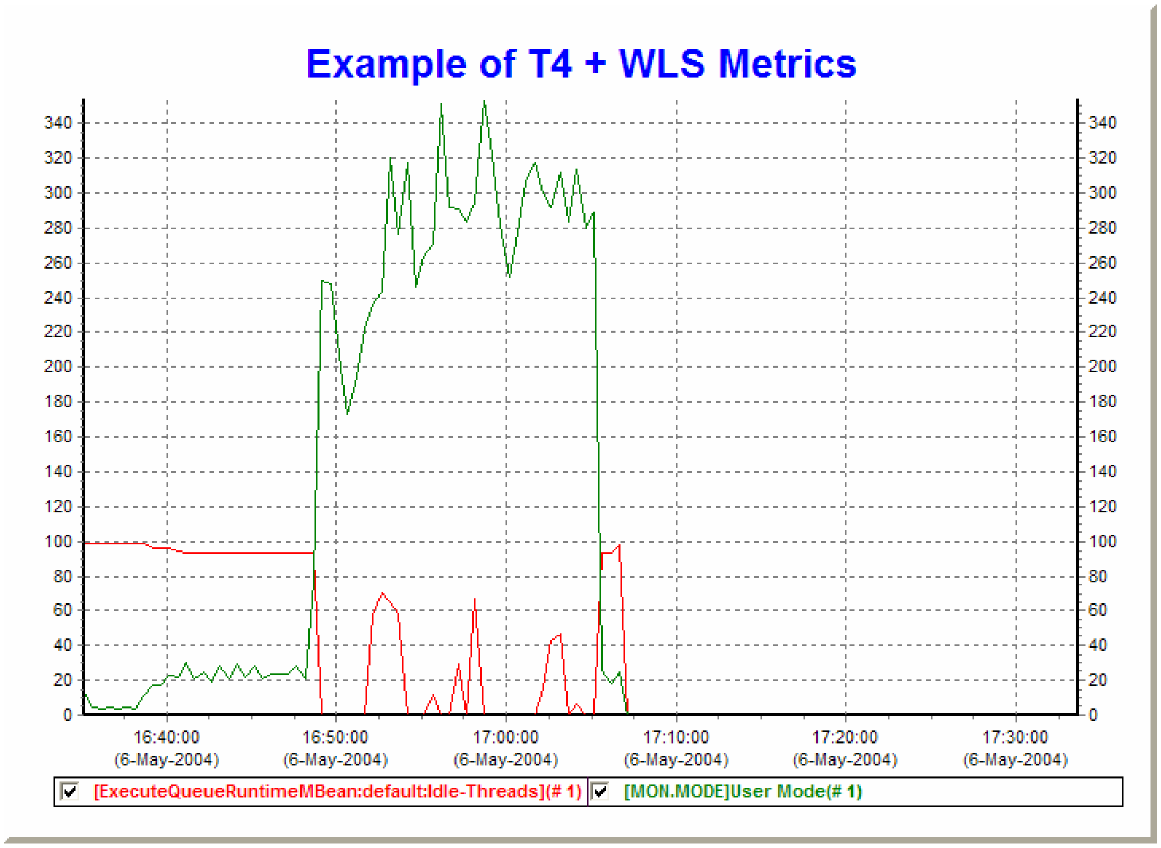


**Figure 4 - Example of Merged T4 and WLS Statistics**

## Summary

This article outlined the steps necessary to extend the default T4 implementation to include the ability to obtain key BEA WebLogic Server 8.1 performance metrics, and then to integrate the BEA WebLogic Server 8.1 information into the composite T4 CSV for later processing by members of the T4 and Friends family. The steps that are outlined within this article can be abstracted to develop other T4 extensions for whatever an OpenVMS performance analyst requires.

## For more information

"*TimeLine-Driven Collaboration with T4 and Friends: A Timesaving Approach to OpenVMS Performance,*" VMS Technical Journal, Volume Three - February 2004

This article is the standard reference for understanding the Timeline Collaboration concept which has driven the development of T4 and Friends. A reading of this article will give the necessary background for the development of extensions to the default T4 implementation.

BEA WebLogic Server Programming WebLogic Management Services with JMX

Release 8.1

Revised: July 18, 2003

This manual presents the use of JMX by BEA WebLogic Server 8.1 to provide access to its performance metrics. It is a clearly written reference that contains numerous Java examples.

SYS$ETC

As mentioned in the body of the article, a default T4 implementation is included in this library from OpenVMS v7.3-2 onward.

# OpenVMS Technical Journal

## HP OpenVMS Clusters and IBM MQSeries Failover Sets

John Edelmann, Technology Consultant

## Overview

This article focuses on the use of IBM MQSeries Failover sets within an OpenVMS Cluster environment. The goal of such a configuration is to maximize MQSeries (messaging middleware) availability.

IBM's product, MQSeries (currently re-branded WebSphere MQ) is a robust and guaranteed delivery messaging subsystem, which supports many diverse vendors' operating systems and hardware servers. This article explores and details high availability deployment on OpenVMS Clusters using a feature called Failover Sets. The combination of this IBM-developed failover capability coupled with OpenVMS Cluster availability, results in a very nearly fault-tolerant environment for customers relying on IBM's product for messaging middleware. It is, therefore, a unique solution.

This article assumes that the reader is familiar with OpenVMS Cluster concepts and describes, with examples, how the failover mechanism can be deployed.

### MQSeries - Basics

MQSeries (recently renamed WebSphere MQ) is a messaging middleware product whose primary function is the transfer of a datagram from one application to another on one computer system, or from one application to an application running on another computer system.

There are many advantages to such a messaging model, with the primary ones being:

- Ease of use from the application standpoint.
- The benefit of secured, guaranteed, and trusted delivery of an individual message.
- The availability of a robust array of recovery features.

Unlike most other message or file transfer protocols (FTP for example), incorporation of MQSeries as a middleware component aids in decoupling message delivery details from other processing that application programs perform.

MQSeries, as with most such messaging products on the market today, is only as robust as the integrity of the underlying operating system and platform allows. Thus, it is both proper and fortuitous that IBM exploited the unique active-active clustering features of the OpenVMS operating system, when it designed the unique fail-over capabilities embodied in the MQSeries 5.1 release.

### MQSeries Clusters

MQSeries includes a powerful clustering capability (not to be confused with OpenVMS Clusters), that provides for a number of queue managers, running on the same or different computer servers/platforms, to share the same named queue object to any application interested in writing a message to the queue. An application writing a message from the context of a queue manager which actually hosts the queue in question will always put messages to the queue of the local queue manager. This is not the real intent behind clustering, however.

An application putting a message to a queue from a queue manager which is a member of the cluster, but which *does not have a local instance of the queue*, will put the message to the queue hosted by other members of the cluster, in a round robin methodology, and with a certain level of fail over provided. This assumes that processes reading the messages from the cluster queues are able to process them in a similar manner.

MQSeries Clusters are most useful for multi-site operations, where the various sites are connected by a WAN or LAN (thus, single or multiple customers put messages to a multi-site operation for workload distribution and high availability), or where multiple sites with many application queues and independent servers need to share queues among themselves. In both cases, the administration of sender/receiver channel pairs is drastically reduced, as well as the number and complexity of remote queue definitions and other standard MQSeries connectivity requirements.

In spite of the inherent availability and workload balancing MQ Clusters provide, there is nevertheless the opportunity for system crashes and other unplanned outages. When such events occur, applications reading the messages from the clustered queues hosted by the failing machine are no longer available for processing. Queue managers communicating with the failed machine likewise are no longer able to communicate with it.

Due to the reliance on standard inter-process communication heartbeat and keep-alive features that may or may not adequately stop a running channel to a failed system (and hence, its queue manager), messages may inadvertently be left uncommitted in a channel until the channel is activated again. This recovery may take a while, and thus, the messages in transit will become unavailable for a time.

For these reasons, an MQ cluster is a valuable asset, but not the most robust implementation that the industry has to offer.

Please refer to the MQ Clustering Manual or other basic IBM MQSeries product documentation for a more in-depth review of the product set.

## OpenVMS Clusters and MQSeries Failover

In OpenVMS, in spite of the robustness of the cluster-locking mechanism, a given queue manager can run only on one node of an OpenVMS Cluster at a time. However, each system can run one or more *different* queue managers.

This becomes especially valuable with multiple queue managers running on separate nodes in the cluster, all being members of an MQSeries Cluster. In such a configuration, OpenVMS clustered applications can read messages being written to MQ Cluster shared queues, hosted by queue managers distributed throughout the OpenVMS cluster. This arrangement provides great processing power among applications requiring shared memory or other resources, while maximizing availability to members of the MQ Cluster on other distributed systems (whether they are running OpenVMS or not).

But an even greater development was deployed in Version 5.1. This feature, called Failover Sets, enables a queue manager to be automatically restarted on another OpenVMS Cluster node if a failure occurs. While the feature can be used with or without MQ Clusters, it certainly enhances the overall efficiency and power of the solution, in either case.

A Failover Set is defined as a single queue manager and the nodes across which it can run. There can be multiple Failover Sets on a given OpenVMS Cluster or standalone system, but each Failover

Set can control only a single queue manager. The *set* component of the name refers to the set of OpenVMS Cluster nodes on which the queue manager can run (it can run only on one node in the cluster at a time).

One or more Failover Sets can be configured into an MQ Cluster, which provides the most robust configuration, mentioned previously.

There are several important issues with regard to MQ 5.1 on OpenVMS that must be considered in advance. At the time of this printing, ECO 2 was available (with support on OpenVMS 7.3-1) and ECO 3 was in process. The latter release is intended to support OpenVMS 7.3-2, as well as to address various important internal issues.

## Manageability Tools on OpenVMS

As with most distributed platforms, MQSeries on OpenVMS can be managed locally via the **runmqsc** command utility, or, it can be managed remotely via the IBM MQSeries Explorer tool. This tool runs on Windows NT, 2000, and XP and uses the SYSTEM.ADMIN.SVRCONN channel to allow privileged access to the queue manager. Usually, it is necessary to create a *userid* within the UAF of the OpenVMS system identical to that of *userid* of the windows session and for the OpenVMS user to be granted the MQM identifier. It is also essential that the MQM identifier be granted with the resource attribute. Failure to do so will cause problems during certain remote management tasks (modifying, stopping, or starting channels, especially).

## Considerations for MQ 5.1 on OpenVMS

In an OpenVMS Cluster, it is critical that a common SYSUAF and Rightslist be used across all nodes of the cluster. If this is not possible, the MQM identifier, *userid*, and UICs must be identical on all systems. This commonality must be in place before any MQSeries is installed on another node in the cluster. If this is not done, the new installation will not have the ownership it requires in the common MQS_ROOT directory.

When defining the MQS_ROOT as a device NOT on the system disk, the LIB subdirectory is not created in the proper location. This causes problems with the command server and other issues relative to data conversion. The LIB subdirectory tree must be placed (copied) under the MQS_ROOT:[MQM] directory tree. The location of the directory following the install is SYS$SYSROOT:[MQM].

It is recommended to use **runmqlsr** to initiate channel listening programs, specifically in the START_QM.COM procedure by uncommenting the provided commands after the queue manager starts. This creates a *<queue-manager_LS>* process. If this feature is used, the **endmqlsr** command must also be uncommented in the END_QM.COM procedure. A 20-second wait added just before the **endmqlsr** command may assist with some timing difficulties (to allow the queue manager to fully shutdown).

When configuring ports used for multiple queue managers (Failover Sets) running in a single OpenVMS Cluster environment, it is important to assign different ports to each Failover Set, especially if the same LAN is used for all such connections. During a failover condition, when the potential exists for having more than one queue manager running on a single system, different port assignments must be used to ensure separation between receiver channels using IP aliases to the same interface, regardless of the number of interfaces physically used on a given server.

## Failover Set Details

The basic idea behind a Failover Set is to permit a given queue manager to be restarted on another node in the OpenVMS Cluster if a failure is encountered by the server on which it is originally running. This is accomplished from a data standpoint by the inherent cluster-wide access afforded by the lock manager in an OpenVMS cluster. It is accomplished from a network connectivity standpoint by incorporating alias IP addressing on a given Internet interface (details are provided in the manuals for varying TCP/IP product variants).

Thus, once a failure is detected, and a queue manager is moved to another node in the cluster, the queue manager is started, and the designated IP address for the Failover Set (which is specifically *not* the same as the IP address of the node itself) is added as an alias to one of the network interfaces on the machine.

This permits the re-connection (automatically with a 2-3 minute delay) by distributed MQ clients and servers using a fixed IP address to the same queue manager, on a different node in the cluster, with neither a client or distributed server being aware of the hardware hosting change. They experience only a channel interruption that will clear either by a manual restart or an automatic retry. It is critical that any such failover complete successfully; a residual alias to the wrong interface (regardless of server) can cause inbound channels to fail. On OpenVMS, the INET master process reports evidence of such a situation, via OPCOM.

A Failover Set consists of an initialization file (FAILOVER.INI), and 3 command procedures, (START_QM.COM, TIDY_QM.COM, END_QM.COM). A Failover Set requires a failover monitor process, which runs each node in the OpenVMS Cluster designated to run the queue manager in question. The resulting process monitors the status of the queue manager to which it applies. To establish a Failover Set, the template FAILOVER.TEMPLATE in MQS_EXAMPLES: is copied to:

```
MQS_ROOT:[MQM.QMGRS.qmgrname]FAILOVER.INI
```

It is customized for the Failover Set in question. These instructions are explained in Chapter 16 in the *MQSeries for Compaq OpenVMS Alpha, Version 5 Release 1 System Administration Guide*.

The FAILOVER.INI file contains all symbol definitions used to establish the Failover Set. These include log directories for errors (which includes the startup and shutdown logs), IP addresses for the interfaces that will support the queue manager, and any other startup information required for automated startup and shutdown. An example FAILOVER.INI file is appended to the end of this article.

The following figures show simple Failover Set configurations. A normal configuration (Figure 1) and a failed configuration (Figure 2) are provided.

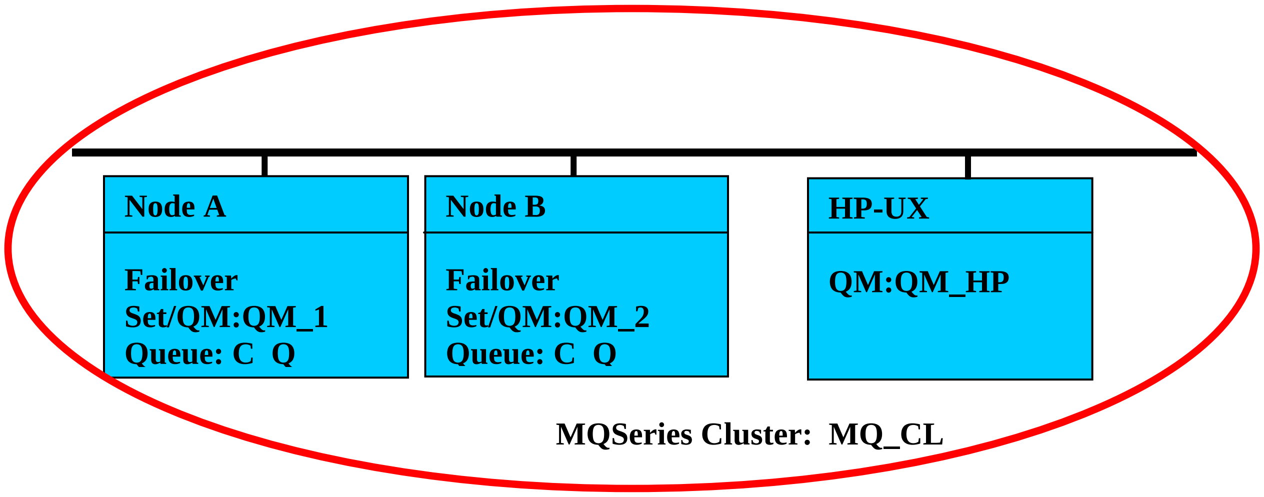


**Figure 1 – Normal Configuration**

Node A and B are OpenVMS Cluster members. Node A and B each support a distinct Queue Manager Failover Set, each of which is a member of an MQ Cluster (MQ_CL), which includes an HP-UX server, running queue manager QM_HP.

Applications running on QM_HP put messages to the cluster queues, C_Q, hosted by QM_1 and QM_2, in a round robin manner. The naming convention used for queue managers and node names is specifically decoupled, so as to reduce unnecessary linkage between a Failover Set and the node on which it runs.

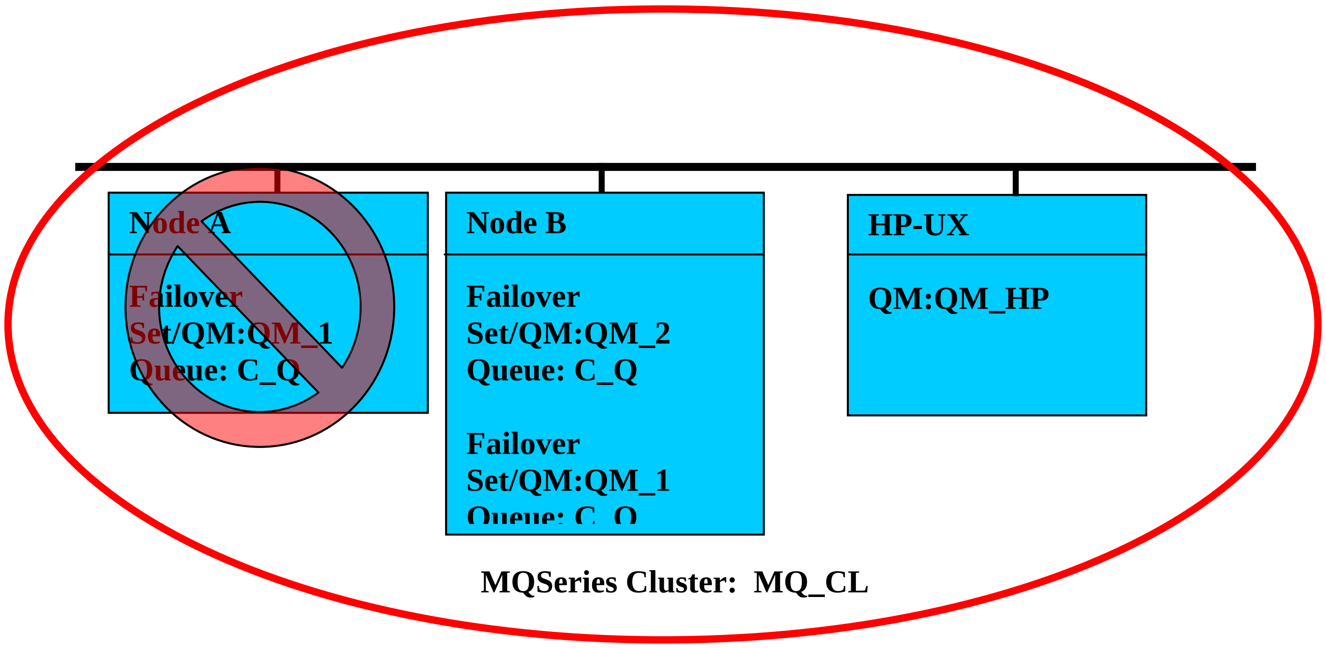


**Figure 2 – Failover Configuration**

Node A has crashed or is otherwise unavailable. The Failover monitor for QM_1 running on Node B has determined that Failover Set QM_1 is no longer running on Node A. Because the Failover Set was configured to run on Node A at priority 1 and Node B at priority 2, the queue manager QM_1 will now start on Node B, using the START_QM procedure specified for QM_1. This procedure ensures that the proper IP address alias is established so that listener programs respond to channel start requests appropriately. Thus, both OpenVMS MQSeries Cluster queue managers are now running on Node B. Data that was in process on QM_1 is now accessible to processes running on Node B, and queue manager availability is sustained throughout the failure of Node A. Any links that existed between QM_HP and QM_1 when it was running on Node A are re-established automatically to Node B, where the queue manager is presently running.

## Summary

In a distributed MQSeries Clustered environment, one might be led to believe that the round-robin and failover capability inherent in that configuration would be adequate for high availability solutions. However, should a processing latency exist for messages delivered to a queue on a node in the OpenVMS Cluster that subsequently fails, those messages would be unavailable until the node rejoins the cluster. With the Failover Set feature, such messages can be accessible as soon as the queue manager on the failed server is restarted on another node in the OpenVMS Cluster.

This is a unique feature that is available with OpenVMS Clusters, owing to their Active-Active clustering design.

Again, much more detailed concept descriptions and configuration guides are available in Chapter 16 in the *MQSeries for Compaq OpenVMS Alpha, Version 5 Release 1 System Administration Guide*.

## For more information

For more information about this topic, please contact John Edelmann at john.edelmann@hp.com.

## Examples

Below is a display of the processes across the clusters that are related to the queue managers running on a two-node cluster. The queue managers are QMD0VMS1 and QMD0VMS2 respectively. The _FM processes are the Failover Set monitors. Note that a monitor for each Failover Set runs on each node of the cluster, whether or not that node is presently running the queue manager at the time or not.

```
DAYEDI> sho sys/proc=q*/cluster

OpenVMS V7.3  on node DAYEDI  20-MAY-2002 13:23:26.06  Uptime  10 02:31:49

  Pid    Process Name    State  Pri      I/O       CPU       Page flts  Pages
21600405 QIO$CONFIGURE   HIB      9       67   0 00:00:00.01       156     64 M
21600411 QUEUE_MANAGER   HIB     10      185   0 00:00:00.04       133    170
21600454 QMD0VMS1_FM     HIB      7      211   0 00:00:02.09       362    327
21600455 QMD0VMS2_FM     HIB      7      132   0 00:00:00.18       360    325
21600462 QMD0VMS1_EC     HIB      6    77241   0 00:00:00.43     54686    160
21600463 QMD0VMS1_LG     HIB      6      299   0 00:00:00.03       449    480  S
21600464 QMD0VMS1_CP     HIB      6      494   0 00:00:00.06       412    443  S
21600465 QMD0VMS1_RM     HIB      6    76189   0 00:00:00.22       477    496 MS
21600466 QMD0VMS1_CI     HIB      6    75981   0 00:00:00.08       569    437 MS
21600467 QMD0VMS1_AG     HIB      6   253029   0 00:00:25.82      1046    887 MS
21600468 QMD0VMS1_LS     HIB      6      735   0 00:00:00.16       580    499 M
21600469 QMD0VMS1_CS     HIB      6   142587   0 00:00:00.08       617    403

OpenVMS V7.3  on node DAYVMS  20-MAY-2002 13:23:26.07  Uptime  10 02:44:22

  Pid    Process Name    State  Pri      I/O       CPU       Page flts  Pages
21400105 QIO$CONFIGURE   HIB      8       67   0 00:00:00.03       230     62 M
21400111 QUEUE_MANAGER   HIB     10      354   0 00:00:00.38       165    196
2140015D QMD0VMS1_FM     HIB      6      104   0 00:00:02.97       476    249
2140015E QMD0VMS2_FM     HIB      6      133   0 00:00:03.35       447    249
21400160 QMD0VMS2_EC     HIB      6    77381   0 00:01:18.78     26676    552
21400161 QMD0VMS2_LG     HIB      6      277   0 00:00:00.63       588    306  S
21400162 QMD0VMS2_CP     HIB      6      496   0 00:00:00.61       598    304  S
21400163 QMD0VMS2_RM     HIB      5    76217   0 00:00:26.04       827    350 MS
21400164 QMD0VMS2_CI     HIB      6    76048   0 00:00:26.08       977    311 MS
21400165 QMD0VMS2_AG     HIB      5   250657   0 00:02:59.54      1942    566 MS
21400166 QMD0VMS2_LS     HIB      6      230   0 00:00:00.54       670    310 M
21400167 QMD0VMS2_CS     HIB      6   161593   0 00:00:37.77       764    304
```

**Example Failover Set initialization file for Queue Manager QMD0VMS1:**

```
# OpenVMS Cluster Failover Set Configuration information – QMD0VMS1
# -------------------------------------------------------
#
# The TCP/IP address used by the OpenVMS Cluster Failover Set
#
IpAddress=nnn.nnn.nnn.nnn
#
# The TCP/IP port number used by the MQSeries Queue Manager
#
PortNumber=1415
#
# The timeout used by the EndCommand command procedure
#
TimeOut=30
#
LogDirectory=mqs_root:[mqm.QMGRS.QMD0VMS1.errors]
#
# The number of nodes in the OpenVMS Cluster Failover Set. The
# number of nodes defined below must agree with this number
#
NodeCount=2
#
# The Name of the OpenVMS node
#
NodeName=DAYEDI
#
# The TCP/IP interface name for the node
#
Interface=ie0
#
# The priority of the node
#
Priority=1
#
# The Name of the OpenVMS node
#
NodeName=DAYVMS
#
# The TCP/IP interface name for the node
#
Interface=we1
#
# The priority of the node
#
Priority=2
```

**Example Failover Set initialization file for Queue Manager QMD0VMS2:**

```
# OpenVMS Cluster Failover Set Configuration information – QMD0VMS2
# --------------------------------------------------------
#
# The TCP/IP address used by the OpenVMS Cluster Failover Set
#
IpAddress=nnn.nnn.nnn.nnn
#
# The TCP/IP port number used by the MQSeries Queue Manager
#
PortNumber=1416
#
# The timeout used by the EndCommand command procedure
#
TimeOut=30
#
LogDirectory=mqs_root:[mqm.QMGRS.QMD0VMS2.errors]
#
# The number of nodes in the OpenVMS Cluster Failover Set. The
# number of nodes defined below must agree with this number
#
NodeCount=2
#
# The Name of the OpenVMS node
#
NodeName=DAYVMS
#
# The TCP/IP interface name for the node
#
Interface=we1
#
# The priority of the node
#
Priority=1
#
# The Name of the OpenVMS node
#
NodeName=DAYEDI
#
# The TCP/IP interface name for the node
#
Interface=ie0
#
# The priority of the node
#
Priority=2
```

# OpenVMS Technical Journal

## Advanced Server for OpenVMS

Hans Hosang

Advanced Server Competence Center Europe, Netherlands

### Overview

This paper is meant to provide you an insight in Advanced Server for OpenVMS.

Discussed are the usage of the product, like its use in clusters and the roles of the server in various domains depending on the Microsoft functionality that you use.

I will try to shed some light on the fields of troubleshooting, monitoring, performance and tuning.

I will briefly talk about future plans for Advanced Server.

### Introduction

Advanced Server for OpenVMS, a file and print server based on Microsoft Windows NT code, gives you most of the Windows NT functionality on OpenVMS.

Printing is one of the Windows functionalities that are supported by Advanced Server. If you have printer queues on your server, you can setup these printers to be used by your Windows based PCs. In the last version the ease of use of printers has been transformed to the way Windows does it. You can manage your printers from any Windows PC and install printer drivers on the server to be used when you setup this printer on a PC.

Windows functionality like the support of domain structures is natural to Advanced Server. Advanced Server can be leading in a domain, maintaining the user community for an organization. Advanced Server can also play a Backup role in a Windows domain or just be a member server. Supporting domains can also mean supporting inter-domain relations like trusts and we do.

You can separate your user's login domain from resource domains and base your security on the user's login domain by trusting that domain.

As we speak about trusting, OpenVMS still is a widely trusted and reliable operating system.

Many customers use Advanced Server to share out the data of their OpenVMS applications to the world of Windows, either for presentation purposes or to use the data in other programs.

Amongst the largest users of Advanced Server are banks, stock exchanges, lotteries and manufactories.

So basically anywhere where the reliability of OpenVMS is needed you can find Advanced Server.

There are other ways of using Advanced Server; several customers use it as the primary source of security in their Microsoft domain. Here again the reliability of OpenVMS is the main reason for the use of Advanced Server.

Before going into more detail about the functionality, let's view some history.

### History

Formerly known as PCSA and Pathworks, Advanced Server has been around since 1988 and has lived through many changes.

A brief overview:

PCSA ( V3.x and lower ) 1988

> Disk services over LAST – LAD (most important)
> Most information was stored, not shared, in a container file on OpenVMS. Sharing was only possible if the container was "mounted" read-only.
>
> File service over DECnet
> From version 2.0 onwards it was possible to directly share files from the OpenVMS file system. DECnet was the only protocol used in those days.

PATHWORKS V4.x ( MSNET server ) 1991

> Remote boot using LAD (Disk Service)
> As all you needed to boot a PC fitted on a floppy, you could create floppy container files to boot PCs off the server and thus secure and centrally control the boot environment.
>
> TCP/IP protocol added for File Service.

PATHWORKS V5.x ( LAN Manager 2.2 server ) 1994

> RIPL Remote Boot (from File Service)
> Slowly abandoning the disk service container files, remote boot was transformed to start off a special file service. This made it more easy to make changes since you didn't need to dismount the read-only container file for every change.
>
> NETBEUI protocol added for File Service
>
> Nowadays only supported on OpenVMS V5.5-2 on VAX ( V5.0F ECO 2 )

PATHWORKS / Advanced Server V6.x 1997

> Now based on Windows NT 3.51 code (LAN Manager 3.0)
> Based on code from AT&T most Windows NT 3.51 functionality became available on OpenVMS.
>
> No remote boot, no Disk services
> Old technologies were discarded, Windows became too big for remote boot, giving network load issues and container files were no longer used as file services became faster and more common.

Advanced Server V7.2 1998

> Introduction of the OpenVMS registry
> Functional this release was the same as Pathworks V6.0. The only differences were the replacement of an ini file by the registry and the introduction of ODS5 disk structures with the use of extended character set and lowercase filenames.

Advanced Server V7.3 2001

> Member server function
> As it lacked from the AT&T code Member server functionality was added by our own labs to enable us to participate in Windows 2000 Active Directory domains. This functionality is also available for VAX in Pathworks version 6.1

Advanced Server V7.3A (ECO-2) 2002

> Windows NT 4.0 based server
> Lots of bug fixes and performance improvements over version 7.3.

This document is based on the functionality of the latest version: V7.3A patch level ECO2.

This version is a complete rewrite of large parts of the product to make the transition from Windows 3.51 to Windows 4.0 code base. The functional changes that have been made in Service pack 4 of NT 4.0 are all implemented in this release. This makes the product interact and function well within Windows 2000 and Windows 2003 Active Directory domains.

## Domain Functionality

In a Windows NT domain there are three different roles that a server can perform within the domain.

1. **Primary Domain Controller (PDC)**
   This is the domain master in maintaining the security identifiers in the domain. All users and groups are given a security identifier by the Primary DC to be used for assigning access throughout the domain. The Primary DC distributes all changes in the domain to the Backup DC's. Without the Primary DC, no change in the domain, like a user's password, is possible.
2. **Backup Domain Controller (BDC)**
   A Backup DC has two main functions. First it aids the Primary DC with logon validation. All users that logon to the domain are checked, validated, by a Domain Controller, PDC or BDC. This is a form of load balancing. A second function of a Backup DC is to maintain domain consistency and availability. Should your Primary DC disappear in smoke, you still have your domain based on the domain copy that is held by all Backup DC's. In such a disastrous event you can promote a Backup DC to Primary DC in order to allow changes to the domain.

   Note: Running a domain without a Backup DC is like living dangerously, it's not the question IF you will heart yourself but when. Since all Windows machines change their password autonomously, you can never be sure that a restore from backup will bring back your domain completely. Some PC's or servers may have to be brought back into the domain manually because they changed their password after the last backup. Also trusts to other domains may fail the same way.
3. **Member Server**
   A Member Server is only file and/or print server, it does not maintain a copy of the domain database and requires the availability of a Domain Controller for validation to allow users access to its shares.

Advanced Server for OpenVMS V7.3A can perform each of these three roles.

Each of these roles has its specific usage and the list below is what we see most in the field (numbering as above).

1. Advanced Server as Primary DC is most often used in environments where system management is split up and where OpenVMS and Advanced Server are managed separately from the Windows environment. In these cases a trust is needed between the Advanced Server world and the Windows world to make management easy and security common. See the next section for more information about trusts.
2. As in any domain, a Windows NT or Advanced Server domain needs a Backup DC to secure the validity of the domain user database (SAM). This BDC can be any out-of-the-box PC running Windows NT or another OpenVMS or UNIX machine running Advanced Server.

Advanced Server can also be a BDC in a Windows 2000 domain as long as this domain is running in mixed mode.

3. The member role for Advanced Server is most often used in conjunction with a Windows 2000 or Windows 2003 domain. If such a domain is running in Active Directory integrated mode, this is the only role which Advanced Server can perform in such a domain.

As we briefly touched inter-domain relations above, I will now go into more details on relations between domains.

### Inter-Domain Relations, Trusts

With many customers, there are multiple domains on the network. This can be due to various departments managing their own domains, a geographical separation or just a boundary between production and test environments.

Often it is needed to give users from one domain access to resources in another domain. In the past this had to be done by creating a username in both domains for such a user and give him the task to keep the passwords in sync. With Windows NT, and Pathworks version 6, the option was introduced to have a username in only one domain and use it throughout the network. To do this, the other domains have to trust your login domain. Once a trust is in place, you can take a username or a global group from the trusted domain and use it in the security masks of trusting domains. Please note that a domain local group cannot be used beyond the own domain. A domain local group can also not be used on a member server because a member server has its own local groups.

Like any Windows domain, a domain with Advanced Server in it can trust any other domain or can be trusted. There is a configurable maximum to the number of domains that can be trusted. If you need to configure your domain for more then 31 trusts, increase the following registry setting;

```
Key: SYSTEM\CurrentControlSet\Services\AdvancedServer\ProcessParameters
NumClient_Session REG_DWORD 5 – 128.
Default value is 32.
```

Description:

Limits the number of trust relationships that a server can maintain with other domains.

This value should be at least one greater than the number of domains trusted by the servers' domain.

You can best set such a registry value by the REGUTL command on OpenVMS to prevent typos, since REGUTL knows which registry keys are known to the server.

For example:

```
$ REGUTL SET VALUE * NUMCLIENT_SESSION 44
```

There is a server configuration element to take into account when using trusts.

Trusts use up sessions, so you will have to increase the number of PC clients that you configure your server for if you have a large number of trusts.

For details about configuring your server, see the section entitled: Monitoring and Performance.

## Protocols

For PCs and servers to communicate with each other you need a common language or protocol.

There are several protocols available for machines to communicate, including TCP/IP, AppleTalk, IPX, DECnet and many more.

Advanced Server can be configured to use one or more three protocols:

- DECnet
- NETBEUI
- TCP/IP

Because the TCP/IP protocol is used more and more these days, we normally see systems that are configured to use TCP/IP only. Some details about this protocol; TCP/IP is built around the usage of network numbers or subnets. This feature is mostly used for the purpose of routing the protocol between several sites of a company but can also be used to divide the local network into manageable parts. Apart from the necessary segmentation of your network, the principle of subnets can give you some headaches. Windows NT connectivity is partly based on the principle of browsing and this functionality is not routable between subnets or sites. There are many OpenVMS systems using multiple TCP/IP subnets on multiple network cards. This could bring a challenge for Advanced Server, especially when this happens in a cluster. For Advanced Server it is not supported to have individual cluster nodes in different subnets only. They must have, at least, one subnet in common on all nodes of the cluster. This has to do with the way Microsoft designed browsing.

Since the NETBEUI protocol and some functionality of the other protocols, like browsing are not routable you may have to configure Advanced Server to use a specific interface on a system that has multiple interfaces.

This can be done by defining a logical name per protocol that you use.

| | | |
|---|---|---|
| For DECnet: | `$ DEFINE /SYSTEM/EXEC NETBIOS$DEVICE` | `FWA0:` |
| For NETBEUI: | `$ DEFINE /SYSTEM/EXEC PWRK$NBDAEMON_DEVICE` | `FWA0:` |
| For TCP/IP: | `$ DEFINE /SYSTEM/EXEC PWRK$KNBDAEMON_DEVICE` | `FWA0:` |

When you define the above logical name for TCP/IP you also have to specify the IP address for this device with the following logical name:

```
$ DEFINE /SYSTEM/EXEC PWRK$KNBDAEMON_IPADDR "16.198.227.15"
```

You can also use this method if you have an interface that is not yet recognized by Advanced Server.

Should you be unable to have at least one subnet in common on your cluster nodes, you will encounter problems with the browser service like a permanent state of “Start Pending” on some nodes in the cluster. In such a case you will have to disable the Browser service. Do this by setting a registry key:

```
$ REGUTL SET PARAM * MAINTAINSERVERLIST NO /CREATE
```

After a restart of Advanced Server you will no longer see the PWRK$LMBROWSER process and you will get a report of this in the system event log, where it reports you the message numbers 7024 and 2550, indicating that the browser service had nothing to do and exited at startup.

So far for protocols, now I come to the way a machine can find another one on the net.

## Name Resolution

There are various ways to file a combination of name and address for a machine. You don’t want to define every machine on every PC, this can largely be automated.

Depending on the version of operating system there is a variety of options that are used in different sequences to find an address for a machine on the network.

As in Windows NT, name resolution for Advanced Server is based on a combination of broadcasting, WINS and DNS. WINS, (Windows Internet Naming System) is for Windows NT, and thus for Advanced Server, the preferred way to find an address for a machine. This is also the first option to try on the network, after checking the local memory cache. If WINS doesn't provide an address for the machine you are looking for, Advanced Server will use broadcasting to get the address. Broadcasting, however, will only travel as far as the TCP/IP subnet reaches. This makes broadcasting very limited and not so useful. Broadcasting relies on the machine itself to answer the call so it will only be effective if the machine is in your own subnet. As a last resort, DNS can be used, in order to find an address for a machine name.

I did not mention a domain name in the sequence above. This is because for searching a domain name, DNS will NOT be used. In the search for a domain name or a domain controller, Advanced Server uses WINS and broadcasting only, Just like Windows NT. This makes it important to keep WINS if you are transferring your domain to a Windows Active Directory domain, based on DNS.

The above mentioned local name cache can be preloaded by the use of a file called: PWRK$LANMAN:LMHOSTS. (without extension). This file has the same layout as in Windows;

```
<TCP-IP Address>  <hostname>  [#PRE [#DOM:Domain-name]]
```

For example:

```
16.11.231.18      TESTPDC            #PRE  #DOM:TESTDOM
```

This line loads the name TESTPDC in cache with the address 16.11.231.18.

This line also loads the domain name TESTDOM in cache and associates it with the same address.

This way your machine knows that it has to go to 16.11.231.18 if it needs to contact a domain controller for the domain TESTDOM.

You can configure the use of WINS, DNS and the use of an LMHOSTS file using the configuration utility: `$ADMIN /CONFIG`, in the TRANSPORTS section. By default none of them is enabled and your system relies on broadcasting only, something you don't want to do in a modern domain.

New in version 7.3A-ECO2 is that you can edit and reload the LMHOSTS file without restarting the server. To do this, you can use the command `$ NBSHOW KNBCACHE RELOAD`. The command `$ NBSHOW KNBCACHE`, without the reload option, can be used to display the current content of the name cache.

## Troubleshooting name resolution

Troubleshooting name resolution can best be started using the following command:

```
$ NBSHOW KNBSTATUS <NAME>
```

You should expect to receive a list of claimed names from the server or client requested, like the output from the Windows following command:

```
$ NBTSTAT -a <NAME>
```

Following is sample output (partial) of `$ NBSHOW KNBSTATUS` command with explanations of the various clamed names:

```
Pwop01-System > nbshow knbstatus pwop08

..............

Local name table (11 names):

Name            Soc Num Status                explanation
```

```
PWOP08          x20   1 Unique Registered  <- Server name, server service
PWOP08          x00   2 Unique Registered  <- Server name, workstation service
PWDOM2          x00   3 Group  Registered  <- Domain name
PWDOM2          x1c   4 Group  Registered  <- Domain Controller in PWDOM2
PWDOM2          x1e   5 Group  Registered  <- Used for Browser elections
PWOP08#D        x00   6 Unique Registered  <- Internal usage
PWOP08#B        x00   7 Unique Registered  <- Internal usage
PWDOM2          x1d   8 Unique Registered  <- Master Browser
^^__MSBROWSE__^ x01   9 Group  Registered  <- Master Browser
PWDOM2          x1b  10 Unique Registered  <- Domain Master Browser (PDC)
PWOP08_65       x00  11 Unique Registered  <- Internal usage
```

Important in this output are the server name and the domain name. They are both listed multiple times in this output and here you can see the functions this server is performing. You can also use this method to search for the PDC or PDC emulator (Windows 2000) of a certain domain. If you encounter difficulties in joining a domain during the configuration of Advanced Server you can use the command `$ NBSHOW KNBSTATUS` to see if you can resolve the domain name to find the PDC. Since version 7.3A, the command `$ NBSHOW KNBSTATUS` <NAME> accepts a third parameter where you can specify the last byte of the NetBIOS name (listed under the column "soc" in the table above). For example, if I issue the command `$ NBSHOW KNBSTATUS PWDOM2 1B`, I will get the same output as above while searching for the PDC in domain PWDOM2.

You can use this if you encounter problems while joining a domain during initial configuration. You can do this because the PWRK$KNBDAEMON process that interfaces to TCP/IP is not shutdown when you leave the configuration procedure.

So far for communication between machines, let's take a closer look at configurations.

## Clusters

### The cluster is the server

This means that there must be one single entity that is seen as the cluster from the outside world.

When you use an OpenVMS cluster, you will have to give this cluster a cluster alias name. You can configure many nodes in a cluster to run Advanced Server. They will all listen to the same cluster alias name.

There is, however, no need to configure all of the cluster nodes for the use of Advanced Server.

### File access in a cluster

It is important to mount the disks that you want to share out to the Windows world on all nodes that you configure to run Advanced Server. Failing to do so will make a file unavailable through some nodes and this will look inconsistent from a user's perspective. Unlike a Windows cluster, you will be able to access all disks through every node in the cluster and at the same time. This feature, unique to OpenVMS and Tru-64 UNIX clusters, does provide some overhead in the server which may cost some performance.

Should one of the nodes in an Advanced Server cluster leave the cluster, the clients connected to it can resume work through one of the other, remaining, nodes without user intervention. To use this feature, clients must use the cluster alias to connect to the cluster and not a specific node name.

Advanced Server has a set of databases to store the reference to shares, users and parts of the security.

In an Advanced Server cluster there is only one set of these database files.

These files must reside on a common disk that is mounted on all nodes that run Advanced Server. This disk must preferably not be local inside one of the nodes but in a separate storage enclosure to prevent unavailability if one node fails.

### The cluster in a domain

As I told you above, the cluster can and should be addressed by its cluster alias name. This is one name for one or more machines. In the Domain, only the cluster alias name must be registered as a server or domain controller, depending on its role in the domain. It is not allowed to create a computer account in the domain for individual nodes of the cluster. This and the fact that there is one set of databases, mentioned above, means that the whole cluster performs the same domain role. If you make your cluster the PDC, this will lead to the situation that you will see multiple PDC's when browsing the domain. This is intended behavior since the browser gathers information on both the cluster alias as well as the individual nodes. If you look at domain members only, you will just see the cluster alias, not the individual node names; they only exist in Browser information.

In the OpenVMS administrators utility this looks as follows:

```
$ ADMIN SHOW COMPUTERS

Computers in domain "PWOP":

Computer       Type                          Description
-------------  ----------------------------  ----------------------------
[PD] PWOP01    OpenVMS (NT 4.0) Primary      Advanced Server V7.3A for OpenVMS
[PD] PWOP02    OpenVMS (NT 4.0) Primary      Advanced Server V7.3A for OpenVMS
[PD] PWOPCLU   OpenVMS (NT 4.0) Primary      Advanced Server V7.3A for OpenVMS
(Alias)
 Total of 3 computers


$ ADMIN SHOW COMPUTERS/TYPE=DOMAIN

Computers in domain "PWOP":

Computer        Type                         Description
--------------  ---------------------------  ----------------------------
[PD] PWOPCLU    Windows NT Primary
 Total of 1 computer
```

Note that when you look at the domain members only, the server comment (description) is omitted since this is browser information and the browser information is not used with this command.

## Dos and Don'ts

This section provides recommendations for Advanced Server and things that you must not do with Advanced Server.

DO:

- A PC client will have more then one session to a server, most likely two sessions per PC. This means that you have to configure our server for a little more then twice the number of PC clients that you expect to use the server.
- You can configure any node in the cluster for as many clients as you expect to use it but make sure to have enough room to accept clients that failover from a node that exits the

cluster. For example, if you have a two-node cluster and 400 clients, configure both nodes for at least 850 sessions.

- All cluster nodes must be in the same TCP/IP subnet.
- Configure the server to use WINS for name resolution.
- Add a multihomed entry for the cluster alias in the WINS server database, even if your cluster has only one node running Advanced Server.

DON'T:

- Don't configure any server for less then 40 PC clients. It will also use client sessions for inter domain sessions, trusted domains and browsing.
- All nodes in the cluster MUST speak the same protocols. Don't let one speak TCPIP only and the other TCP/IP and DECnet.
- Don't configure your server to speak DECnet if your PDC doesn't. The Primary Domain Controller must speak all the protocols that are in use throughout the domain. See below for details.
- Don't build a domain with only a PDC. Always add at least one BDC to the domain.
- Don't create a computer account in the domain for the individual cluster nodes, just for the cluster alias.

A little explanation on the protocols of your PDC, mentioned above:

Suppose your PDC is a Windows machine that is configured to run TCP/IP only and you want some old PC's to connect to Advanced Server through DECnet.

In this scenario your Advanced Server for OpenVMS could be the only server that speaks DECnet. If that is the case, it will become Domain Master Browser on the DECnet protocol since there is no higher machine that claims this role. The Browser service is not built to have a different role on different protocols so this will bring your PDC in trouble because the PDC should be the Domain Master Browser.

This situation will create an error condition on TCP/IP which is hard to find since the cause is not on TCP/IP but on the DECnet protocol.

You can avoid this by installing DECnet on the PDC. DECnet is supported on all current Windows versions and can be installed from the Pathworks 32 CD kit.

## Licensing

Any connection to Advanced Server must be licensed.

Nowadays there is only one type of license that you need to install in order to get access to shares on Advanced Server. This is the Client Access license, named PWLMXXXCA07.03.

There are two principal ways to use this license.

1. Server-based licensing

   If you have only a few servers you can buy a set of licenses for each node and just load the license into LMF. In this setup, you configure Advanced Server to NOT use the License Server.

   If you do this, Advanced Server will subtract a license for each PC that makes a connection to the server. There is nothing to manage with this license setup, just make sure that you have enough licenses on each server.
   You can check the license usage with the command `$ PWLIC`. Please keep in mind that

license usage in a cluster is always cluster-wide, so PWLIC will show you the combined license usage for the whole cluster.

One more remark on licenses in a cluster: The license processes on the cluster nodes talk to each other to give this combined license usage. You **must** setup your cluster to use **one** license database for the whole cluster. If you don't do this and use multiple license databases (e.g. per node), Advanced Server nodes will negotiate about the total number of licenses for the cluster and the node with the smallest number of licenses will overrule the others. To have multiple license databases in a cluster is officially unsupported.

This is the easiest and preferred setup and is called server-based licensing.

2. Client-based licensing

   If you have many servers or when PCs make use of multiple servers, it may be cheaper to setup one or more servers as a license server.

   To do this, like with option 1, load your licenses into LMF on one node in your network and configure Advanced Server on this node to run WITH License Server. Now the License Server will allocate all these licenses and will give them to PCs that request a license.

   To make a PC client request a license, you need to install license software on each PC client that needs to connect shares on Advanced Server. This license software is available on the Pathworks 32 CD kit and on the share pwlicense on each Advanced Server node.

   With the license software installed, the PC client will present its license to any server it needs to make a connection to. The server that a PC client with a loaded license connects to does not need to have any license installed.

   This license will remain with the PC client that requested it until it is released manually.

   The license server will maintain a database for all the licenses it has given out.

   You can create reports on the usage of licenses through the license utility:
   ```
   $ ADMIN /LICENSE
   ```

   As mentioned above, you need manual intervention to release a license from the database if a PC client leaves the network.

   This is called Client-based licensing and is more labor intensive.

3. You can also create a mix of options 1 and 2.

   If you run your server with the License Server, you can move a number of licenses to a special group called "Server-based". By this mean you can make a new PC client connect to this server to install the client license software from a share on this server.

   You can also install the client license software from the Pathworks 32 CD kit that comes with your OpenVMS distribution.

License shortage will be reported in the general event log that you can examine with the following command:

```
$ ADMIN /ANALYZE [/SINCE[=<DATE-TIME]]
```

## Monitoring and Performance

Basic monitoring of Advanced Server is pretty easy and can be done at three places.

1. There is a general error log about OpenVMS related issues and licensing.
   This file is: `PWRK$ROOT:[000000]EVTLOG.DAT`.

   You can access this error log with the following command:

   ```
   $ ADMIN /ANALYZE [/SINCE[=<TIME>]]
   ```

   There is no need for the server to be running when you issue this command.

   All you should see here is a few startup messages and maybe an autoshare message at startup time when you have disks with a very long label, like a mounted CD.

   It's a good practice to cleanup the error log at a regular basis by the command:

```
$ ADMIN /ANALYZE /PURGE [/BEFORE=<TIME>]
```

2. As a second source of information, there is the Windows NT compatible event log.
   These files are: `PWRK$LMLOGS:SYSEVENT.EVT`, `APPEVENT.EVT` and `SECEVENT.EVT`.

   You can access these event logs with the following command:

   ```
   $ ADMIN SHOW EVENT [/FULL] [/TYPE=SECURITY | SYSTEM |
   APPLICATION]
   ```

   The system event log is default when `/TYPE` is omitted.

   You can also examine these events through the event viewer application on your Windows platform.

   It's also good practice to cleanup this event log at a regular basis by the command:

```
$ ADMIN CLEAR EVENTS [[/TYPE=SECURITY | SYSTEM | APPLICATION]
```

   Alternatively, you can use the Windows event viewer.

   You should make sure that his event log is setup to overwrite older events as needed or you will loose messages. This setting can only be changed from the Windows platform.

3. The individual processes that makeup the server produce log files.

   There are two directories where you can find these log files. These are: `PWRK$LMLOGS` for those processes whose name starts with `PWRK$LM`, like `PWRK$LMSRV`, the other location is `PWRK$LOGS` for processes whose name starts with `PWRK$,` without the LM, like `PWRK$KNBDAEMON`.

   The most important of these log files is the actual file server log file,
   `PWRK$LMLOGS:PWRK$LMSRV_<NODE NAME>.LOG`

   These filenames always contain your machine name to distinguish between nodes in a cluster.

   A simple `TYPE /TAIL command` will show you the end of the file to take a quick look at the current status.

We consider it good management if you take a regular look at these three places for errors.

More monitoring options will be discussed in the next section.

## Performance and Tuning

As with any product, you have to know what to expect if you want to do some tuning.

This means that you will have to know what the normal workload is for your system and for Advanced Server, a baseline. In other words, if you take your first look at performance when your users start complaining about the performance you don't know what you are looking at.

Advanced Server performance breaks down into OpenVMS performance and Advanced Server itself.

Advanced Server performance should be about equal to a Windows NT server.

Only when many small files are involved, like in a user profile, we are a bit slower.

BUT, as often done, you should not compare last century's Alpha server 2100 with a Proliant server that you bought recently, even if they are about the same size.

My intention with this chapter is to give you a few handles to do basic tuning to your server, not to give a detailed, five day training in a nutshell.

What aspects are there in tuning?

There are CPU, memory, disks, the transport protocol and the network itself.

Then there is OpenVMS and the Advanced Server product.

Let's start with Advanced Server itself.

## Configuration

When you have installed Advanced Server, you have to configure it. In the first part of the configuration you will go through a menu that you can also start at a later stage with the command `$ ADMIN /CONFIGURE.`

The first screen of this menu contains important settings for your server.

**Example**

```
+------- Advanced Server Configuration for node PWOP01 -------+
| Options  Help                                                |
|                                                              |
| +Server's Client Capacity--------------------------------+ |
| |                                                          | |
| | ( ) Maximize Client Capacity Using AUTOGEN/Reboot        | |
| | ( ) Maximize Client Capacity Without AUTOGEN or Reboot | |
| | (*) User Supplied Client Capacity                        | |
| |                                                          | |
| |  Client Capacity: 50                                     | |
| +----------------------------------------------------------+ |
|                                                              |
|   Percent of Physical Memory Used: 80                        |
|   Data Cache Size (Kbytes): 131072                            |
|                                                              |
|   Maximum Concurrent Signons: 10                             |
|   OpenVMS Process Priority: 9                                |
|                                                              |
|  +--------+   +------+   +-------------+   +---------------+ |
|  | Verify |   | Quit |   | Advanced... |   | Transports... | |
|  +--------+   +------+   +-------------+   +---------------+ |
|                                                              |
       + Test for supportable configuration ————————————————————————+
```

Some remarks about this first page, in order of importance:

- Never use a client capacity below 40, your server normally needs sessions to other servers and domains. And each PC client creates, most often, two sessions to your server. So, at average, you setup a server for 2 times the number of expected PCs plus some extra (at least 20 extra).
- Choose a proper value for your data cache size. Shown here is the upper limit which could be an overkill for a 15 user system like this one (Client Capacity: 50). 32000 to 64000 is often adequate. The default value of 8192Kb can be considered a small cache size but could do for a small system in a small domain. The Advanced Server data cache only caches open files! I'll show you in more detail how to examine the data cache, later in this article.
- Percent of Physical memory used: 80. This is the default and this number is only used to make a calculation for the system parameter WSMAX (process memory usage) and to see if your system can support the number of clients that you specify. It is good practice to reduce this number only if there is also a lot of interactive usage on your system. Please keep in mind that your server process may be representing a 1000 user workload or more and in such a case, the server will need quite an amount of resources.

There are two other sections in this menu:

- The Advanced section of the menu is normally not changed.

  Go here if you need to change permission settings to include OpenVMS rights like the usage of resource identifiers in your security masks.

  Or, to turn off the Open File Cache. This is a file header cache only, not the file data and could be in your way if you have an OpenVMS job that needs to process a file as soon as it is put on the server. If the OpenVMS job can wait the five seconds of the Open File Cache timeout, then do that and leave the Open File Cache enabled. The Open File Cache is a major win for batch jobs and applications like Word that open and close a file 10 or more times before they really start to read it.
- The Transports section of the menu.

  Normally you do visit this part of the menu to turn off DECnet and enable TCP/IP. You must choose a way of name resolution like WINS or DNS. Remember that Advanced Server uses WINS as the primary way for name resolution, so if you need to access any machine outside your TCP/IP subnet, you must have a WINS server and enable WINS.

  We do advise you, these days, to enable the usage of the LMHOSTS file, even if it is not there, since in this version you can reload it dynamically. This can be very handy should you be confronted with sudden changes in your network. The way to reload the LMHOSTS file is through the following command:

```
$ NBSHOW KNBCACHE RELOAD
```

## Configuration When the Server Is Running

You can use the `$ ADMIN /CONFIGURE` command at any time during normal system operation; it will not interrupt the system or the server. One good moment to use this command is when you anticipate growth of your PC community. You can use it to see if your sever needs parameter maintenance in order to accept more pc client connections. Increase the client capacity and do "`verify`" to see what message it will produce. You can always bail out of this action by not accepting the change.

My advice: don't let the menu run AUTOGEN for you because it will never use the feedback option.

## Gathering Data

As I wrote before, when you want to do tuning, you need to know what the system's normal behavior is.

With the command `$ PWMON` you can get a good overview of server activity.

**Notes**: In the older versions of Advanced Server you have to set your screen to a width of 132 columns to use the PWMON command.

All PWMON commands in version 7.3A-eco2 will work on a screen that is set to a width of 80 columns but several commands will give you more information if you set your screen to 132 columns.

I'll give a few examples of how to use PWMON.

Let's start with the data cache, as promised.

I'll show you the most important part of the output to keep things limited.

```
$ PWMON DATA_CACHE

           Advanced Server Data Cache/Process=PWRK$LMSRV

Cache buffer size.           8192 Cache buffer count           8192
Cache buffers free           7808 Cache buffers used            384
Nr of assigned handles         22 Cache user limit                0

Data lookup rate          5423213 Cache misses                    0
Cache hits                4298693 Cache hitrate (%)              79
```

In the above screen, first line, you see a count of 8192 buffers, all 8192 bytes in size. This system is configured for 65536 Kb data cache size with `$ ADMIN /CONFIG`. (See for a reference the screen shot of the menu two pages back where you see "`Data Cache Size (Kbytes): 131072`").

On the second line you see there are 7808 buffers free which means there is plenty of free space.

On the bottom line you see a cache hit rate of 79%. Depending on your system usage, this can be good or bad.

In this case the usage of the system consist for 90% out of large files that are opened and closed regularly and both read and written to, so the hit rate number is good. Remember only open files are cached, so if your application keeps the file open, cache results will improve.

Another Scenario
Your system is generally OK but sometimes very slow.

In this scenario it is likely that someone is hammering your server with some batch job, you can find out quite simple who that is by the command: `$ PWMON CLIENT`

If you issue this command without parameters or switches you will just get a list of connected PCs and how many commands they issues to your server. See below:

```
                         Advanced Server Clients

Name                      Tot. SMB's  Tot. API's  Tot. RPC's  Tot. Alrts
CLIENT_LM_BARK01                  31           0           0           0
CLIENT_LM_HOUBENE02             1015           0           0           0
CLIENT_LM_HHOSANG02             1468           0           0           0
CLIENT_LM_VANGEESTR03         113765           0           0           0
CLIENT_LM_UTOCWARN1            14447           0           0          19
..................
```

This output has little value if the number of PC's connected to your server is more then will fit on one screen. You will scroll from page to page without getting the real overview.

It will be much handier to make use of the new qualifiers available in this version. There is `/OPERATIONS`, `/RESOURCES` and `/TIMES` but also `/TOP=(OPERATIONS | RESOURCES | TIMES)` to sort on the usage by the PC clients.

PWMON client is one of the PWMON commands where you will get more columns at a screen width of 132.

```
$ PWMON CLIENTS /TOP=OPERATIONS
```

```
                                 Advanced Server Clients
                                    by top operations
Name                   File Opens  File Locks  Dir Lookup  Transact's  Latest SMB
Latest SMB   Latest API
CLIENT_LM_VANVELZEN01          26          12           0         199           5
CLIENT_LM_UTOJSCH01             2           0           6          53         113
CLIENT_LM_HOUBENE02            43          40          15         652         113
CLIENT_LM_HHOSANG02            46          26          11         556          50
```

I omitted the last column in the display to make it fit on the page; this is showing the latest API call.

This screen will give you an overview of the top working PCs. Please pay special attention to the column "Dir Lookup" as this is known to generate a large load on the server and could point you to a badly designed program, running on a client.

Should the badly designed application be unavoidable, your next step is to take a look at the disks; how is the server dealing with disk caches?

## Disk Cache Statistics

```
$ PWMON ODS2
```

This will show you disk related caches in Advanced Server.

The important ones are File ID cache, Directory cache and the Path cache. All of these caches should have a hit rate of around 90% or more.

```
                  Advanced Server ODS2 Cache/Process=PWRK$LMSRV


Nr of PATH cache lookups        1246127 Nr of PATH cache hits          1222885
Nr of PATH cache misses           23242 PATH cache HIT RATE (%)             98
Nr of PATH cache invalidates          0 Nr of PATH cache searches            0

Nr of DIR  cache lookups        2078357 Nr of DIR  cache hits          1962628
Nr of DIR  cache misses          115729 DIR  cache HIT RATE (%)             94
Nr of DIR  cache invalidates          1 Nr of DIR  cache refreshes         614
Nr of DIR  cache thrashes          3553 Nr of DIR  cache CTXT's free      256
Nr of DIR  cache entries free         3 Nr of DIR  cache blocks free     1391
Nr of DIR  cache buf's total          4 Nr of DIR  cache buf's in use       0

Nr of FID  cache lookups        5842722 Nr of FID  cache hits         5452852
Nr of FID  cache misses          389870 FID  cache HIT RATE (%)            93
Nr of FID  cache invalidates      35963 Nr of FID  cache searches    13334112
Nr of FID  cache thrashes             0        FID  cache size        1152519
Nr of FID  cache entries           2250

Nr of DAT  cache lookups          28588 Nr of DAT  cache hits           26400
Nr of DAT  cache misses            2188 DAT  cache HIT RATE (%)            92
```

Most important in this page is the FID cache. As all Microsoft products want to know information from the file header like date and size, this cache is used a lot.

To influence the FID cache you can run the configuration utility `$ ADMIN /CONFIG`, go to the advanced menu and increase the number of open files per client. The number of FID cache entries can grow up to 16384.

A restart of the server is needed to effectuate this change.

If you have a system that is configured for a small number of clients, the above modification will not lead to much of a change. In this case you can better create a logical name as follows:

```
$ DEFINE /SYSTEM /EXEC PWRK$TEMP_ODS2_FID_CACHE_SIZE 16384
```

And then restart the server.

Defining this logical is, of course, not a one time event. You must put this in the system startup procedure.

To check the value that is used by the server you can check the file server log file;

```
$ SEARCH PWRK$LMLOGS:PWRK$LMSRV_UTRACK.LOG FID_CACHE

21-APR-2004 13:09:24.45 20400458:007B3948         ODS2_FID_CACHE_SIZE:   2250
```

This shows that our server is using the value of 2250 for the file ID cache.

The same mechanism can be used for the other ODS2 caches.

**Note**: Everywhere where ODS2 is mentioned, this is also used for ODS5 disks.

## CPU Usage

I've recently come across a customer who complained about a very high CPU usage.

After a lot of testing I ran the command `$ PWMON CLIENT /TOP=OPERATIONS` as described above.

I discovered that a few PCs had high and rapid increasing numbers in the transactions column.

Transactions are commands that are not file related like domain queries or device queries.

To find out what such a PC is doing you have to take a look at the network.

You can take a snapshot of the traffic between a PC and the server using the command `$ TCPTRACE`.

TCPTRACE is a part of HP TCP/IP for OpenVMS and is not available if you use TCPware.

The syntax is:

```
$ TCPTRACE /PACKET=10000 /OUTPUT=FILE.TXT <IP-ADDRESS-OF-PC>
```

When you omit the TCP/IP address of the PC you will capture all traffic to and from the server.

When looking at the output of `$ TCPTRACE`, I found that the PCs were constantly looking for printer information while they had no need for a printer on this server. I de-installed this printer on those clients and the amount of CPU usage decreased by about 5% per PC.

The easiest way to take a look at the output of TCPTRACE is through the use of the product ETHEREAL which you can download for free from http://www.ethereal.com. But in this case all the traffic to this PC contained the comment of the printer share, a type of the output file was sufficient. This example shows that you don't always need in depth knowledge of network protocols to see what is happening. Just don't be scared to take a look.

Another known CPU intensive item is large directories. If you have directories that contain thousands of files then you will certainly have a high CPU usage if they are accessed frequently. The simple reason is the fact that Windows requests information from the file header of each file. The only thing you can do against this is to reconsider the layout of your directory tree.

## Troubleshooting Tips

When something "undefined" goes wrong, where do you start searching?

Say, a user calls with the report that your server is unresponsive.

I'll give you a few hints and tips where to look.

To start, make sure that you defined the specific commands for Advanced Server in your LOGIN.COM by:

```
@SYS$STARTUP:PWRK$DEFINE_COMMANDS.COM
```

I'll start with the command; `$ PWSHOW [CLUSTER]`

This will give you an overview of all Advanced Server related processes, per node or cluster wide, if you specify the parameter "cluster". You have to know what processes your server normally runs to discover if one is missing. Normally you should see between 6 and 10 processes.

The minimal process list is:

```
20400456 PWRK$ADMIN_0    LEF      6       56   0 00:00:00.01       153    106
2040044C PWRK$KNBDAEMON  HIB     12     1812   0 00:00:04.93       341    393
2040044E PWRK$LICENSE_R  HIB     11      244   0 00:00:04.31       538    479
20400454 PWRK$LMMCP      HIB     11     2058   0 00:00:04.85      1247    505
20400458 PWRK$LMSRV      HIB     11    15723   0 00:00:09.57      3008   2234
20400452 PWRK$MASTER     HIB      6      186   0 00:00:04.04       460    175
```

Should one or more processes be missing, issue the following command:

```
$ ADMIN /ANALYZE [/SINCE[=DATE-TIME]]
```

This will give you an error message and the name of the failing process, if any.

You may also find license related errors here.

### Example 1

`$ PWSHOW` tells you that all the PWRK$LM processes are missing and `$ ADMIN /ANAL` tells you that the process PWRK$LMSRV was the one that exited first.

Then the next step would be to take a look at the process log file:

```
$ TYPE PWRK$LMLOGS:PWRK$LMSRV_<NODENAME>.LOG
```

Here you can find the signature of the real problem. Please report this to your support centre to get a solution.

**Example 2**

Let's look at another scenario, same report from a user; server is unresponsive.

`$ PWSHOW` tells you that all processes are there. So there is no failure, its just slow.

The next step would be to take a better look at the output of `$ PWSHOW`, and maybe repeat the command.

Look at the process PWRK$LMSRV, this is the process that is handling all File and Print Server traffic.

Is the PWRK$LMSRV process in HIB state, in LEF state or in RWSCS state?

1. If the state is HIB(ernating), take a look at `$ PWMON CLIENT /TOP=OPERATIONS`.
   It is likely that you will find a PC client here who is heavily communicating with the server. You should check what the PC is doing. For example, it could have turned on virus scanning of network drives.
2. If the state of the PWRK$LMSRV process is LEF, you may have a problem with one of your disks. This state means that it is waiting for an I/O completion.
   Start looking at `$ SHOW DEVICE D`. This may indicate, for example, a device in trouble like mount verification. Another approach to disk related issues could be `$ ANAL /SYSTEM`. Then, within `SDA>`, do a "`SHOW PROCESS PWRK$LMSRV /CHANNEL`" and look for busy channels to a disk or file.
3. If the state is RWSCS it indicates the process is communicating to another node in the cluster. Very often this is locking, you can take a look at `$MONITOR DLOCK` and `$MONITOR RLOCK`. If `$MONITOR DLOCK` reports high activity and `$MONITOR RLOCK` shows little lock remastering, there is nothing you can do about it, the server is working hard. If `MONITOR RLOCK` shows high lock remastering, you could contact your support center to discuss a change of system parameters to influence lock remastering. What is considered high lock remastering depends completely on your system capabilities, specially the channel between the systems.

**Example 3**

Your user cannot get a connection to his shares but others can still work.

This could have various sources.

**Example 3a**

`$ ADMIN /ANALYZE /SINCE` can give you a lead to a licensing issue. You could see an error like:

```
Event Time:   18-JUN-2003 20:35:45.96       Node:  UTURBO

Process Id:   20A00285

Event:        No server license for client - access denied

Event Source: LAN Manager Server

Event Class:  Warning
```

```
      Client:   SMETSERSDENNI
```

This error report will also show the PC that is struck by the error which you can use to verify the user that is complaining.

In such a case, check with the command: `$ PWLIC` to see if you have sufficient licenses available.

```
UTURBO > PWLIC
Advanced Server for OpenVMS (V7.3-120A): Server-Based License Report:
    License              Total    Cluster Use    Available
    PWLMXXXCA07.03        1750        29          1721
```

Note here, PWLIC displays cluster-wide license usage.

The available count could have reached 0.

If this is the case, your action depends on whether you have the license server running.

If you have, the process PWRK$LICENSE_S is running, then you may consider moving some licenses to the group called server-based.

If you do not use the license server, you will have to buy more licenses or reduce the number of clients that is using the system. Reducing the number of clients cannot be achieved by configuring the server for fewer clients since you cannot tell how many sessions each client will create to your server. You can only achieve this by changing the usage, for example, turn off browsing. Please be careful when you consider turning off browsing. A PDC will need the browser service but a member server will normally not need it. In case of a Backup Domain Controller it completely depends on your network topology.

To turn off browsing use the REGUTL utility to create or set the MAINTAINSERVERLIST key to NO.

**Example 3b**

`$ ADMIN /ANAL /SINCE` does not show any license errors.

You may have configured your server for too few clients. To monitor this you have to check a few things.

First start with `$ ADMIN SHOW SESSIONS` and check the current number of PCs using your system.

This can give a rough indication but does not include inter-domain sessions and browsing sessions.

Next thing to check is the protocol usage: `$ NBSHOW KNBSTATUS`

The output of this command shows the sessions that are in use over TCP/IP ONLY.

There is a similar command for the NETBEUI protocol: `$ NBSHOW NBSTATUS`

Take a look at the line: `Sessions:  In use:    99 of   100;`

This will tell you if you ran out of session slots. If these numbers get close, like in the example above, you will have to reconfigure your server and increase number of clients.
To do this start `$ ADMIN /CONFIG`, you will find the number of clients as "`Client Capacity`" on the first screen.

Please also note that the list of sessions at the end of the output of `NBSHOW (K)NBSTATUS` only lists maximal 112 sessions. The line containing "`Sessions:  In use:`", `mentioned above,` is the one you should check.

You do not have to shutdown your server immediately to do this reconfiguration but you will have to reboot to effectuate it.

## Documentation

Documentation for Advanced Server is available on the internet. Please look at the HP documentation site; http://h71000.www7.hp.com/doc/advserv73.html

## Futures

There will be continuous development of Advanced Server for OpenVMS.

The first ECO release ( V7.3A-ECO3 ), which is planned for July 2004, will have a special cache around the SpoolSS printing interface to improve the performance of NT-style printing.

There is another eco release for version 7.3A planned to add support for OpenVMS Alpha 8.2.

The first major step will be to bring the current product on the new Itanium release of OpenVMS.

Another step that is planned is the addition of Active Directory integration. This will mean that Advanced Server can publish its resources in the Active Directory.

## Ask The Wizard

Should you have just a question about the product, not an error report, you can ask this to one of our wizards on: http://h71000.www7.hp.com/wizard/

# OpenVMS Technical Journal

## Revision and Configuration Management (RCM) for OpenVMS

Pat Moran, HP Services

### Overview

This article describes the HP Services tool, Revision and Configuration Management (RCM), which collects detailed system configuration information from HP systems at customer sites. The data is stored on the RCM server at HP and is used to create configuration, change, comparison, and analysis reports that the customer and HP account team can access through the Electronic Site Management Guide (eSMG). Customers with valid service contracts access their own information using encrypted (https) connections.

RCM is available for OpenVMS VAX and Alpha systems from Version 6.2 through Version 7.3-2, as well as for HP Tru64 UNIX, Windows NT/2000 and HP-UX systems. This paper describes the RCM architecture and focuses on the design of the RCM OpenVMS collector. It describes the main features of the RCM reports, such as recommendations for critical patches as well as detailed information including the following:

- Disk and tape devices on the system
- Installed software and patches
- Firmware revision levels; hardware part numbers and revision levels
- SAN controller details and topology map
- Enterprise Virtual Array (EVA) configuration reports.

By collecting system configuration information on a regular schedule, RCM change reports make it possible for the customer and HP Service personnel to quickly diagnose problems that configuration changes might have introduced.

RCM, which was developed by the HP Mission Critical and Proactive Services group, has been deployed on thousands of OpenVMS systems worldwide and is a key part of the HP Services portfolio.

### RCM Architecture

Figure 1illustrates the overall RCM architecture. Numbers correspond to numbers in the figure.

1. The RCM data collector is installed on a system at a customer site and gathers selected information from the system software.
2. The collected information is transported to the RCM Server at HP using DSNlink, Email or FTP. DSNLink is currently being retired as a transport mechanism; however, future releases of RCM will use the HP Instant Support Enterprise Edition (ISEE) to transport RCM data securely from the customer site to HP.
3. At the RCM server, the parser-loader software processes the collected information, verifying that the information conforms to the necessary standard (system type, revision, format, and so on) before loading it into the Configuration Snapshot Repository (CSR) database. This database is used as the source of information for various RCM reports. The Configuration Reference Database (CRDT), an additional database that is maintained on the RCM server, contains product reference information such as current hardware revisions.

4. The Electronic Site Management Guide (eSMG) serves as a repository for RCM reports, system healthcheck reports, availability statistics, contact information, and any additional site-related documents that the customer wants to store.
5. Customers and HP Technical Account Managers (TAMs) can access eSMG through a web-based user interface to request reports for specific customer systems. These reports can be used to track historical changes or to make comparisons with a reference system.

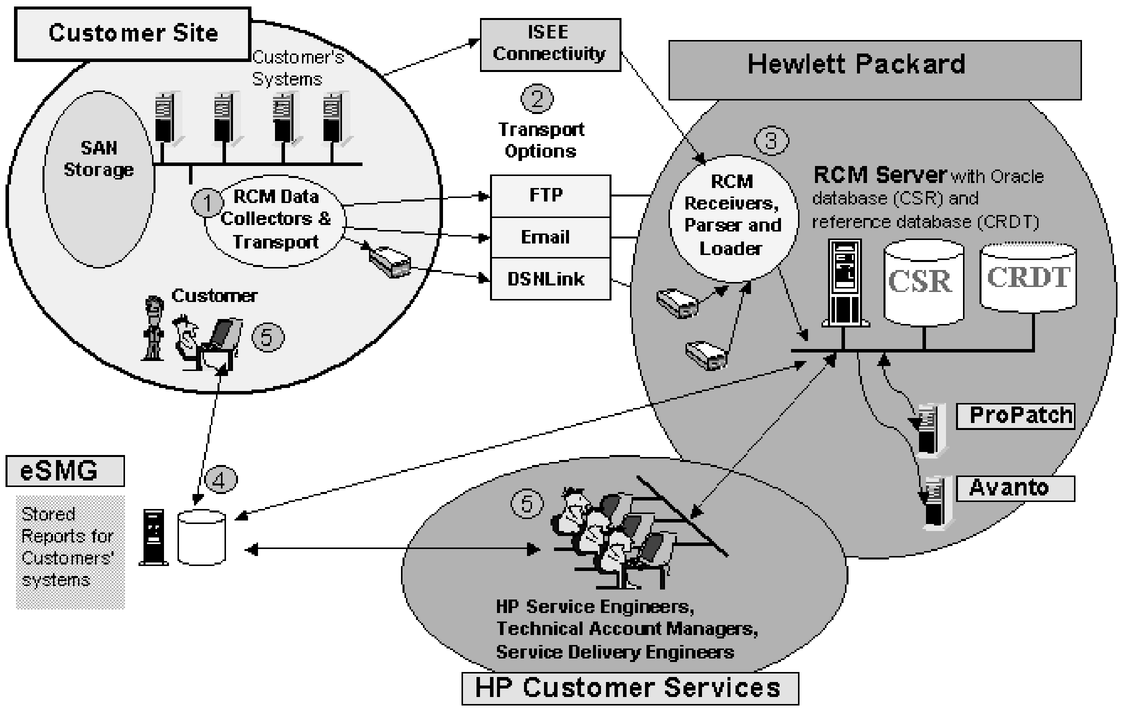

Customer Site
Customer's Systems
SAN Storage
1 RCM Data Collectors & Transport
Customer
5
ISEE Connectivity
2
Transport Options
FTP
Email
DSNLink
Hewlett Packard
3
RCM Receivers, Parser and Loader
RCM Server with Oracle database (CSR) and reference database (CRDT)
CSR
CRDT
ProPatch
Avanto
eSMG
Stored Reports for Customers' systems
4
5
HP Service Engineers, Technical Account Managers, Service Delivery Engineers
HP Customer Services

**Figure 1 RCM Architecture**

## RCM Security

The Electronic Site Management Guide incorporates security features to ensure that customer information stored in the eSMG remains confidential. Each customer's information is accessible only by authorized members of the HP Mission-Critical and HP support team in charge of that customer. Customer access to the eSMG is reserved to customers who have valid service contracts.

Authorized customers can access their data from the Internet through a secured port using an SSL 3.0 protocol and 128-bit key. All information sent and received over the Internet is encrypted. Customers access their personal information using a personal username and password that an HP representative provides. By default, Internet access to customer information is set to read-only mode.

Initially, only the Technical Account Manager assigned to a customer and the eSMG administrator have access to customer data. Granting access to the customer information to additional HP Services employees requires the approval of the TAM.

### Access IDs

All data stored in the RCM Server is secure. The RCM server and eSMG use access identifiers to identify data collections and to control access to these collections. An identifier is assigned to a system when the RCM collector is configured. As a result, every RCM data collection has an associated access identifier, which represents the system, or group of systems, from which data was collected. All RCM reports associated with that access identifier are available in eSMG within one hour of the time data is sent from a customer's system.

## RCM Reports

The reports generated by RCM and available from eSMG are the following:

| **Report Type** | **Description** |
|---|---|
| Configuration | Displays a detailed inventory of hardware and software components installed on a selected system. |
| Change | Shows configuration changes that might have occurred on a selected system by comparing data from two different collections from this system. |
| Comparison | Shows the differences between any two similar systems by comparing data collected from each system. This is useful, for example, to indicate changes between a test and a production system. |
| Patch | Recommends any patches that should be applied to the system. |
| Raw Data | The text file that the collector produces for each system. The RCM Server software parses it and loads it into the database. |
| SAN Configuration | Shows the topology of the SAN as well as configuration data of the devices in the SAN. |
| EVA | Shows details of the specified EVA devices including the controllers, LUNS, disks, and so on. |
| AVANTO | An HP availability modeling tool; RCM can generate an input file for the storage portion of a customer's system. |

Further explanations and examples of these report types are in the last major section of this paper.

## Collector Installation

The RCM OpenVMS collector consists of a number of Digital Command Language (DCL) command procedures as well as several OpenVMS VAX and Alpha binary executables that parts of the collection require. One node initiates the collection process for all the selected nodes. This node manages the overall collection process and sends the collected data to the RCM server at HP.

The RCM OpenVMS data collector is available as a free download from the HP software depot site and is installed using the POLYCENTER Software Installation Utility.  You use the DCL command PRODUCT INSTALL RCM to install the kit. By default, RCM will be installed on the system disk, but you can install it on another disk by specifying the /DESTINATION=*device_name:[directory_name*] qualifier.

To de-install RCM, you use the DCL command PRODUCT REMOVE RCM.

**Cluster Installation**
You usually need to install RCM on only one node in a cluster, and it can collect data from all nodes. The only requirement is that each node must have read and write access to the disk on which RCM is installed.

## Collector Configuration

The RCM_START.COM procedure initiates the collection process. Execute this procedure on one node in a cluster.

When you run RCM_START for the first time, as part of the installation procedure, it prompts for configuration information that is written to a configuration file in the RCM$DATA directory. By default, the configuration file is named <*nodename*>.CFG.

The RCM_START procedure also asks you to select the nodes from which to collect data, and it tries to obtain the serial numbers of the selected systems. On VAX systems and some older Alpha models, it is not possible to obtain the serial number from the operating system; therefore, RCM prompts for the serial number. Because RCM and ESMG use the serial number as part of the identifying information, it is important to enter the number correctly. On newer Alpha systems, the serial number is obtained from the system using the output from the SHOW CPU/FULL command, which is included in the RCM_START procedure.

The RCM configuration file specifies all aspects of the RCM collection including the following:

- Selected nodes and serial numbers
- Collection frequency
- Date and time of next collection
- Transport option
- SAN switch data collection
- EVA data collection
- Run RCM on reboot
- Number of data collections to archive on the system

You can edit the configuration file using an editor or the built-in editing feature of RCM_START.

Example 1shows a sample RCM configuration file.

```
; RCM V5.0-601 Configuration file ALPHA.CFG
; created by RCM_START.COM on node ALPHA1 at 20-APR-2004 16:51:00.09
;
[COMPANY NAME]                        xxxxxxxxxx
[CONTACT NAME]                        xxxx xxxxx
[CONTACT TELEPHONE]                   xxx-xxx-xxxx
[CONTACT EMAIL]                       xxx.xxxxx@xxxxxx.xom
[ACCOUNT MANAGER]                     xxxxx xxxxxx
[ACCOUNT MANAGER EMAIL]               xxxxx.xxxxxx@hp.com
[CUSTOMER ACCESS ID]                  DEMO
[TRANSPORT OPTION]                    DSN
[COLLECTION FREQUENCY]                WEEKLY
[NEXT COLLECTION TIME]                2-MAY-2004 13:53:25.15
[MAX ARCHIVES]                        10
[ARCHIVE DIRECTORY]                   RCM$ROOT:[RCM_ARCHIVE]
[COLLECTION DIRECTORY]                RCM$ROOT:[DATA]
[FTP AREA]                            rcm.support.compaq.com/to_rcm/
[FTP PROCEDURE]
[EMAIL ADDRESS]                       xxx.xxxx@hp.com
[LOCAL SITE]                          Y
[LOCAL TRANSPORT OPTION]              FTP
[LOCAL EMAIL ADDRESS]
[LOCAL HOSTNAME]                      xxxxx.xxx.xxxxxx.com
[LOCAL LOGIN]                         anonymous
[LOCAL UPLOADDIR]                     /pub/vms
[LOCAL FTP PROCEDURE]
[SUPPRESS IP ADDRESSES]               N
[RUN ON REBOOT]                       N
[FCT BINARY IMAGE]                    N
[DATA FOR ANALYSIS]                   F
[SAN SWITCH DATA COLLECTION]          Y
[SAN SWITCH SNMPWALK_TIMEOUT]         15
[EVA DATA COLLECTION]                 Y
[NODES]                               ALPHA1,ALPHA2,ALPHA3,ALPHA4,ALPHA5,ALPHA6
[ALPHA1 SYSTEM SERIAL NUMBER]         AY22001634
[ALPHA2 SYSTEM SERIAL NUMBER]         AY33407373
[ALPHA3 SYSTEM SERIAL NUMBER]         AY42602765
[ALPHA4 SYSTEM SERIAL NUMBER]         AY22602766
[ALPHA5 SYSTEM SERIAL NUMBER]         AY23407373
[ALPHA6 SYSTEM SERIAL NUMBER]         AY12001637
```

**Example 1 RCM Configuration File**

The RCM_START procedure starts a detached process, which runs the RCM_COLLECT.COM procedure, on each selected node in the cluster. The process name of each detached process is "RCM_COLLECT". The data for each system is written to a text file in the RCM$DATA directory with a naming convention of RCMO-<nodename>-yyyymmdd-hhmmss.TXT.

The RCM_COLLECT process on the node that initiated the RCM collection is responsible for managing the overall collection. When all the other collection processes have completed, the initial node's collect process adds each system's RCM data files to a ZIP archive, which is then transported to HP using the selected transport option.

If email is selected as the transport option, the ZIP file is first converted to a text file using the uuencode utility included in the RCM Kit; it is then emailed to HP using VMSMAIL. If no transport option is selected, the user should send the data to the RCM server in HP using the most convenient option, such as sending the ZIP file as an attachment to an email or using ftp to send the data. The size of an RCM collection ZIP file depends on many factors, such as the number of nodes and devices and whether SAN and EVA collection is enabled. The size of the archive is usually less than 1MB.

The RCM collected data can also be sent to another system at the customer site, using either FTP or email, by setting the LOCAL SITE option to Y and entering the transport details. You can use this method to centralize all site RCM data in one location. You might need to use this method if you have a system without a connection to the internet in order to move data to a gateway node for later transport to HP.

### Scheduling Collections of Data

The required collection schedule for RCM data is a configuration option. The default setting is to collect RCM data once a month but you can choose daily, weekly, or quarterly collections instead. If you do not need a regular schedule, you can configure RCM to collect information only on demand. After you set up a schedule, a detached process is started, and the process hibernates until the next scheduled time. During the collection process, the date for the next collection is updated, and another detached process is started.

RCM adds an entry to the SYSMAN startup database so that when a system reboots, the collection schedule is preserved. To see the RCM entry, enter the following command:

```
$ MCR SYSMAN STARTUP SHOW FILE RCM/FULL
```

This command shows the entry RCM$STARTUP_ *nodename*.COM enabled on the node with RCM installed. The RCM$STARTUP_*nodename*.COM procedure in SYS$STARTUP defines the required RCM logicals and resets the collection schedule when the system reboots. Note that only the system on which RCM was installed runs the RCM procedure at startup.

### Monitoring and Stopping an RCM Collection

The duration of an RCM collection depends on the system type and configuration; however, it usually takes less than fifteen minutes. You can monitor the progress of a collection by executing the RCM$DIR:RCM_STATUS.COM procedure. This procedure shows the current status of the collection on each system and the scheduled date and time of the next collection.

You can use the RCM$DIR:RCM_STOP procedure at any time to stop the collection process and to cancel future collections. Running RCM_START again resets the scheduled collections.

## Data That Is Collected

The RCM_COLLECT process runs on each selected system and collects detailed system information. Even for customers without a service contract, the raw data file is a useful source of information for the system manager.

The collected data file is written as a series of tag-delimited sections as follows:

```
--- START RCM <section name> ---

--- END RCM <section name> ---
```

Some examples of sections of the raw data file are shown in Example 2.

```
 --- START RCM OPERATING SYSTEM ---

Operating System = VMS V7.3-1

Console Version = V6.6-1111

Palcode Version = 1.98-2

--- END RCM OPERATING SYSTEM ---



--- START RCM NODE ---

Node Host Name = ALPHA1
Node System Type = AlphaServer GS160 6/1224
Node Domain Name =
--- END RCM NODE ---
```

**Example 2 Raw Data Sections**

The following sections describe the types of data that RCM collects.

### Configuration Tree Data (FRU)

A configuration tree is a memory structure containing system hardware resource configuration and associated Field Replaceable Unit (FRU) information on AlphaServer systems. The configuration tree, which is built by the console firmware, is a permanent data structure in system memory; it is also written to the binary error log. RCM uses two separate products, DECevent and WEBES, to read the FRU data:

- DECevent

  On AlphaServer systems with FRU Version 4.0 (models 1200, 4X00, 8X00, GS60, GS140), DECevent is required to translate the FRU table. DECevent is a hardware fault-management diagnostic tool that reads hardware configuration information from an error log. RCM uses the OpenVMS Analyze/System command CLUE FRU to generate a file with a dummy binary error log record for DECevent to analyze. Note that no Product Authorization Kit (PAK) is required for the error log translation feature of DECevent.

- WEBES

  The HP Service Tool WEBES is required on systems with FRU Version 5.0 to enable RCM to collect additional hardware and firmware information from the configuration tree. Some of the AlphaServer systems supporting FRU Version 5.0 are models DS10, DS10L, DS20, DS20E, DS25, ES40, ES45, GS80, GS160, GS320, and GS1280. RCM determines if a system is a FRU Version 5.0 system by checking the value that F$GETSYI("SYSTYPE") returns. Alpha systems with a system type higher than 33 are FRU Version 5.0 systems. Note that the "Desta Director" process is not required to be running on RCM Version 4.2.

The collected configuration tree data is written to the RCM data file as a series of name=value pairs as shown in the short extract in Example 3.

```
--- START RCM CT5 ---
CT5START
NODE NODE_ROOT FRU Version 5.2
NodeHandle=0x0
Child=0x240
Child=0x2c0
Child=0x10c0
FRAME GCT_Node_HD GCT_Node_HD
N_Type=1
N_Sbtyp=0
N_Size=x0240
Hd_extension=x00000000
Owner_Handle=x00000000
Current_Owner_Handle=x00000000
Node_ID=x0000000000000000
Node_Flags=x0000000000000000
Direct_Ancestor_Handle=x00000000
Affinity_Handle=x00000000
Parent_Handle=x00000000
Next_Sibling_Handle=x00000000
Previous_Sibling_Handle=x00000000
Child_Handle=x00000240
.
. (thousands of lines omitted!)
.
--- END RCM CT5 ---
```

**Example 3 Portion of Configuration Tree Data**

### Clue Configuration Data

The collector uses the Analyze/System SDA command CLUE CONFIG to obtain detailed system, memory, and device configuration information.

### Clue SCSI Data

RCM uses the Analyze/System SDA command CLUE SCSI/SUMMARY to collect SCSI configuration data, including all ports, targets, and connections with attached devices. Device types and hardware revision information are included.

### Installed Software Product Information

RCM uses the DCL command PRODUCT SHOW PRODUCT/FULL to find which products and patches have been installed using the DCL command PRODUCT INSTALL. RCM also searches the VMSINSTAL.HISTORY file to find any products installed using the older installation tool VMSINSTAL.

### License Information

RCM uses the OpenVMS DCL command SHOW LICENSE/UNIT_REQUIREMENT to collect details of licenses loaded on the system.

### Device Information

To collect detailed information about all disk and tape devices attached to the system, RCM uses the DCL command SHOW DEVICE/FULL and the lexical F$GETDVI with various item codes.

### Network Configuration Information

Detailed network configuration information is usually collected by displaying hardware and IP addresses. Customers who do not want to display this information can skip this section by setting the [SUPPRESS IP ADDRESSES] option in the configuration file to Y.

### Hard and Soft Partition Information

RCM can collect details of the hard and soft partitions for AlphaServers systems. Hard partitioning is a physical separation of computing resources by hardware-enforced access barriers. No resource sharing exists between hard partitions. Soft partitioning is a separation of computing resources by software-controlled access barriers. Read and write access across a soft partition boundary is controlled by the operating system. OpenVMS Galaxy is an implementation of soft partitioning.

Hard and soft partition information, which RCM and ESMG require to properly identify a system, is found in configuration tree data. If a system does not support partitions, the hard and soft partition values are set to -1. Example 4 shows the hard and soft partition information in the raw data file for a typical system:

```
--- START RCM PARTITION IDS ---

hard: 0
soft: 1,ALPHA2

--- END RCM PARTITION IDS ---
```

**Example 4 Collected Hard/Soft Partition IDs**

Example 5 shows a section of an RCM configuration report with platform partition information.

| Hard Partition Id | Soft Partition Id | Instance Name | Instance OS | Date of Last Collection |
|---|---|---|---|---|
| 0 | 0 | ALPHA1 | VMS V7.3-1 | 2004-04-27 02:52:32 |
| 0 | 1 | ALPHA2 | VMS V7.3-1 | 2004-04-27 02:58:31 |
| 0 | 2 | ALPHA3 | VMS V7.3-1 | 2004-04-27 03:01:26 |

**Example 5 Platform Partition Table from an RCM Report**

### GALAXY and RAD Data

For systems with OpenVMS Galaxy software, additional information is collected using the following F$GETSYI Galaxy item codes:

```
galaxy_platform, galaxy_member, galaxy_id, community_id,
partition_id
```

RCM uses the Analyze/System SDA command SHOW GALAXY to capture the state of all the instances in the Galaxy configuration and collects Resource Affinity Domain (RAD) data using the following F$GETSYI item codes:

```
rad_max_rads, rad_cpus, rad_memsize and rad_shmemsize
```

The output from the following command, which shows how much memory is mapped in each RAD, is also collected:

```
$ MCR SYS$TEST:RADCHECK.EXE –allprocs
```

## ProPatch Data

RCM collects input data for the separate tool ProPatch and writes it to a .LIF file. The RCM server automatically submits the LIF file to the ProPatch server, which returns a report that recommends installing specific patches. This functionality is derived from the former Digital Equipment Corporation tools DASC and LIFE.

The LIF files consists of a listing of all the executable images in several system directories such as SYS$SYSTEM, SYS$LOADABLE_IMAGES and SYS$SHARE. Each entry in the LIF file shows the image name, link time, and version identification. This information is extracted from the image header.

The ProPatch server uses the image link times to determine if any patches are required. See Example 6 and Example 7 for portions of a typical .LIF file and the corresponding ProPatch report, which recommends installing a critical patch based on the link time of a system executable file. In this example, ProPatch recommends installing the critical patch VMS731_XFC-V0200 because it has detected that the system has an old version of SYS$SHARE:ALPHA_XF$SDA.EXE.

```
--- START CustomerInfo START ---
HP
ALPHA1
ALP
12-APR-2004
RCM V5.0-601
--- END CustomerInfo END ---
~Hardware
~Software
OpenVMS Alpha|001|V7.3-1|$3$DKB1:
SYS$SYSTEM:RRV.EXE|04-OCT-2003 08:47:55.97|050989GKF|$3$DKB1:
SYS$SYSTEM:ACC.EXE|18-JUL-2002 19:52:58.23|X-16|$3$DKB1:

...

SYS$SHARE:ADARTL.EXE|18-JUL-2002 19:50:57.52|V7.3-2|$3$DKB1:
SYS$SHARE:ALPHA_XFC$SDA.EXE|07-MAR-2003 11:33:37.85|V1.0|$3$DKB1:
SYS$SHARE:AS$SDA.EXE|18-DEC-2002 01:05:23.32|V02.00|$3$DKB1:
```

**Example 6 Section of a .LIF File**

```
  Image on your system:
           VMS version:  OPENVMS ALPHA V7.3-1
            image name:  $3$DKB1:SYS$SHARE:ALPHA_XFC$SDA.EXE
         image file id:  V1.0
        link date/time:  07-MAR-2003 11:33:37.85

        Service action:  VMS731_XFC-V0200
  Patch reference name:  DEC-AXPVMS-VMS731_XFC-V0200--4.PCSI    --CRITICAL--

 XFC fixes.
```

**Example 7 Portion of a ProPatch Report**

## Storage Controller Information

RCM collects information from storage controllers using the HP StorageWorks Command Scripter utility, which is bundled with the RCM collector. This is a command-line tool that can safely interrogate a storage controller. Earlier versions of RCM used HSZTERM, which is no longer supported.

The Command Scripter command line (CLI) command '`-j subsysdata`' is used to find available HSJ devices and the '`-f subsysdata`' command is used to obtain a list of available HSZ and HSG storage controllers.

The following command script CLI commands are used for HSJ devices:

```
SHOW THIS_CONTROLLER FULL
SHOW OTHER_CONTROLLER FULL
SHOW STORAGESETS FULL
SHOW FAILEDSET FULL
SHOW SPARESETS FULL
SHOW UNITS FULL
SHOW DEVICES FULL
```

For HSG and HSZ devices, enter these commands:

```
SHOW THIS_CONTROLLER FULL
SHOW ASSOCIATIONS FULL
SHOW CONCATSETS FULL
SHOW CONNECTIONS FULL
SHOW REMOTE_COPY FULL
SHOW STORAGESETS FULL
SHOW UNITS FULL
SHOW DEVICES FULL
SHOW OTHER_CONTROLLER FULL
SHOW EMU
```

If the Command Console LUN (CCL) is enabled at the console, a CCL device is used to access a HSG80 controller. Otherwise, one of the attached devices is used to access the controller. The CCL devices have device names such as $1$GGAnnnn.

The firmware revision of the storage controllers is in the output of the SHOW THIS_CONTROLLER command as shown Example 8.

```
--- START RCM HSG ---

SANworks Command Scripter V1.0B Build 076
SHOW THIS_CONTROLLER FULL
Controller:
        HSG80 ZG11305292 Software V87F-1, Hardware E12
        NODE-_ID         = 5000-1FE1-0011-8F20
        ALLOCATION_CLASS = 0
        SCSI_VERSION     = SCSI-3
        Configured for MULTIBUS_FAILOVER with ZG11305742
            In dual-redundant configuration
```

**Example 8  SHOW THIS_CONTROLLER Output**

To obtain the firmware revision of Fibre Channel Adapters (for example, KGPSA), use the following Analyze/System command:

```
SDA> FC SHOW DEVICE FGxx:
```

Example 9 shows typical output of this command in the raw data file.

```
--- START RCM SDA FC SHOW DEV FG ---

FGAO: operational firmware revision DS3.81A4
port_name(adapter_id) = 1000-0000-C92B-0098, node_name(host_id) = 2000-0000-C92B-0098
FGB0: operational firmware revision DS3.81A4
Port_name(adapter_id) = 1000-0000-C92A-FF07, node_name(host_id) = 2000-0000-C92A-FF07

--- END RCM SDA FC SHOW DEV FG ---
```

**Example 9 KGPSA Firmware Revision**

## SAN Switch Data Collection

You can configure RCM to collect configuration information from HP-supported switches that are used in Storage Area Networks. To collect this data, RCM requires the IP address of each SAN switch as well as its SNMP community string. The RCM collector issues a passive request for data to each designated SAN switch using the SNMP protocol.

This option is available only on systems running the HP TCPIP product because the OpenVMS RCM collector uses the TCPIP utility SYS$SYSTEM:TCPIP$SNMP_REQUEST.EXE to collect the SAN Management Information Base (MIB) data shown in Example 10.

SAN configuration reports are generated from the collected MIB data and are made available to customers in the eSMG. Excerpts from sample reports are in the last sections of this paper.

```
--- START RCM SAN SWITCH OUTPUT ---

START SAN SWITCH 192.168.12.10
1.3.6.1.2.1.1.1.0 = Fibre Channel Switch.
1.3.6.1.2.1.1.2.0 = 1.3.6.1.4.1.1588.2.1.1.12
1.3.6.1.2.1.1.3.0 = 1126391968 = 130 d 8:51:59
1.3.6.1.2.1.1.4.0 = Field Support.
1.3.6.1.2.1.1.5.0 = SAN01
1.3.6.1.2.1.1.6.0 = End User Premise
1.3.6.1.2.1.1.7.0 = 79
1.3.6.1.2.1.2.1.0 = 3
1.3.6.1.2.1.2.2.1.1.1 = 1
1.3.6.1.2.1.2.2.1.1.2 = 2
1.3.6.1.2.1.2.2.1.1.3 = 3
1.3.6.1.2.1.2.2.1.2.1 = lo
1.3.6.1.2.1.2.2.1.2.2 = eth0
1.3.6.1.2.1.2.2.1.2.3 = fc0
1.3.6.1.2.1.2.2.1.3.1 = 24
1.3.6.1.2.1.2.2.1.3.2 = 6
1.3.6.1.2.1.2.2.1.3.3 = 56
1.3.6.1.2.1.2.2.1.4.1 = 16436
1.3.6.1.2.1.2.2.1.4.2 = 1500
1.3.6.1.2.1.2.2.1.4.3 = 2024
1.3.6.1.2.1.2.2.1.5.1 = 0
. . .
```

**Example 10 SAN Switch MIB Data**

## EVA Data

RCM V5.0 can collect configuration information from HSV110-based Enterprise Virtual Array Storage Systems. To enable EVA data collection, the RCM configuration file must have the EVA DATA COLLECTION option set to Y.

RCM uses the Storage System Scripting Utility (SSSU) to communicate with a Command View EVA (also called the Storage Element Manager), running on the SAN Appliance Manager. This communication is necessary to find out what EVA storage systems, or cells, the appliance manager is responsible for and also to collect revision and configuration information for a particular cell. Each cell represents, at a logical level, all the components that make up the EVA storage system, including cabinets, power supplies, disks, and controllers.

To communicate with the Element Manager, the RCM collector requires the IP addresses and valid access details for each SAN Appliance Manager. This information is stored in the file RCM$DATA:EVA.INI using the format shown in Example 11.

Supported versions of the Element Manager are Version 2 and Version 3. SSSU Version 2 is required for Element Manager Version 2; SSSU Version 3 is required for Element Manager Version 3. Both versions of the OpenVMS Alpha SSSU utility are included in the RCM collector kit. It is important to know the versions of Element Manager running on each appliance manager so that you use the correct version of SSSU.

```
# The following section is used to list the required information for SAN
# Appliance Managers which are running Element Manager V2.
#
#[SSSU v2]
#<IP address>:<username>:<password>
#<IP address>:<username>:<password>
# The following section is used to list the required information for SAN
# Appliance Managers which are running Element Manager V3.
#
[SSSU v3]
192.168.0.10:administrator:password
```

**Example 11 EVE.INI with Details of a V3 Element Manager**

### SYSGEN Parameter Information

To collect OpenVMS System Generation Utility (SYSGEN) parameters, RCM uses the SYSGEN commands SHOW/ALL and SHOW/SPECIAL. The contents of the file SYS$SYSTEM:MODPARAMS.DAT are also included in the RCM collected data.

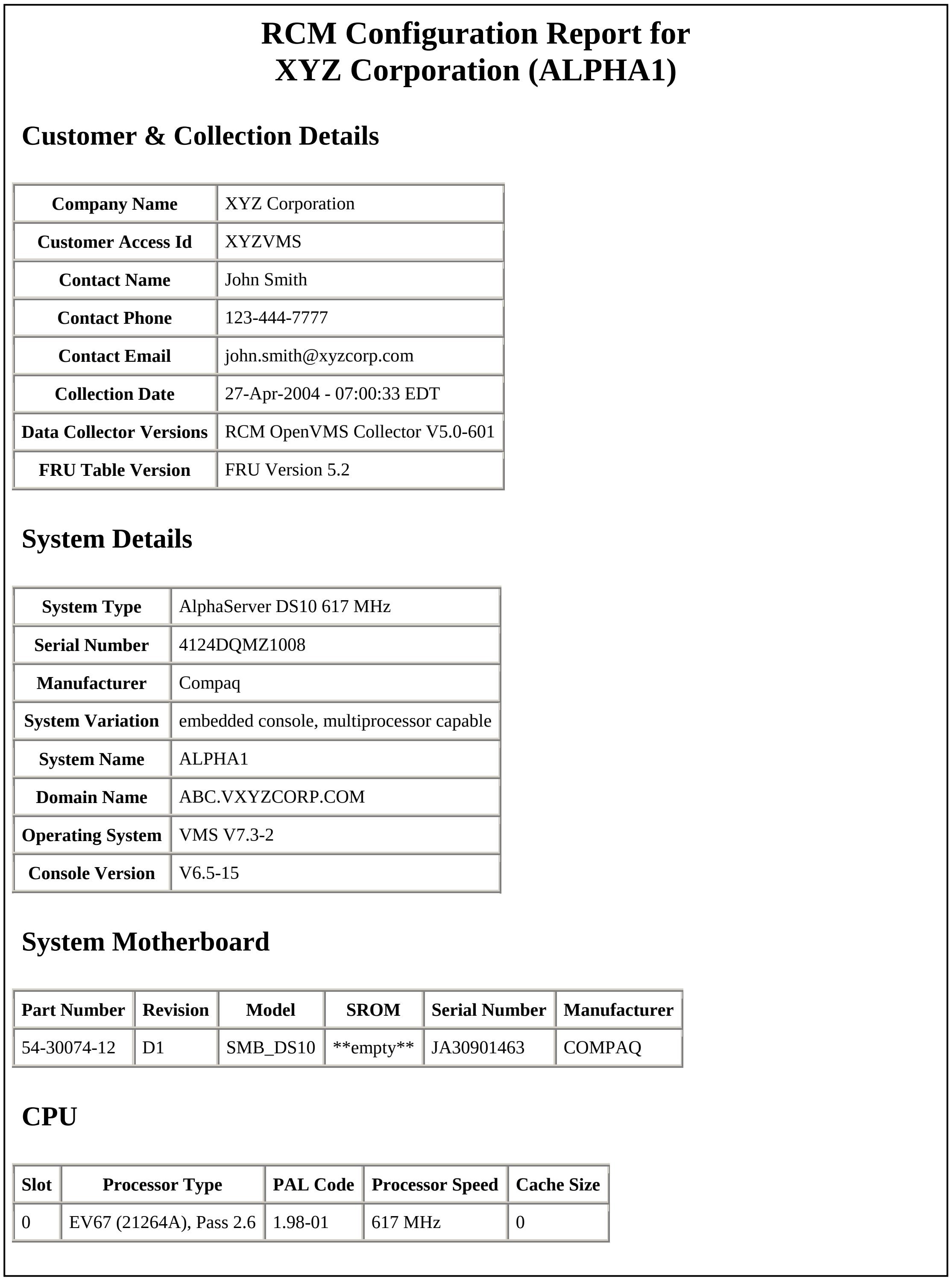
The RCM Configuration Report does not show all the SYSGEN parameters; however, the Change Report shows any differences in the SYSGEN parameters between any two collections. This can be useful in diagnosing problems that might be due to changed SYSGEN parameters. To see all the parameters, you can view the raw data.

## Reports

The following sections contain examples and descriptions of sample RCM reports.

### Configuration Report

The following figures show excerpts from a sample RCM Configuration Report (many sections have been omitted from these examples). The data for the various tables is collected from many parts of the raw data. Hardware part numbers and revision information are derived from configuration tree data.



## RCM Configuration Report for XYZ Corporation (ALPHA1)

### Customer & Collection Details

| | |
|---|---|
| **Company Name** | XYZ Corporation |
| **Customer Access Id** | XYZVMS |
| **Contact Name** | John Smith |
| **Contact Phone** | 123-444-7777 |
| **Contact Email** | john.smith@xyzcorp.com |
| **Collection Date** | 27-Apr-2004 - 07:00:33 EDT |
| **Data Collector Versions** | RCM OpenVMS Collector V5.0-601 |
| **FRU Table Version** | FRU Version 5.2 |

### System Details

| | |
|---|---|
| **System Type** | AlphaServer DS10 617 MHz |
| **Serial Number** | 4124DQMZ1008 |
| **Manufacturer** | Compaq |
| **System Variation** | embedded console, multiprocessor capable |
| **System Name** | ALPHA1 |
| **Domain Name** | ABC.VXYZCORP.COM |
| **Operating System** | VMS V7.3-2 |
| **Console Version** | V6.5-15 |

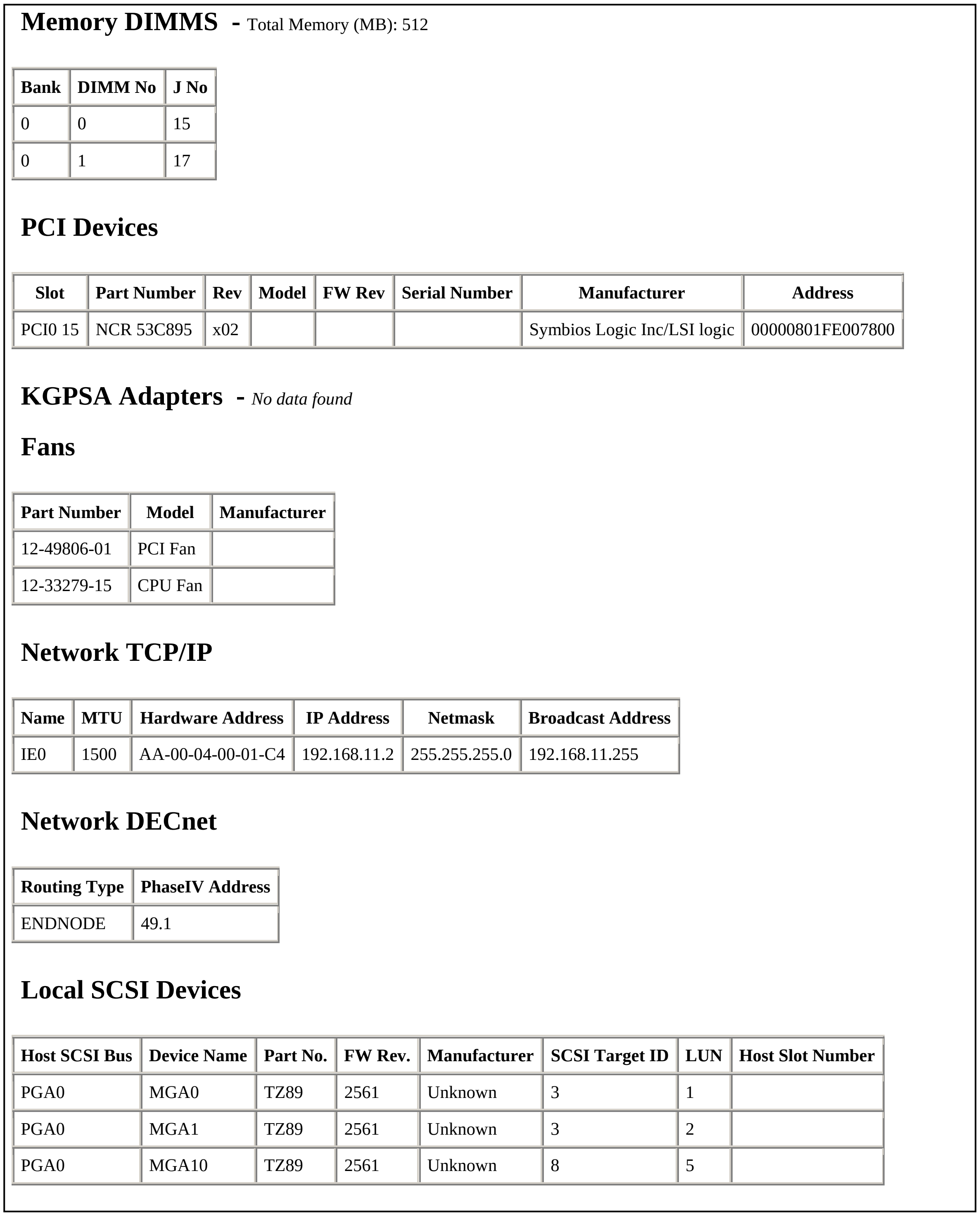
### System Motherboard

| Part Number | Revision | Model | SROM | Serial Number | Manufacturer |
|---|---|---|---|---|---|
| 54-30074-12 | D1 | SMB_DS10 | **empty** | JA30901463 | COMPAQ |

### CPU

| Slot | Processor Type | PAL Code | Processor Speed | Cache Size |
|---|---|---|---|---|
| 0 | EV67 (21264A), Pass 2.6 | 1.98-01 | 617 MHz | 0 |

**Figure 2 Sections of an RCM Configuration Report**



## Memory DIMMS - Total Memory (MB): 512

| Bank | DIMM No | J No |
|---|---|---|
| 0 | 0 | 15 |
| 0 | 1 | 17 |

## PCI Devices

| Slot | Part Number | Rev | Model | FW Rev | Serial Number | Manufacturer | Address |
|---|---|---|---|---|---|---|---|
| PCI0 15 | NCR 53C895 | x02 | | | | Symbios Logic Inc/LSI logic | 00000801FE007800 |

## KGPSA Adapters - *No data found*

## Fans

| Part Number | Model | Manufacturer |
|---|---|---|
| 12-49806-01 | PCI Fan | |
| 12-33279-15 | CPU Fan | |

## Network TCP/IP

| Name | MTU | Hardware Address | IP Address | Netmask | Broadcast Address |
|---|---|---|---|---|---|
| IE0 | 1500 | AA-00-04-00-01-C4 | 192.168.11.2 | 255.255.255.0 | 192.168.11.255 |

## Network DECnet

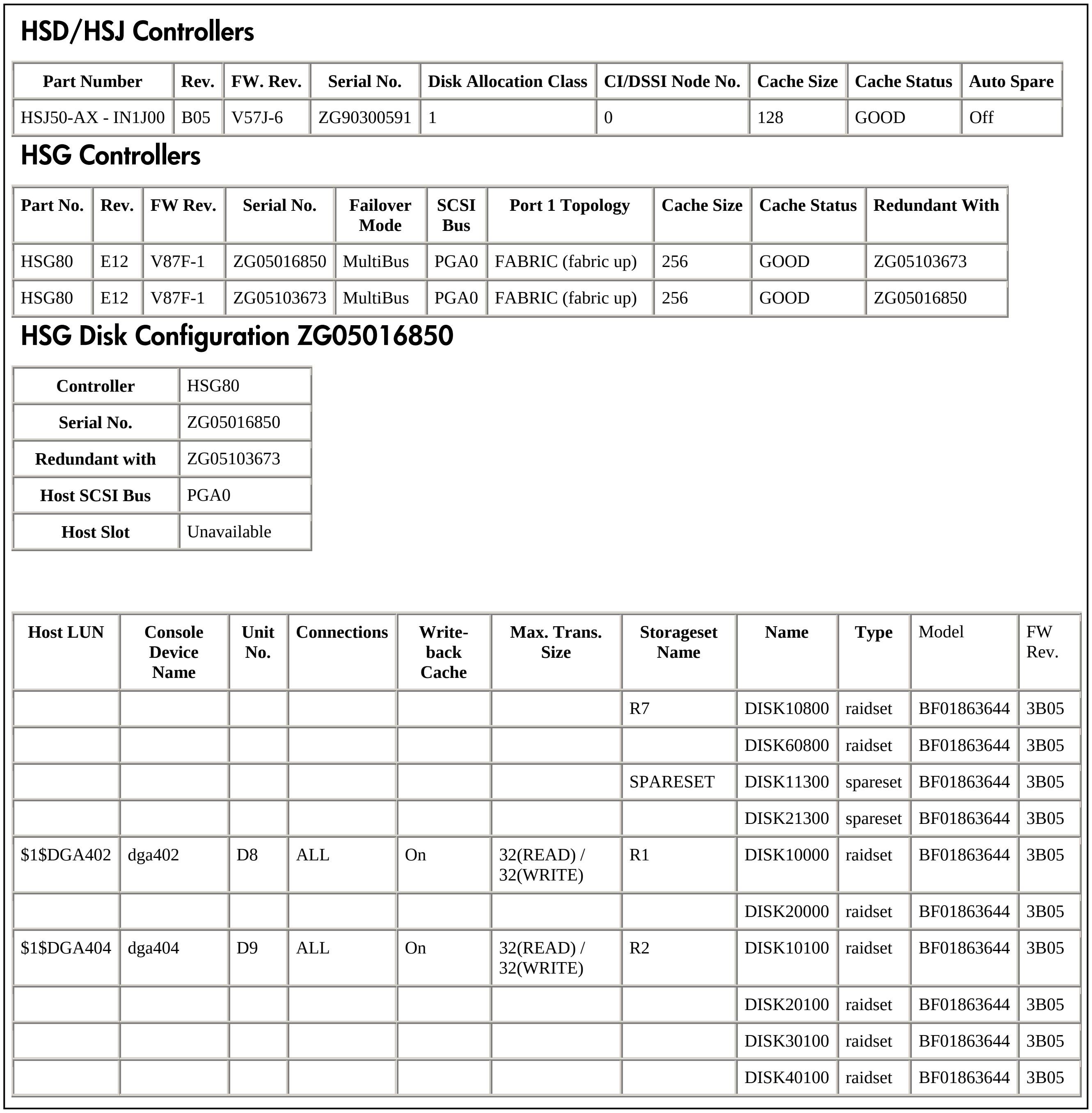
| Routing Type | PhaseIV Address |
|---|---|
| ENDNODE | 49.1 |

## Local SCSI Devices

| Host SCSI Bus | Device Name | Part No. | FW Rev. | Manufacturer | SCSI Target ID | LUN | Host Slot Number |
|---|---|---|---|---|---|---|---|
| PGA0 | MGA0 | TZ89 | 2561 | Unknown | 3 | 1 | |
| PGA0 | MGA1 | TZ89 | 2561 | Unknown | 3 | 2 | |
| PGA0 | MGA10 | TZ89 | 2561 | Unknown | 8 | 5 | |

**Figure 3 Additional Sample Tables from Configuration Reports**

Figure 4 gives some examples of HSJ and HSG Storage controller information from a configuration report. Several columns and rows have been removed for readability.



### HSD/HSJ Controllers

| Part Number | Rev. | FW. Rev. | Serial No. | Disk Allocation Class | CI/DSSI Node No. | Cache Size | Cache Status | Auto Spare |
|---|---|---|---|---|---|---|---|---|
| HSJ50-AX - IN1J00 | B05 | V57J-6 | ZG90300591 | 1 | 0 | 128 | GOOD | Off |

### HSG Controllers

| Part No. | Rev. | FW Rev. | Serial No. | Failover Mode | SCSI Bus | Port 1 Topology | Cache Size | Cache Status | Redundant With |
|---|---|---|---|---|---|---|---|---|---|
| HSG80 | E12 | V87F-1 | ZG05016850 | MultiBus | PGA0 | FABRIC (fabric up) | 256 | GOOD | ZG05103673 |
| HSG80 | E12 | V87F-1 | ZG05103673 | MultiBus | PGA0 | FABRIC (fabric up) | 256 | GOOD | ZG05016850 |

### HSG Disk Configuration ZG05016850

| | |
|---|---|
| **Controller** | HSG80 |
| **Serial No.** | ZG05016850 |
| **Redundant with** | ZG05103673 |
| **Host SCSI Bus** | PGA0 |
| **Host Slot** | Unavailable |

| Host LUN | Console Device Name | Unit No. | Connections | Write-back Cache | Max. Trans. Size | Storageset Name | Name | Type | Model | FW Rev. |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | R7 | DISK10800 | raidset | BF01863644 | 3B05 |
| | | | | | | | DISK60800 | raidset | BF01863644 | 3B05 |
| | | | | | | SPARESET | DISK11300 | spareset | BF01863644 | 3B05 |
| | | | | | | | DISK21300 | spareset | BF01863644 | 3B05 |
| $1$DGA402 | dga402 | D8 | ALL | On | 32(READ) / 32(WRITE) | R1 | DISK10000 | raidset | BF01863644 | 3B05 |
| | | | | | | | DISK20000 | raidset | BF01863644 | 3B05 |
| $1$DGA404 | dga404 | D9 | ALL | On | 32(READ) / 32(WRITE) | R2 | DISK10100 | raidset | BF01863644 | 3B05 |
| | | | | | | | DISK20100 | raidset | BF01863644 | 3B05 |
| | | | | | | | DISK30100 | raidset | BF01863644 | 3B05 |
| | | | | | | | DISK40100 | raidset | BF01863644 | 3B05 |

**Figure 4 HSx Storage Controller Tables from Configuration Reports**

## ProPatch Report

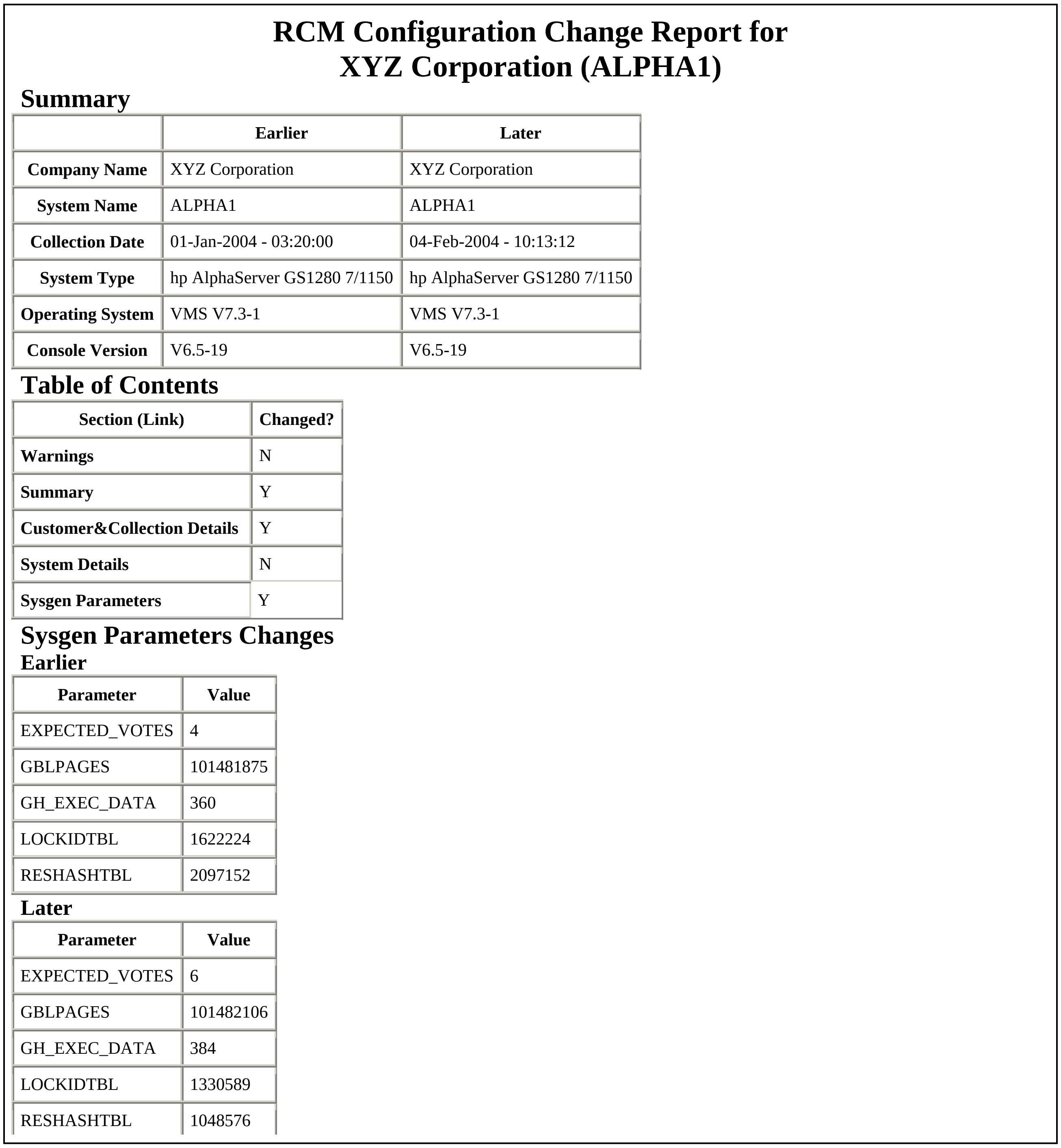
Figure 5 shows a partial ProPatch Report.

```
This Proactive Patch report is being provided to XYZ Corporation
as part of your Business Critical Support Services. The purpose of
this report is to assist in managing your system by identifying
patches and allowing you to proactively decide the applicability
to your environment and business needs.

Data within this report was collected from the system disk of
ALPHA node ALPHA11 and will pertain to all nodes within the cluster
that share the same system disk. If the cluster has multiple
system disks then a separate Proactive Patch report will need
to be generated for each system disk.

NOTES:

1. Some patches in this report are identified as being --CRITICAL--.
   This indicates that the problem(s) addressed by the patch have the potential
   to cause system outages (i.e. hang or crash).

2. Some patches require the prior installation of other patches. For
   patches that have this prerequisite, please read the patch release
   notes for additional information.

3. If you plan to install multiple patches simultaneously it is recommended you
   install them in a test environment to test with your applications before
   implementing in a production environment.

============================================================================
============================================================================

    LIFE Analysis Report
        For: ALPHA1
        Date: Tue Apr 27 05:43:26 2004
        Report Type: BRIEF
        Rules Database Version Number: MBM-MAR-2004
        ProPatch executable Version Number: V4.30
============================================================================
============================================================================
    Image on your system:
               VMS version:  OPENVMS ALPHA V7.3-2
                image name:  ALPHA1$DKA0:SYS$SHARE:DECW$DRM_PRIV.EXE
             image file id:  DW V7.3-2031001
            link date/time:  01-OCT-2003 21:37:43.86

            Service action:  VMS732_GRAPHICS-V0200
      Patch reference name:  DEC-AXPVMS-VMS732_GRAPHICS-V0200--4.PCSI

    Graphics fixes.

============================================================================
    Image on your system:
               VMS version:  OPENVMS ALPHA V7.3-2
                image name:  ALPHA$DKA0:SYS$SYSTEM:DCL.EXE
             image file id:  X-36
            link date/time:  09-OCT-2003 15:46:09.42

            Service action:  VMS732_DCL-V0100
      Patch reference name:  DEC-AXPVMS-VMS732_DCL-V0100--4.PCSI         --CRITICAL--

    DCL fixes.

============================================================================
```

**Figure 5 Sample ProPatch Report**

## Change Report

Figure 6 shows part of a Change Report illustrating SYSGEN parameter changes.



**RCM Configuration Change Report for XYZ Corporation (ALPHA1)**

**Summary**

| | Earlier | Later |
|---|---|---|
| **Company Name** | XYZ Corporation | XYZ Corporation |
| **System Name** | ALPHA1 | ALPHA1 |
| **Collection Date** | 01-Jan-2004 - 03:20:00 | 04-Feb-2004 - 10:13:12 |
| **System Type** | hp AlphaServer GS1280 7/1150 | hp AlphaServer GS1280 7/1150 |
| **Operating System** | VMS V7.3-1 | VMS V7.3-1 |
| **Console Version** | V6.5-19 | V6.5-19 |

**Table of Contents**

| Section (Link) | Changed? |
|---|---|
| **Warnings** | N |
| **Summary** | Y |
| **Customer&Collection Details** | Y |
| **System Details** | N |
| **Sysgen Parameters** | Y |

**Sysgen Parameters Changes**

**Earlier**

| Parameter | Value |
|---|---|
| EXPECTED_VOTES | 4 |
| GBLPAGES | 101481875 |
| GH_EXEC_DATA | 360 |
| LOCKIDTBL | 1622224 |
| RESHASHTBL | 2097152 |

**Later**

| Parameter | Value |
|---|---|
| EXPECTED_VOTES | 6 |
| GBLPAGES | 101482106 |
| GH_EXEC_DATA | 384 |
| LOCKIDTBL | 1330589 |
| RESHASHTBL | 1048576 |

**Figure 6 Portion of a Change Report**

## SAN Reports

A typical SAN report contains the following information:

- Configuration information for individual fabric servers
- Configuration information for fabric storage
- Configuration information for SAN switches, inter-switch links, and switch ports
- SAN topology maps

Some examples from a typical SAN report are shown in the following figures:

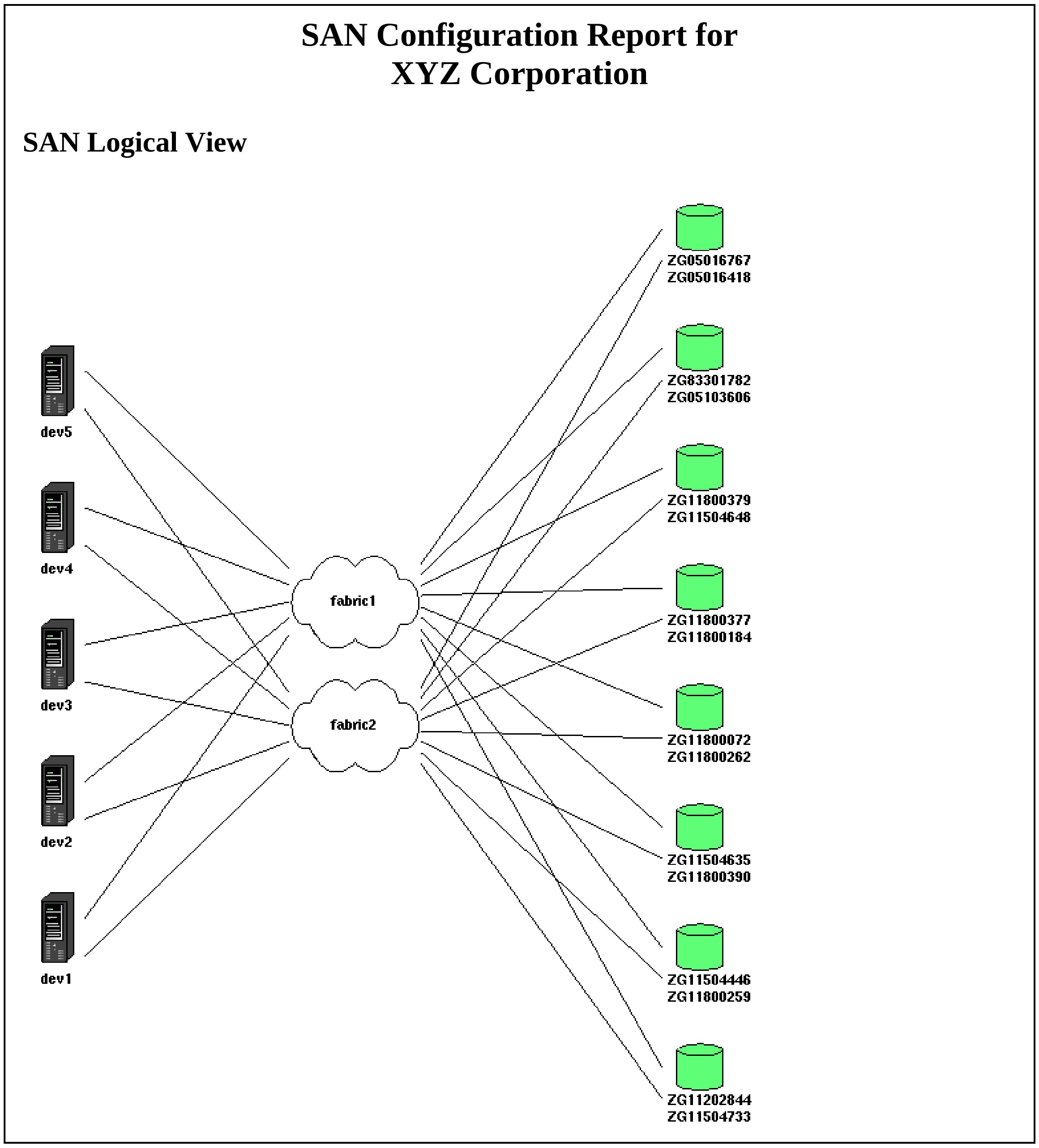
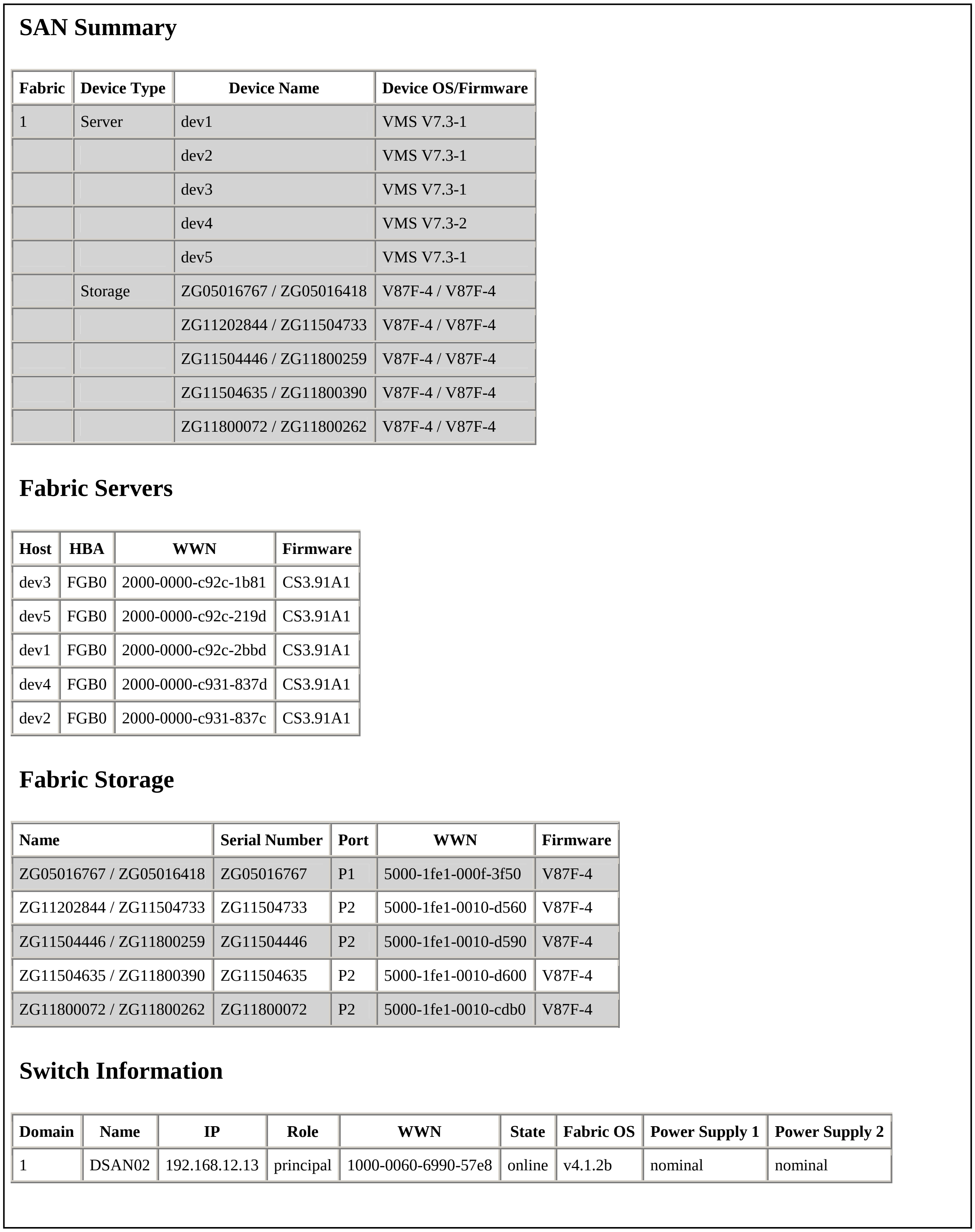


**Figure 7 Part of a SAN Report**



## SAN Summary

| Fabric | Device Type | Device Name | Device OS/Firmware |
|---|---|---|---|
| 1 | Server | dev1 | VMS V7.3-1 |
| | | dev2 | VMS V7.3-1 |
| | | dev3 | VMS V7.3-1 |
| | | dev4 | VMS V7.3-2 |
| | | dev5 | VMS V7.3-1 |
| | Storage | ZG05016767 / ZG05016418 | V87F-4 / V87F-4 |
| | | ZG11202844 / ZG11504733 | V87F-4 / V87F-4 |
| | | ZG11504446 / ZG11800259 | V87F-4 / V87F-4 |
| | | ZG11504635 / ZG11800390 | V87F-4 / V87F-4 |
| | | ZG11800072 / ZG11800262 | V87F-4 / V87F-4 |

## Fabric Servers

| Host | HBA | WWN | Firmware |
|---|---|---|---|
| dev3 | FGB0 | 2000-0000-c92c-1b81 | CS3.91A1 |
| dev5 | FGB0 | 2000-0000-c92c-219d | CS3.91A1 |
| dev1 | FGB0 | 2000-0000-c92c-2bbd | CS3.91A1 |
| dev4 | FGB0 | 2000-0000-c931-837d | CS3.91A1 |
| dev2 | FGB0 | 2000-0000-c931-837c | CS3.91A1 |

## Fabric Storage

| Name | Serial Number | Port | WWN | Firmware |
|---|---|---|---|---|
| ZG05016767 / ZG05016418 | ZG05016767 | P1 | 5000-1fe1-000f-3f50 | V87F-4 |
| ZG11202844 / ZG11504733 | ZG11504733 | P2 | 5000-1fe1-0010-d560 | V87F-4 |
| ZG11504446 / ZG11800259 | ZG11504446 | P2 | 5000-1fe1-0010-d590 | V87F-4 |
| ZG11504635 / ZG11800390 | ZG11504635 | P2 | 5000-1fe1-0010-d600 | V87F-4 |
| ZG11800072 / ZG11800262 | ZG11800072 | P2 | 5000-1fe1-0010-cdb0 | V87F-4 |

## Switch Information

| Domain | Name | IP | Role | WWN | State | Fabric OS | Power Supply 1 | Power Supply 2 |
|---|---|---|---|---|---|---|---|---|
| 1 | DSAN02 | 192.168.12.13 | principal | 1000-0060-6990-57e8 | online | v4.1.2b | nominal | nominal |

**Figure 8 Additional Details from a SAN Report**

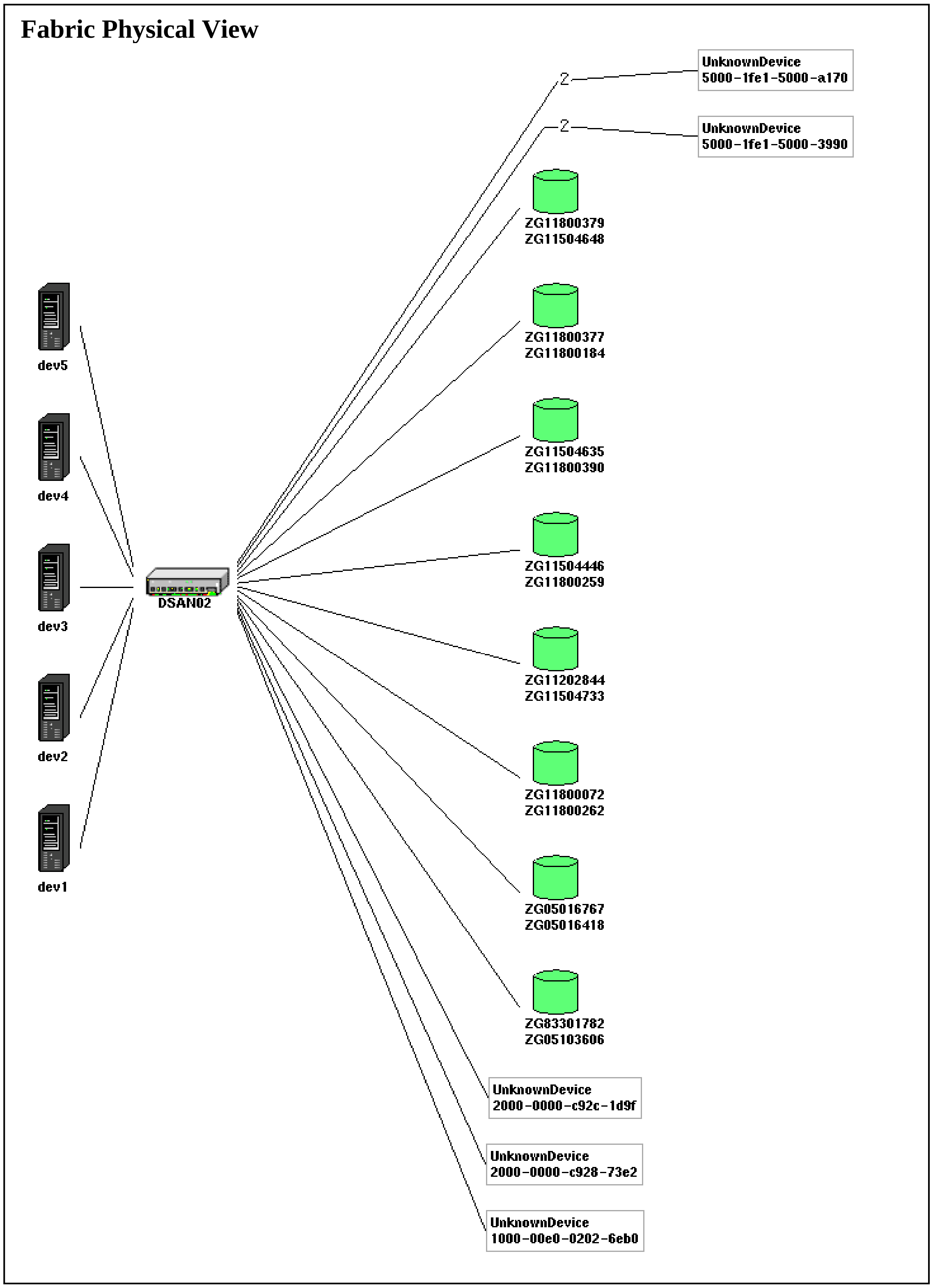


**Figure 9 Physical FABRIC Diagram from a SAN Report**

## Summary

RCM is an extremely useful tool for mission-critical customers and HP technical account managers in managing OpenVMS systems. The automated collection process ensures that the configuration information in eSMG is always up-to-date. RCM helps diagnose problems that can arise due to configuration changes, and it maintains a historical record of all changes to a system.

## For More Information

The RCM OpenVMS collector kits and documentation, as well as the HP-UX, Tru64 Unix, and Windows collector kits, can be freely downloaded from the HP Software Depot at:

http://www.software.hp.com/portal/swdepot/displayProductInfo.do?productNumber=RCMBASE01

VAX and Alpha RCM kits are separate. Each kit is a self-extracting OpenVMS executable file that unpacks into a PCSI kit. You can also unpack the kits using the OpenVMS UNZIP utility.

Information about Electronic Site Management Guide is available from:

http://www.hp.com/go/esmg

WEBES and DECevent kits are available at:

http://h18000.www1.hp.com/support/svctools/

# OpenVMS Technical Journal

## Setting Up a High-Available E-mail Server Using OpenVMS

Andreas Fassl, Senior Consultant

### Abstract

The OpenVMS Cluster technology provides unique features in the standard distribution of the operating system. One can very easily create highly available, highly scalable, and highly secure configurations.

This article shows how to set up an electronic mail system to meet the multifarious needs of companies, starting with small business and ending in typical enterprise solutions.

### Prerequisite

While knowledge of OpenVMS is a plus, this article's intention points to a broad audience.

The reading of Keith Parris's article, *Using OpenVMS Clusters for Disaster Tolerance* (http://h71000.www7.hp.com/openvms/journal/v1/disastertol.htm), is a must for a deeper understanding of the capabilities of OpenVMS.

If you're not familiar with the IP world, you'll find a very comprehensive introduction in *TCP/IP Services for OpenVMS Concepts and Planning* (http://h71000.www7.hp.com/doc/73FINAL/6523/6523pro.html).

A good, hands-on practice book, *Linux and OpenVMS Interoperability,* by John Robert Wisniewski, is available from Digital Press.

### Importance of Email

Electronic Mail has changed to one of the most important forms of communication. Everyone who had to turn down using email half a day did experience this the hard way.

Most smaller email server installations do meet the need of their operators, but with ascending demands even the smallest risk has to be avoided.

### Failures and Counter Measures

To analyze your needs you can use a matrix of the possible failures to your setup and the available counter measures. Table 1 shows possible failures and counter measures.

| **Failure of…** | **Single Node Counter Measure** | **Cluster Counter Measure** |
|---|---|---|
| System disk | Shadow system disk | Same or cluster shared system disk |
| Storage controller | Multipath storage | Multipath storage |
| Network interconnect | Second production LAN | Same |
| Node | - | Cluster IP |
| Power | UPS | UPS |
| Internet connection | Second/third ISP | Second/Third ISP |

**Table 1: Possible Failures and Counter Measures**

Many more failures are possible; the mileage of your experience shows it.

Many vendors do provide counter measures mentioned above, but many of them are very costly and often you have to do a complete redesign combined with a major migration project to scale up your environment.

## IP Services

Email is nowadays mostly based on TCPIP. Some of the other protocols are more reliable, aren't that vulnerable against misuse, but the decision has been made in favor of an easy to configure protocol. Another disadvantage of the more secure protocols has been their proprietary design and more restricted configuration rules.

The IP protocol suite has a couple of services. The services needed for an email server are:

### SMTP (Simple Mail Transfer Protocol)

Usually tied to the port 25 of a given IP host this protocol is the core for most of all mail transfers. Being created in the "dark ages" of electronic communication, people suffer under the design flaws made in the past. The most "popular" suffering is called SPAM.

Only a few commands are needed to create an email  (to illustrate the use of SMTP):

```
VMAL06 $ telnet /port=25 vmcl02.progis.net
%TELNET-I-TRYING, Trying ... 194.49.54.3
%TELNET-I-SESSION, Session 01, host vmcl02.progis.de, port 25
220 vmal06.progis.de V5.3-18E, OpenVMS V7.3 Alpha ready at Sun,
11 May 2003 18:03:09 +0200 (MET DST)
hello
250 vmal06.progis.de Hello vmcl02, pleased to meet you, friend
mail from:afassl@progis.net
250 <afassl@progis.net>... Sender OK
rcpt to:afassl@progis.net
250 <afassl@progis.net>... Recipient OK
data
354 Start mail input; end with <CRLF>.<CRLF>
```

```
Short Message
.
250 OK
quit
221 vmal06.progis.de Service closing transmission channel
%TELNET-S-REMCLOSED, Remote connection closed
-TELNET-I-SESSION, Session 01, host vmcl02.progis.de, port 25
```

Due to this easiness, SMTP had to be extended many times to fight misuse.

Be very careful while setting up an SMTP service attached to the internet – an improper configuration will make your (or your company's) mail server to another SPAM provider. There are many automatic robots scanning around to find active SMTP-ports and probing them to determine their usability for spammers' needs.

Spammers are very creative in their work. A new way observed recently uses an improper proxy configuration of a web server to use a local web server as a relay to a local mail server. The local mail server will accept mail this way because it has been identified the local web server as a trusted source.

The "guest" will leave marks like this:

```
64.70.45.253 - - [13/Apr/2003:07:47:43 +0200] "CONNECT
64.12.138.89:25 HTTP/1.0" 200 8959 "-" "-"
```

This is just an example to remind you - NEVER believe your system (even if it's OpenVMS) is 100% secure.

Keep in mind – OpenVMS engineering uses a very secure-wise approach while designing a new piece of software. When using IP based protocols it always implies inheriting security problems.

A more detailed discussion about this topic you'll find in the previous mentioned book "Linux & OpenVMS Interoperability".

Read more about SMTP on OpenVMS, see *HP TCP/IP Services for OpenVMS Management*(http://h71000.www7.hp.com/doc/73final/6526/6526pro_030.html#smtp_chap)

If you need more features and a very extensive solution, I'll recommend Madgoat Software's MX package. It isn't very expensive; more information can be found at:

http://www.madgoat.com/mx053.html

## POP3 (Post Office Protocol Version 3)

Via POP3, Email clients connect to a server supporting this protocol on port 110. The Emails are downloaded to the client and can optionally be deleted on the server after download. Usually POP3 is used for clients who are not online all the time.

Read more about POP3 on OpenVMS, see the *HP TCP/IP Services for OpenVMS Management* manual:

http://h71000.www7.hp.com/doc/732final/6526/6526pro_041.html#pop_chap

## IMAP4 (Internet Message Access Protocol Version 4)

IMAP is the most sophisticated non-proprietary Email protocol. IMAP4 (using port 143) is a client/server application offering lots of features to govern Email. The protocol has been implemented in TCP/IP-Services for OpenVMS.

Read more about IMAP4 on OpenVMS:

http://h71000.www7.hp.com/doc/732final/6526/6526pro_043.html

## BIND (Berkeley Internet Name Domain)

The Domain Name System (DNS) is a system that maintains and distributes information about Internet hosts. DNS consists of several databases that store host names and host IP addresses. With DNS, there is no central storage of data -- no one server knows everything about all the Internet domains.

TCP/IP-Services Version 5.4 supports Version 9.2.1 of the BIND protocol.

## DNS Cluster Load broker

Unique to OpenVMS – the load broker. Other system vendors have to refer to costly hardware based load balancing (CISCO arrowhead, Nortel alteon, etc.) with the disadvantage of another system in the chain between client and server. The load broker gets a key role when scaling up system.

Read more about DNS Load Balancing on OpenVMS, see "Using DNS to Balance Work Load" in the *HP TCP/IP Services for OpenVMS Management* manual:

http://h71000.www7.hp.com/doc/732final/6526/6526pro_016.html

## Some Thoughts About…Firewalls

Whilst planning an internet access structure, one of the first claims is a firewall. In several discussions I got a different view to the benefit of firewalls.

- Most attacks are done by insiders.
- Most successful attacks are using (unknown) flaws within the application (buffer overflow e.g.)

In other words – it is verisimilar a firewall is defenseless. You never are discharged from security liabilities just by setting up a firewall. Actually a good secure application server without a firewall is even superior to an unsecured application server protected by a perfect firewall.

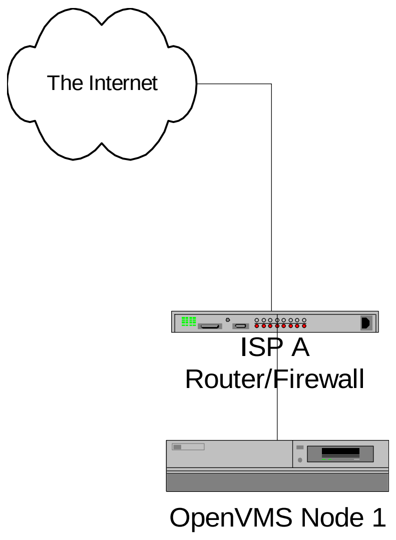
With a simple accounting/summary you'll see, for example, the trails of the attempts to misuse an ftp server (users like: admin, ftproot, informix, linux, lizdy, oracle, pwrchute, rethat, suese, sybase, win2000,etc.)

## Hardware Configuration

### Single node

We'll start with the smallest configuration, the router will be maintained by your ISP (Internet Service Provider), probably you want to add your own router to separate more networks, but this is the minimum.

**Figure 1: Single Node Configuration**



To increase availability, one can configure Volume Shadowing for the system and the data disks. This will minimize the risk of failing disks. For more information, see HP *OpenVMS Volume Shadowing*,

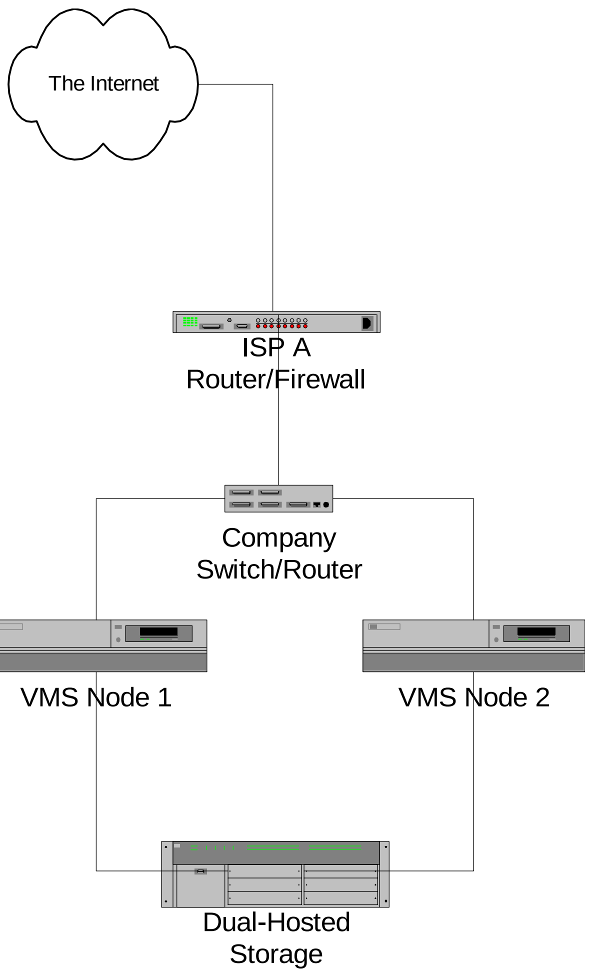
http://h71000.www7.hp.com/doc/732FINAL/aa-pvxmj-te/aa-pvxmj-te.HTML

The configuration consists of a simple OpenVMS installation using a plain SMTP/IMAP4/POP3 setup.

## Dual Node

A dual node configuration will reduce the risk of a failing system.

**Figure 2: Dual Node Configuration**



The configuration uses OpenVMS Cluster Technology, but you have to consider some additional requirements.

### Cluster Shared Environment

Most resources can be shared in an OpenVMS Cluster environment. For more information, see "Preparing a Shared Environment" in *HP OpenVMS Cluster Systems,*

http://h71000.www7.hp.com/doc/731FINAL/4477/4477pro_006.html#startup_ch

This must be done for all user data.

## Quorum Disk

To ensure the clusters integrity, especially in a small cluster setup, you have to configure a quorum disk. For more information, see "OpenVMS Cluster Concepts" in *HP OpenVMS Cluster Systems,* (http://h71000.www7.hp.com/doc/731FINAL/4477/4477pro_002.html#connection_management)

The quorum disk must be accessible for both members directly. Reliable solutions are based on HSx-controller technology with the corresponding interface cards in the nodes. Please **never** use unsupported configurations. The components have a higher price with a reason. If you're lucky, it will work. If not, no one will (and can) help you. Don't think about standards when talking about SCSI or Fibre Channel, there are so many proprietary enhancements. This is unavoidable to implement the features customers want like improved monitoring, fault detection, etc. The standards don't include those requirements, so every company has to develop and verify their own supported configuration for a comparable small market. This means lower trade volumes, higher prices.

## Configuration of the IP Services

All the TCP/IP services do support OpenVMS Clustering. For more information, see "Configuring and Managing BIND Version 9 in *HP TCP/IP Services for OpenVMS Management,*

http://h71000.www7.hp.com/doc/73final/6526/6526pro_008.html#config_cluster

In this small configuration a simple cluster IP-address should be sufficient.

Both nodes have a node-specific IP-address and a cluster-wide IP-address:

```
VMAL06 $ mc sysman
SYSMAN> set env/clus
SYSMAN> do tcpip show inter/clus
%SYSMAN-I-OUTPUT, command execution on node VMAL05
                                                          Packets
Interface   IP_Addr         Network mask          Receive          Send     MTU

 LO0        127.0.0.1       255.0.0.0                 107           107    4096
 WE0        194.49.54.1     255.255.255.0          324377        203105    1500
   Cluster  194.49.54.3     255.255.255.0

%SYSMAN-I-OUTPUT, command execution on node VMAL06
                                                          Packets
Interface   IP_Addr         Network mask          Receive          Send     MTU

 LO0        127.0.0.1       255.0.0.0                2003          2003    4096
 WE2        194.49.54.2     255.255.255.0         1210418        988863    1500
      Cluster  194.49.54.3     255.255.255.0   Impersonator
```

If a client program connects to a cluster address, it will get typically (due to certain limitations in the implementation) only one node.

At the latest from this stage on OpenVMS will show it's power. You can take over the complete configuration of the single node solution as it is and you only need to move it in a cluster shared directory.

A single line in your SMTP Startup (for the TCPIP Services SMTP)

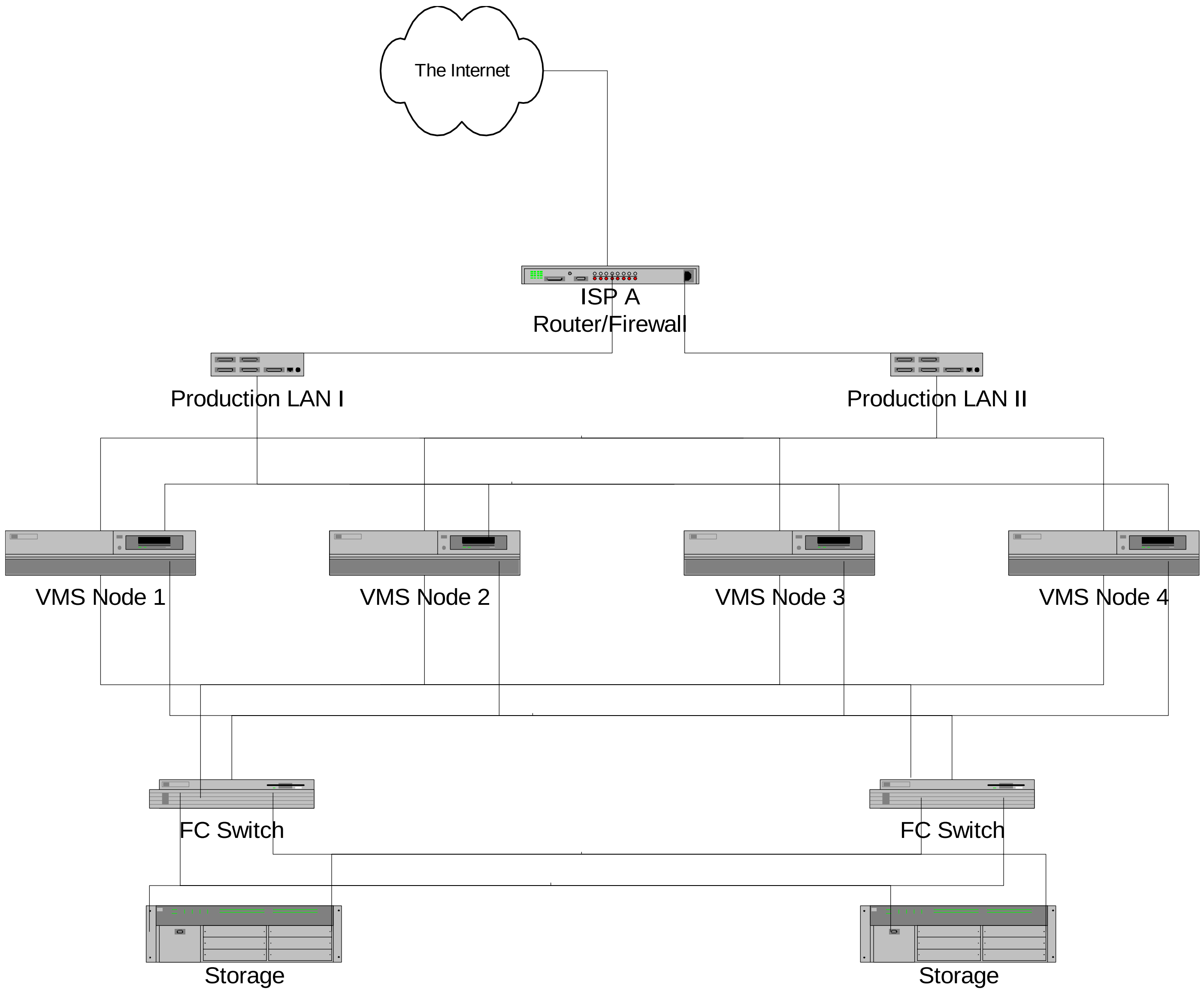
$ DEFINE/SYSTEM TCPIP$SMTP_COMMON cluster_device:<clusterdir>

represents the major changes to your configuration.

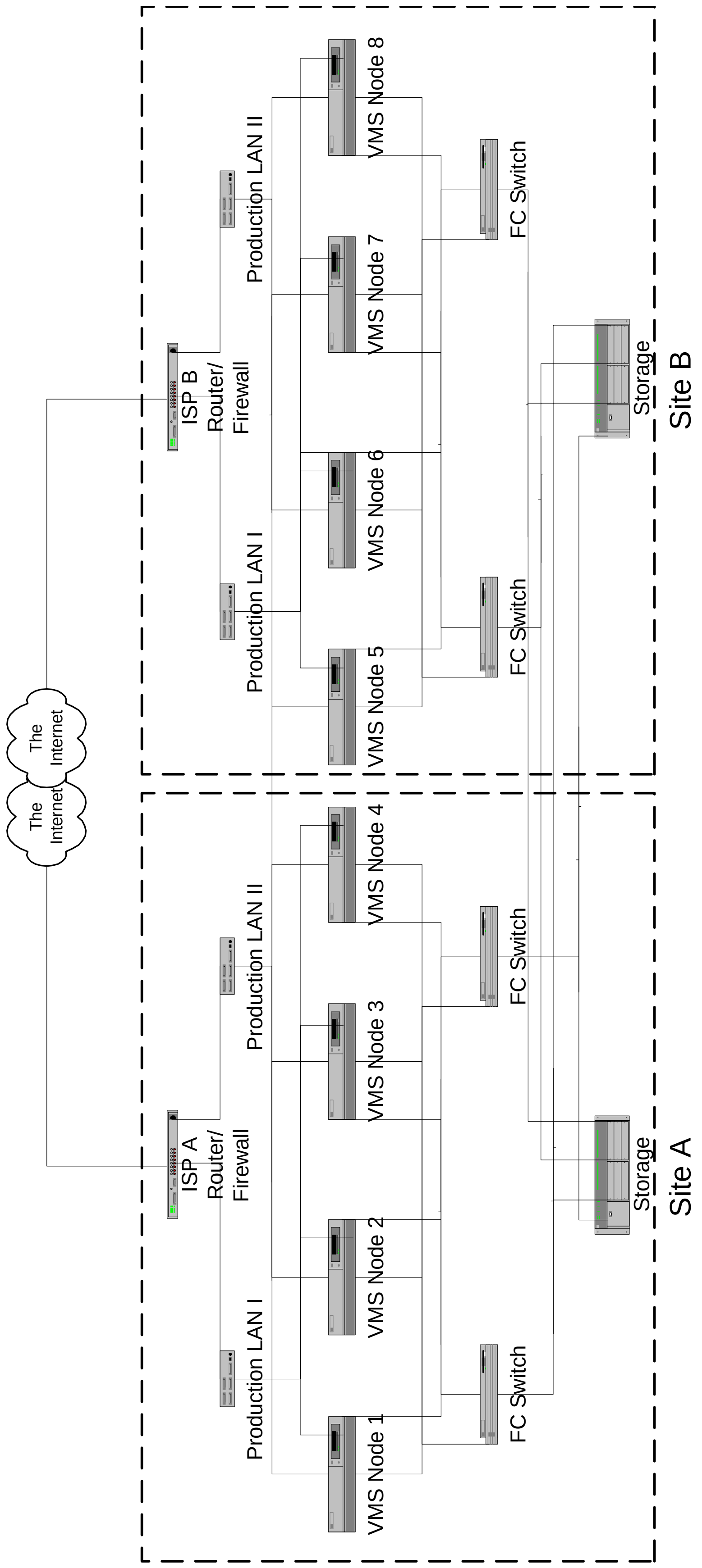
### Multi Node

The next step is to integrate more nodes to balance the load among the members. As you can see, this setup has an obvious flaw – the contact to the Internet.

**Figure 3: Multi Node Configuration**



To eliminate this bottleneck, you have to choose the next complex solution.


The Internet
The Internet
ISP A Router/ Firewall
Production LAN I
Production LAN II
VMS Node 1
VMS Node 2
VMS Node 3
VMS Node 4
FC Switch
FC Switch
Storage
Site A
ISP B Router/ Firewall
Production LAN I
Production LAN II
VMS Node 5
VMS Node 6
VMS Node 7
VMS Node 8
FC Switch
FC Switch
Storage
Site B

To be honest, this is a setup needing a very careful engineering, but it is possible with justifiable effort. The most expensive part of this configuration are fibre channel links that have to configured, providing possible line lengths up to 100 kilometers per link, using single-mode fiber and up to 600 kilometers per link with FC/ATM links.

And – most important – you don't need to reengineer your previous setup – you can use all the setup, all the user data, all the files without any changes.

## Benchmarking

How does one proof the proper configuration of his environment? There are many benchmarks available, I recommend the mstone benchmark developed by Netscape now available as an open source project.

The former Netscape Server development had a customer oriented stress test called Mstone.

Mstone is a multi-protocol stress and performance measurement tool. Mstone can test multiple protocols simultaneously and measures the performance of every transaction. The performance can be graphed throughout the duration of the test.

### Where To Get Mstone

The complete source tree had been given to mozilla.org.

Surf to: http://www.mozilla.org/projects/mstone/

### Supported Operation Systems

Mstone currently runs on recent versions of: Linux, Solaris, AIX, OSF, HPUX, and NT. Any OS with POSIX threads support should be an easy port. The test client machines can each be running different operating systems. Common utilities like perl, gnuplot, and gd are used and can be packaged with mstone for completeness.

### Installation

The easieast way to install mstone is to use a linux box. In a later step we will provide a OpenVMS port of mstone.

### Setting CVSROOT

```
$ CVSROOT=":pserver:anonymous@cvs-mirror.mozilla.org:/cvsroot"
$ export CVSROOT
```

The password for user anonymous is anonymous.

### Getting the Source

```
$  cvs co mozilla/mstone
cvs server: Updating mozilla/mstone
U mozilla/mstone/Building
```

```
$ cd mozilla/mstone
$ gmake rpackage
```

You should now have a complete runnable tree under a platform-specific directory under build/package.

### Initial Configuration

Run "mstone config."  It will ask you about your system configuration. Fill in the appropriate values and create the optional user accounts and broadcast account.  When it asks about client machines, enter them seperated by commas, with no spaces (e.g. host1,host2,host3).  If you need to re-configure, run "mstone config".

The machine starting the test may also be a client.  For accurate results, clients should not be run on the test mailserver machine (or its directory server).  If all the client machines are not running the same operating system version, see "Configuring Client Machines" below to configure for different OSes.

Setup only configures the most important parameters.  If you have more advanced needs, edit conf/general.wld appropriately.

Run "mstone setup".  It will now push the necessary files to each client machine.  If there are problems (i.e. with rsh permissions), fix them and re-run "mstone setup" until everything works.

### Install Test Accounts

With this small DCL-program you can create for example 100 users.

```
$ ! DCL-Script to generate MSTONE Test-Accounts
$ !
$ !
$ ! Start User Number
$ NUM=1
$ ! Start Group
$ START_GROUP=%o4000*%o200000
$ loop:
$ ! Create an username based on num
$ USERNAME="MST_''num'"
$ ! Calculate the UIC
$ uic = f$fao("!%I",START_GROUP + NUM)
$ write sys$output "Creating User : ''USERNAME'"
$ mc authorize add /UIC='uic'
'USERNAME'/owner=mstone/account=MBench/
dev=$100$dka0:/dir=[mstone.accounts.'username']
$ create/dir/owner='username' $100$dka0:[mstone.accounts.'username']
$ num=num+1
$ if num.lt.100 then goto loop
$ exit
```

And with this one you can dump them

```
   $ ! DCL-Script to remove MSTONE Test-Accounts
   $ !
   $ !
   $ ! Start User Number
   $ NUM=1
   $ ! Start Group
   $ START_GROUP=%o4000*%o200000
   $ loop:
   $ ! Create an username based on num
   $  USERNAME="MST_''num'"
   $ ! Calculate the UIC
   $ uic = f$fao("!%I",START_GROUP + NUM)
   $ write sys$output "del User : ''USERNAME'"
   $ mc authorize rem  'USERNAME'
   $ num=num+1
   $ if num.lt.100 then goto loop
   $ exit
```

If you want to, you can do some more programming to be more flexible with your test setup, for the start this will be enough.

## Run Tests

Try it out. Use small process and thread counts until everything is working.

```
   mstone smtp -t 30s
```

The script will tell you how many processes and threads it is running on each system and where errors are logged. At the end of the test, it will print out a URL for the test results and an indication of the size of the errorlog file (stderr).

The results of the mstone run will display statistics for each protocol that was tested. The results are presented in both a HTML web page and a text file. The text file is simple and uniform, while the web page is more user readable. The web page has links to the test configuration files, error log, and the text version.

## Resources to watch

With this test suite it is possible to simulate all thinkable load profiles. So can first watch and tune for a single user with one message size, mixed message sizes, ascending user accounts, etc.

**What you have to watch:**

- TCPIP-Ressources like per process memory
- IO-Saturation
- CPU-Load

You can use the basic tools like MONITOR, switching over to more sophisticated tools like the „Availability Manager" or for more professional (but costly) analysis use products like CAs performance advisor (formerly known as Polycenter Performance Advisor). The newer (and free) ECP (Enterprise Capacity Planner) should be worth a look as well.

## Customize tests

Copy and edit the scripts (e.g. "conf/pop.wld") to define new tests. The CONFIG section specifies all the attributes used in the test.Other sections specify the protocols to be tested and the parameters for them.

All switches can be overridden on the command line to facilitate easier testing. The exact configuration (include command line overrides) is stored with the results from each test.

## Maintenance

You can run "mstone setup" at any time (except during a test :-) to update the files on the client machines.

Use "mstone cleanup" to remove the files created by "mstone setup". After the test is finished, the directories under "tmp/" can be compressed or deleted to save space. All the information about a test run is stored in the "results/" directories.

## Configuring Client Machines

Edit conf/general.wld to include CLIENT sections for each machines to use. You can also specify the OS type for each client machine. Set the "Arch" parameter in each CLIENT section as appropriate (e.g. SunOS5.6, Linux2.2_x86, AIX4.2, HP-UXB.11.00, IRIX6.5, OSF1V4.0, WINNT4.0). The directories under "bin" specify the available OS types. For NT4.0 clients with a UNIX test master, you will need to configure "command" and "tempDir" for proper operation. See the "HOSTS=winnt01" example in conf/sample.wld.

The total number of processes and threads that can be supported on a client is dependent on the number of commands in the test, the OS, and available memory. Check the stderr log for messages about not being able to create processes or threads. Check on the client machines during the test and make sure they aren't running out of CPU. The UNIX programs "top" and "vmstat" are good for this. If the client CPU is more than 75% busy, use more machines. Also watch out for network saturation. You may have to use machines with separate networks to the server to reach full server load.

## Summary

**The discussed configurations aren't very complicated. And they are:**

- Easy to design
- Easy to implement
- Easy to maintain

If you start your configuration with the smallest cluster configuration, the biggest advantage you'll get is an environment, that will have no down time, especially for:

- OS Upgrades
- HW-Upgrade of single nodes
- Adding additional nodes
- Doing site movements

These are the great demands companies have on IT solutions. OpenVMS meets the demands, and more ...

This configuration can be used for many other IP-based services.

To name only a few of them:

- NTP (for high reliable time stamps)
- NFS (as a 100% available UNIX file server)
- SMB (as a 100% available windows file server)

## For more information

Contact:

ProGIS Software & Beratung

Dipl.Ing. Andreas Fassl

Spichernstraβe 59

50672 Köln

Deutschland

http://www.progis.de
mailto:afassl@progis.de

OpenVMS Technical Journal

# Including OpenVMS Oracle and Oracle Rdb database servers in a heterogeneous backup solution.

Shaun Ellis - Product Manager for LEGATO NetWorker on OpenVMS

## Overview

Many companies enjoy the benefits of OpenVMS, such as its industry leading clustering capabilities and extremely high reliability, and continue to use it for mission critical applications. These applications are often based on Oracle DBMS, or Oracle Rdb. However, OpenVMS has often not been included in management suites. An example of such a limitation of OpenVMS in heterogeneous systems environments has been the lack of integrated backup and archival solutions, increasing Total Cost of Ownership for OpenVMS and, thus, reducing its perceived value. Until recently, no single solution has been available which could simultaneously meet the backup demands of file systems and databases for OpenVMS, Windows, UNIX and other Open system platforms

The LEGATO OpenVMS products enable OpenVMS systems, and their databases, to be backed up across a LAN, or SAN, or to locally attached storage while sharing resources with other operating systems and thus increasing Return On Investment, and reducing Total Cost of Ownership. Regular backup and archival activities are scheduled and initiated by a central NetWorker Server. Users of OpenVMS systems can schedule ad hoc backups or recoveries of individual files, directories or disks as necessary LEGATO has taken great care to ensure that its OpenVMS products support important features of OpenVMS including the unique capabilities of the ODS-2 and ODS-5 file systems, OpenVMS clusters, and OpenVMS shelving. DBA's can perform backups of Oracle and Rdb databases using the backup tools standard to these databases, RMAN and RMU, so that on-line backups can be performed, and with minimal changes to any existing scripts. These backups can be run manually, scheduled locally on the OpenVMS system, or centrally managed from the NetWorker server. This is all achieved by a product that looks and feels the same, regardless of platform, reducing learning time for System Managers, Operators, and users. This paper discusses the integration of OpenVMS systems in a NetWorker environment.

## Limitations of Backup for OpenVMS

Thousands of customers around the globe depend on OpenVMS to run critical business and scientific applications. It often runs "bet your business" applications, where downtime is intolerable. However, despite the wide use of OpenVMS, it has been difficult to integrate OpenVMS with Enterprise backup solutions that include UNIX, Linux, Windows, and/or other system platforms. This issue is exacerbated by the wide use of relational databases on OpenVMS, as the tools provided by the database vendors have not been integrated into any heterogeneous backup solutions.

Solutions exist that allow UNIX and Windows clients to be backed up across a local area network (LAN) to a system running OpenVMS, or that allow OpenVMS systems to be backed up, over a LAN, to UNIX, Linux, or Windows systems. However, because of the rapid growth in data, solutions that rely entirely on LAN backup are proving inadequate.

Even with the evolution of Ethernet to 1Gb speeds, the amount of data stored on many UNIX and Windows systems has simply become too great for efficient LAN backup to an OpenVMS system or cluster. Similarly, older VAX systems are being consolidated into larger and larger Alpha Server systems that, due to the amount of data, now become poorly suited for network backup. Compound that with increasingly large databases, that can never be taken off-line, we have a result that many sites have been faced with the necessity of using one solution to backup OpenVMS and another for their Windows, UNIX or other systems, resulting in duplicate hardware and processes that increase management complexity and cost of ownership.

### Overcoming Limitations of Backup for OpenVMS

With the availability of the NetWorker Client for OpenVMS and NetWorker Storage Node for OpenVMS, LEGATO Software was the first storage software vendor to completely integrate OpenVMS into a heterogeneous Enterprise backup and archival solution that allows sharing of resources. Now, with the addition of the NetWorker Module for Oracle on OpenVMS, LEGATO is the first vendor to bring fully integrated relational database backup to OpenVMS in a heterogeneous backup environment.

## LEGATO NetWorker Support for OpenVMS

The NetWorker Client for OpenVMS software enables network backup of OpenVMS systems, and is appropriate for smaller OpenVMS systems. The NetWorker Storage Node for OpenVMS software allows OpenVMS systems to write data to direct-attached or SAN-attached backup devices *at least* as fast as OpenVMS Backup. The Storage Node also allows an OpenVMS system to provide backup services to network clients of any platform. In addition to traditional backup, it is often necessary, for regulatory reasons to Archive data for long periods. Both products, on Alpha and Itanium, support archival operations in addition to backup.

NetWorker on OpenVMS offers full support for critical OpenVMS features such as OpenVMS Cluster environments, co-existence with OpenVMS Shelving, and full support for the ODS-2 and ODS-5 file system including the ability to accommodate all file semantics, access controls, file types, file versions, and directory structures. In addition, both products make the full capabilities of NetWorker available on OpenVMS systems.

The Oracle module for NetWorker on OpenVMS uses the Oracle SBT V2.0 API. This means that LEGATO supports Oracle 8i, 9i, 10g, and Rdb 7.1.2.1.0 (or later) databases to be backed up, and can back them up on-line, enabling organisations to meet modern Service Level Agreements. The Module uses standard Oracle or Rdb backup scripts that do not need major modification to work. The scripts can be client, or NetWorker server initiated, with NetWorker accepting the stream of data being pushed by the database, and then writing it to the backup device.

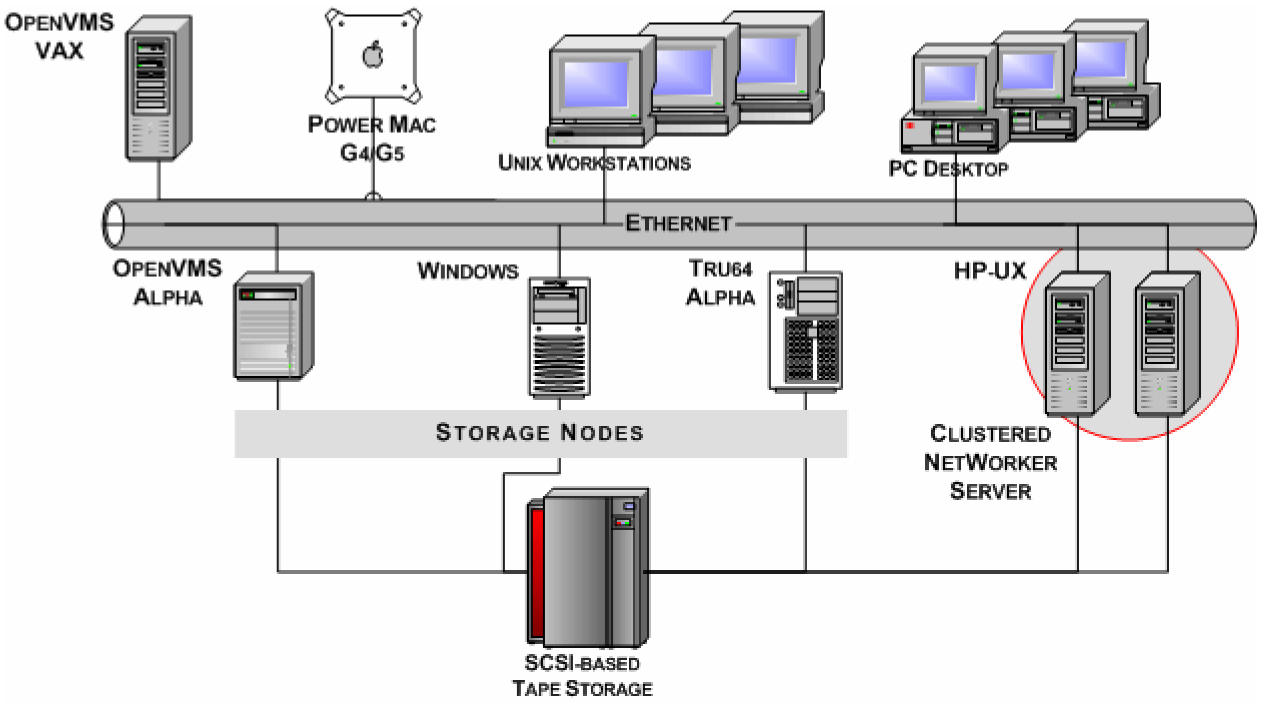


**Figure 1: A Typical NetWorker Environment, with SCSI-Based Library Sharing**

A typical NetWorker environment consists of a NetWorker Server, one or more Storage Nodes, and multiple LAN clients and associated tape libraries and/or standalone drives.[1] The NetWorker Server controls and directs all NetWorker operations, stores the Media Database, the configuration files, and also the Client File Indexes . Both the NetWorker Server and storage nodes can receive client backup data over LAN connections and write that data to an available tape, or disk. A storage node or server can also backup its own local data.

In a SAN environment, systems with a large amount of online data storage—such as database servers, file servers, and application servers—may be configured as storage nodes, allowing them to backup their data directly to SAN-attached backup hardware. SAN configurations can dramatically improve utilization of tape resources.

On the NetWorker Server, the Media Database contains information such as which savesets are stored on which tapes, and when those savesets will expire. It also manages where those tapes currently reside, and when they are available to be recycled, or over-written. The Client File Index (CFI) contains information about each file that was backed up in each saveset. A browse period can be set separately on each saveset so that individual file data can be retired whilst preserving the knowledge of the saveset. Without this capability, the CFI could become a backup issue in its own right, as it would continue to grow as long as data is kept.

The NetWorker server also performs bootstrap backups of itself so that, in the event of a catastrophic failure, the NetWorker server can be completely restored with no loss of data.

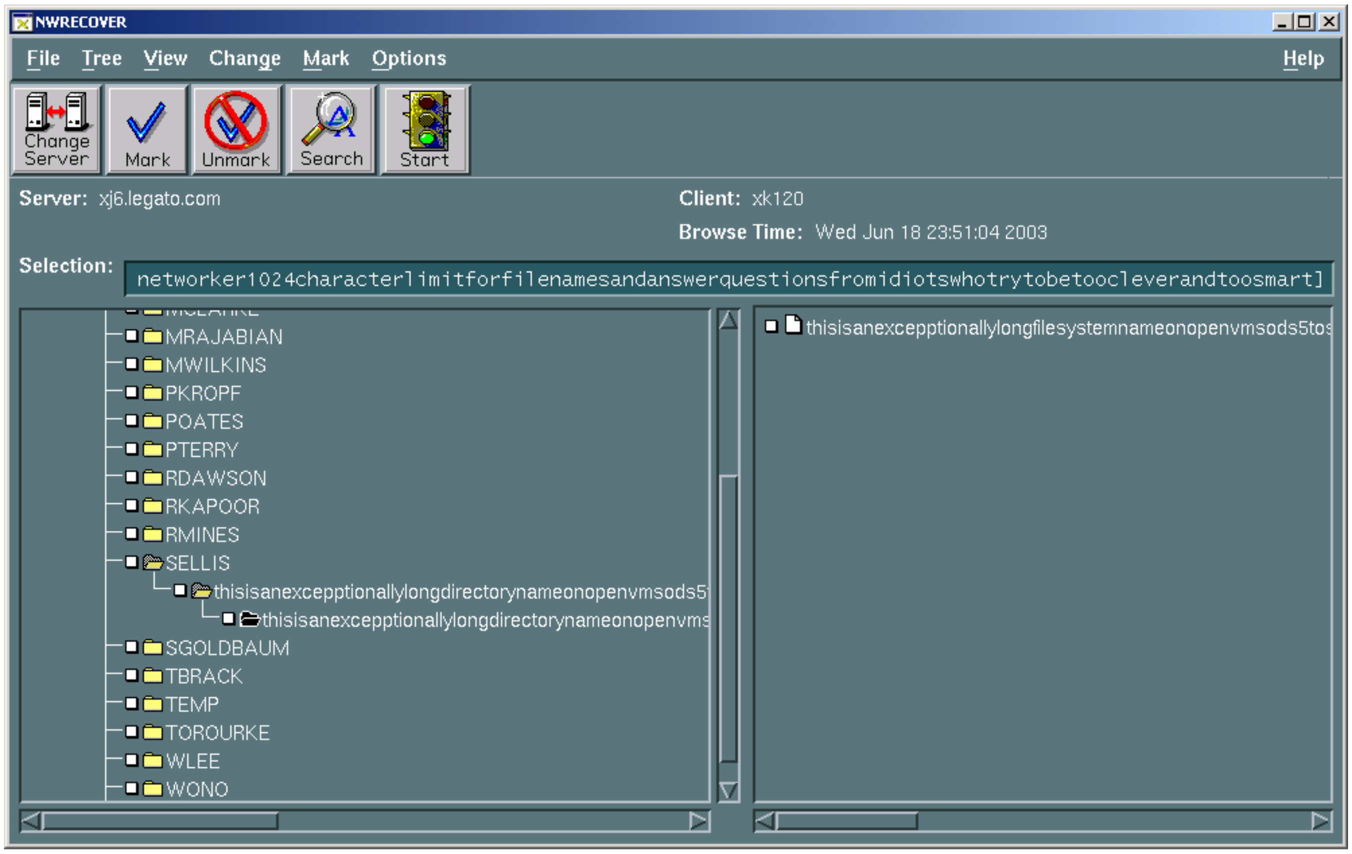
## NetWorker Client for OpenVMS

A NetWorker OpenVMS client normally operates under the control of a NetWorker Server in accordance with guidelines and schedules established by an administrator. An administrator can

[1] Note that only an OpenVMS 64-bit system can act as a client, storage node and backup and restore databases, A VAX can act as a client only. The NetWorker server is not available on OpenVMS at this time.

schedule backup and archival operations using an administrative graphical interface. Users on a client system can initiate backup and recover actions as necessary, using a separate graphical interface or through a command line interface (CLI). For instance, if an important file needs to be recovered from backup, that action can be initiated from the client without administrator intervention, providing the user has privilege to access those files. Likewise, files that are important to the user can be backed up manually without waiting for regularly scheduled backups to occur. The ability to do this is a privilege provided, by default, by the NetWorker Administrator. However, this privilege can be removed giving the NetWorker administrator complete control over when backups occur.

**Figure 2 NetWorker for OpenVMS Restore Interface Showing Very Long File Names from an ODS-5 Volume**



Server: xj6.legato.com
Client: xk120
Browse Time: Wed Jun 18 23:51:04 2003
Selection: networker1024characterlimitforfilenamesandanswerquestionsfromidiotswhotrytobetoocleverandtoosmart]

MRAJABIAN
MWILKINS
PKROPF
POATES
PTERRY
RDAWSON
RKAPOOR
RMINES
SELLIS
thisisanexcepptionallylongdirectorynameonopenvmsods5
thisisanexcepptionallylongdirectorynameonopenvms
SGOLDBAUM
TBRACK
TEMP
TOROURKE
WLEE
WONO

thisisanexcepptionallylongfilesystemnameonopenvmsods5tos

The NetWorker Client for OpenVMS also includes customisations that allow the software to backup files with all the important options associated with the OpenVMS BACKUP command, such as the ability to backup open files, recording the backup date, various options for handling file aliases, etc. (See the later section entitled *NetWorker Equivalents to OpenVMS backup Functions* for more information.)

To improve backup speed, the NetWorker Client for OpenVMS is multi-threaded to enable several backup operations to occur in parallel. Optionally, client side compression can be used to reduce LAN traffic.

The file system support is essentially the same as the native OpenVMS BACKUP utility. Data can be backed up and restored to and from ODS2, and ODS-5 volumes, and even between the file systems, where the file system rules allow.

The NetWorker Client for OpenVMS also includes the ability to perform a point in time restore. You do not have to worry about which tapes need to be restored in which order. Provide NetWorker with the date & time that you wish to restore the system to, and NetWorker will take care of it all for you.

## NetWorker Storage Node for OpenVMS

The NetWorker Storage Node for OpenVMS software allows an OpenVMS system to take on all the functions of a NetWorker storage node including receiving data from network clients and backing up local information direct to a storage device. Because of the storage capacity of many Alpha Servers running OpenVMS, backing up data locally, or to a SAN attached device is often the best solution.

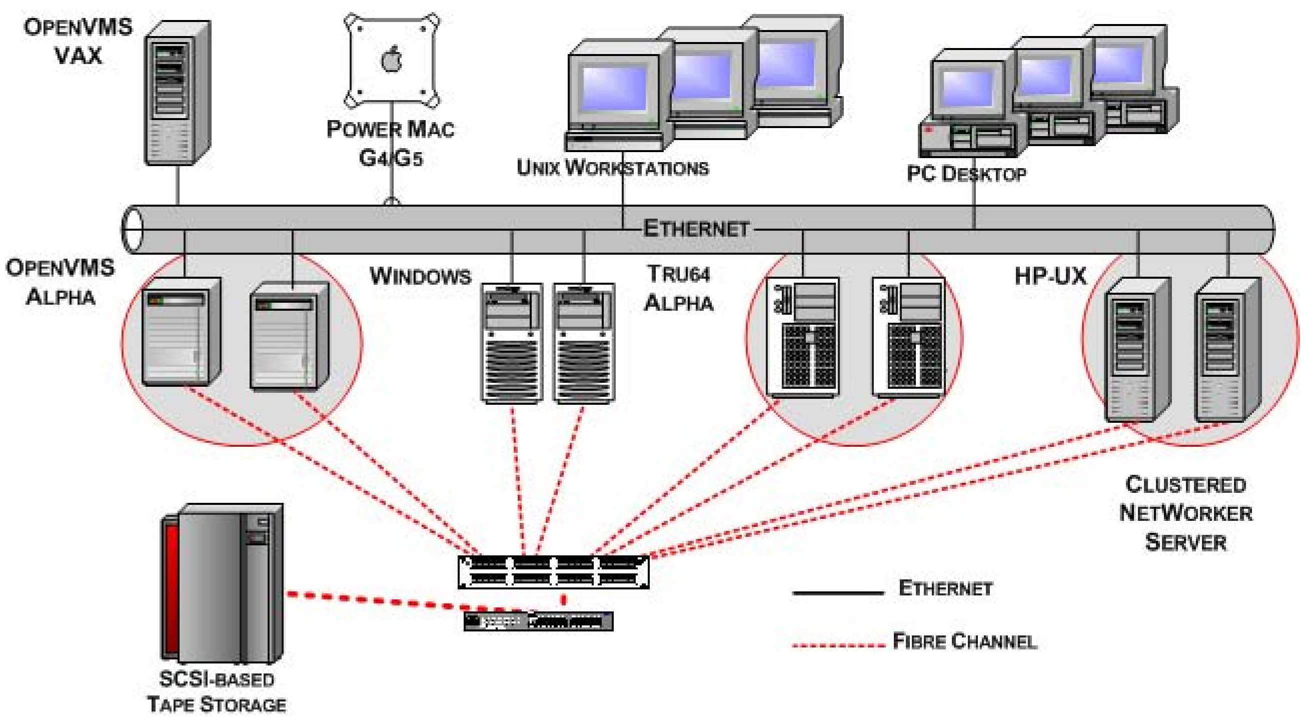
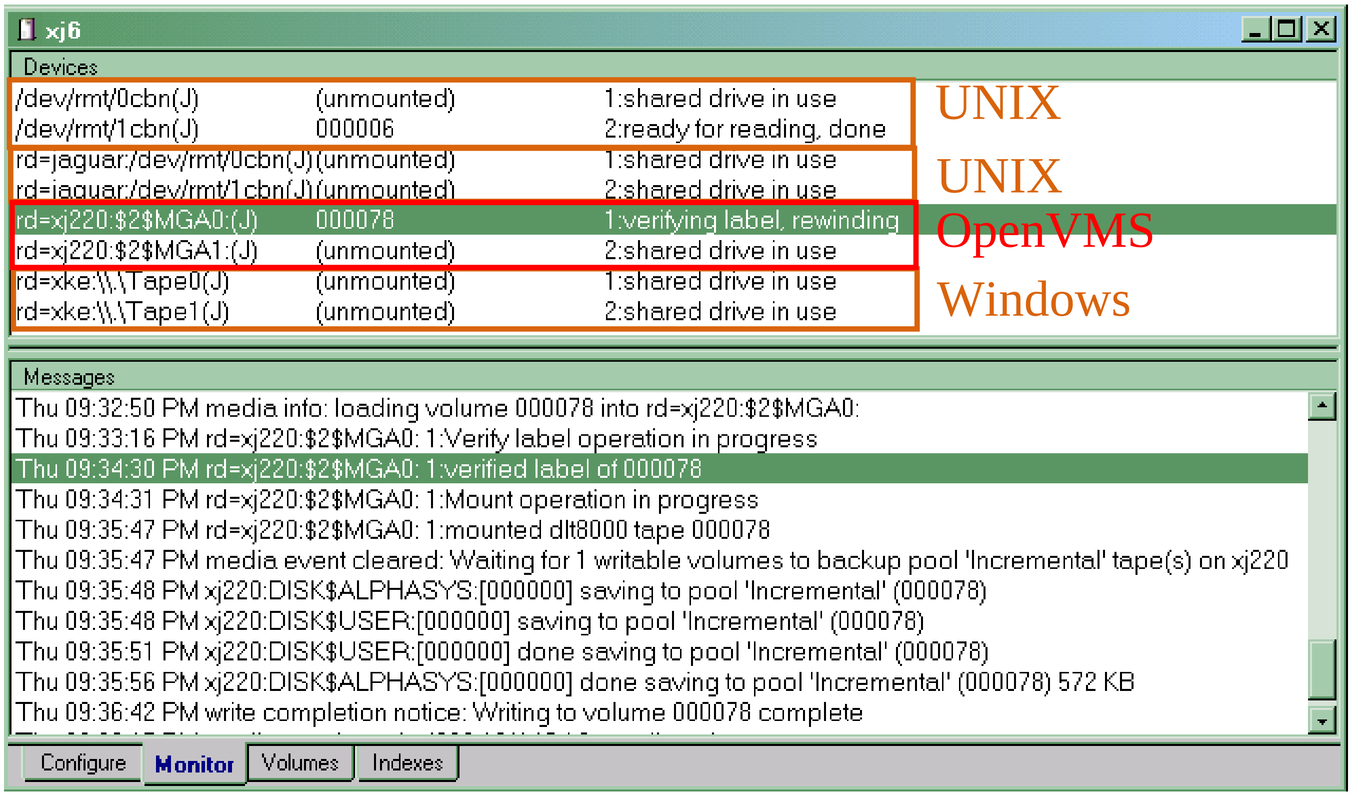


**Figure 3: Sharing tape resources on a SAN with DDS. Tape devices within a shared library are not allocated to an individual server. The NetWorker Server and each Storage Node can utilise any available tape device.**

An OpenVMS storage node has the backup, archival, and restore capabilities of an OpenVMS client as discussed in the previous section, plus the ability to direct data to direct-attached or SAN-attached storage devices. All OpenVMS SCSI and FC tape devices are supported using standard OpenVMS naming conventions, as well as OpenVMS File Systems for Disk-to-Disk backup.

An OpenVMS storage node has all the capabilities of any NetWorker V6 storage node including Dynamic Drive Sharing (DDS) (when used with a NetWorker Server running version 6.1 or later), staging, and cloning. DDS allows individual tape drives, within a tape library, connected to a fibre channel SAN to be dynamically shared between multiple storage nodes of differing operating systems and a NetWorker Server. A system initiating a backup is allocated a drive only for the time it takes for its backup activity to complete. Upon completion, the tape is unloaded, and the drive is made available for use by other hosts.

The use of SAN-based backup storage allows storage to be centralised for better protection and much more efficiently managed. For instance, a single large capacity tape library can take the place of many smaller direct-attached libraries, substantially decreasing complexity and associated management costs, while increasing the level of protection for critical data.



```
/dev/rmt/0cbn(J)              (unmounted)       1:shared drive in use
/dev/rmt/1cbn(J)              000006            2:ready for reading, done
rd=jaguar:/dev/rmt/0cbn(J)    (unmounted)       1:shared drive in use
rd=jaguar:/dev/rmt/1cbn(J)    (unmounted)       2:shared drive in use
rd=xj220:$2$MGA0:(J)          000078            1:verifying label, rewinding
rd=xj220:$2$MGA1:(J)          (unmounted)       2:shared drive in use
rd=xke:\\.\Tape0(J)           (unmounted)       1:shared drive in use
rd=xke:\\.\Tape1(J)           (unmounted)       2:shared drive in use
```

```
Thu 09:32:50 PM media info: loading volume 000078 into rd=xj220:$2$MGA0:
Thu 09:33:16 PM rd=xj220:$2$MGA0: 1:Verify label operation in progress
Thu 09:34:30 PM rd=xj220:$2$MGA0: 1:verified label of 000078
Thu 09:34:31 PM rd=xj220:$2$MGA0: 1:Mount operation in progress
Thu 09:35:47 PM rd=xj220:$2$MGA0: 1:mounted dlt8000 tape 000078
Thu 09:35:47 PM media event cleared: Waiting for 1 writable volumes to backup pool 'Incremental' tape(s) on xj220
Thu 09:35:48 PM xj220:DISK$ALPHASYS:[000000] saving to pool 'Incremental' (000078)
Thu 09:35:48 PM xj220:DISK$USER:[000000] saving to pool 'Incremental' (000078)
Thu 09:35:51 PM xj220:DISK$USER:[000000] done saving to pool 'Incremental' (000078)
Thu 09:35:56 PM xj220:DISK$ALPHASYS:[000000] done saving to pool 'Incremental' (000078) 572 KB
Thu 09:36:42 PM write completion notice: Writing to volume 000078 complete
```

**Figure 4: An actual backup of an OpenVMS system being performed to DLT8000 drives that are shared with UNIX and Windows systems.**

Storage nodes also have the option of staging backup data to disk to dramatically increase backup speed. Data thus stored can be automatically transferred to tape at a later time. The recent availability of high capacity, low cost disk backup solutions also makes permanent or semi-permanent disk storage of backup data a possibility. The NetWorker DiskBackup option allows an unlimited number of NetWorker clients and storage nodes to be backed up directly to disk.

NetWorker storage nodes also have the ability to make duplicates, or *clones,* of complete tape volumes or individual backups. Clones are frequently sent to offsite storage, used to transfer data to another location or are simply retained to enhance the level of protection for important data.

## Oracle Module for OpenVMS

Oracle 8i saw the introduction of RMAN (Recovery MANager), which brought to Oracle Database similar on-line backup functionality to Oracle Rdb's RMU (Rdb Management Utility). Both of these utilities support an API and, since Rdb 7.1.2.0.0 they both support the SBT (System Backup to Tape) API V2.0. LEGATO is the first heterogeneous vendor to support both Oracle Database and Oracle Rdb on OpenVMS using this API, via the optional NetWorker Module for Oracle. LEGATO already enjoys a close relationship with Oracle, with NetWorker single server being the exclusive backup application shipping with Oracle on every UNIX, Linux, and Windows distribution. The introduction of this optional module for NetWorker on OpenVMS extends the support of on-line backup of all Oracle databases that support the SBT V2.0 API. These databases are Oracle 8i, 9i, 10g, and Rdb 7.1.2.1.0 and later. LEGATO has worked closely with the Rdb development team to make sure that the Rdb implementation of the SBT API and NetWorker would work well together, expanding upon LEGATO's extensive experience with Oracle Databases.

Whilst it is possible to back up the files that make up a databases with the file system utilities, there can be no guarantee of recovery if the databases are actually on-line at the time. This means that databases must be closed so that reliable file system backups can be taken. For many companies, this is not acceptable. Both RMAN and RMU enable online backups of databases. The NetWorker Module for Oracle interfaces with the SBT API and provides a conduit for data to be written to backup devices by NetWorker, bringing all the advantages of NetWorker whilst maintaining a fully supported solution for Oracle Database and Oracle Rdb that is familiar to the DBA's. Indeed, the DBA's continue to use RMAN and RMU just as they do today.

Backups can be initiated by the server, or the client via a script or command line. With server initiated backups, the script is called by the server. Many of these scripts already exist today and take very little modification to work with NetWorker. For Rdb, one need only add the /LIBRARIAN qualifier in RMU backup and then define a single logical name to identify NetWorker as the custodian of the backup data. It is that simple.

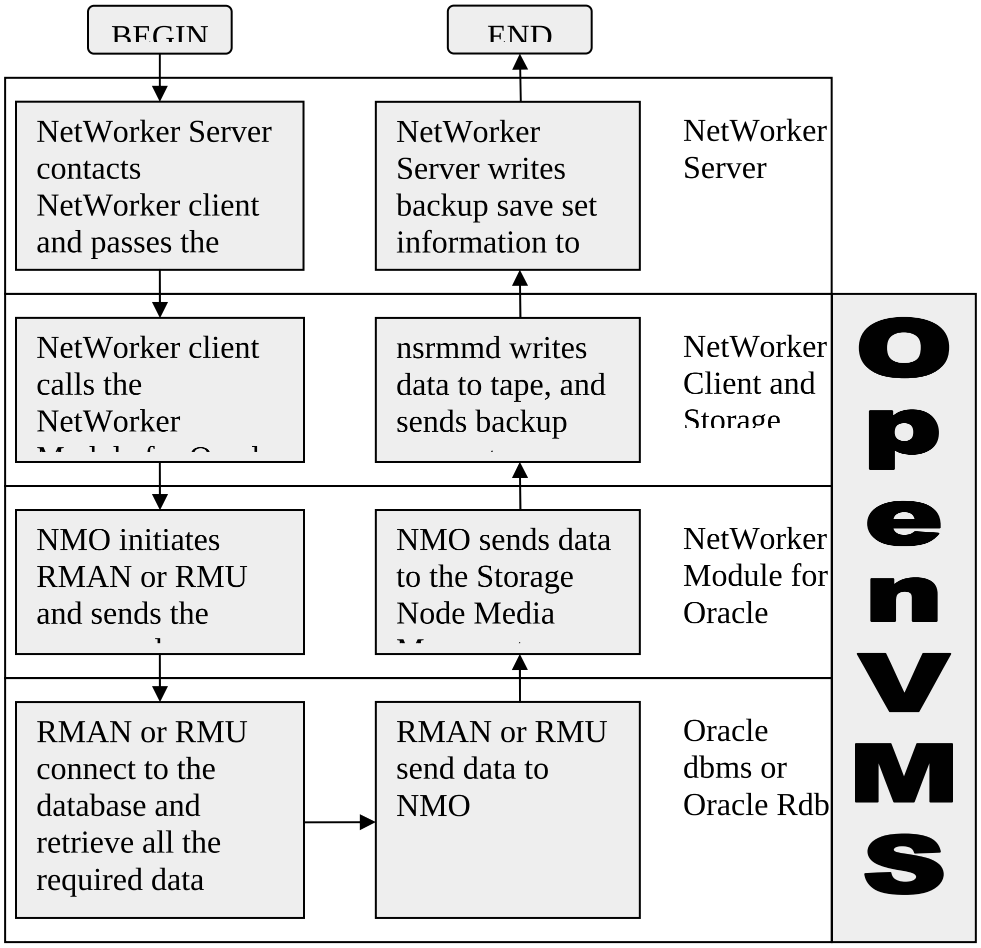

BEGIN
END
NetWorker Server contacts NetWorker client and passes the
NetWorker Server writes backup save set information to
NetWorker Server
NetWorker client calls the NetWorker
nsrmmd writes data to tape, and sends backup
NetWorker Client and Storage
NMO initiates RMAN or RMU and sends the
NMO sends data to the Storage Node Media
NetWorker Module for Oracle
RMAN or RMU connect to the database and retrieve all the required data
RMAN or RMU send data to NMO
Oracle dbms or Oracle Rdb
OpenVMS

**Figure 5: Flowchart of data flow and processes involved in a database backup.**

## Protecting an OpenVMS System Disk

The ability to quickly recover an OpenVMS system disk is critical should a disaster occur. NetWorker correctly handles OpenVMS file aliases and contiguous files, making sure that superfluous data is not backed up, and files are restored correctly.

Although NetWorker for OpenVMS does not, currently, have an equivalent to the OpenVMS standalone backup utility, there are a number of options for recovering an OpenVMS system disk in a NetWorker environment, such as:

- Create a small standby system disk (including NetWorker software) that can be booted in case of failure. NetWorker can then be used to recover backups to a replacement system disk.
- In environments with OpenVMS clusters, boot the failed system from the system disk of an operational system that has NetWorker installed. Use NetWorker to recover backups of the failed system disk to a replacement disk.

- Occasionally use OpenVMS backup to manually backup the system disk. This results in a system disk image that can be restored using standalone backup. Once this Image is restored, the disk can be brought up to date using NetWorker.

## NetWorker Equivalents to OpenVMS backup Functions

OpenVMS includes a backup utility as part of the Operating System. It has special features to deal with the specifics of the OpenVMS File System, some of the capabilities are used by default; others are enabled by NetWorker's equivalent to a qualifier, called a "directive".

The NetWorker Client for OpenVMS has the similar functionality to the OpenVMS backup utility, as well as SLS or ABS:

- **/Alias**. NetWorker automatically handles RMS File aliases correctly
- **/Incremental** and **/Image**. NetWorker performs Full, Incremental, or level backups. Level backups are not available to OpenVMS backup, but are available to users of HP's ABS. NetWorker does not have an exact equivalent to an Image backup. However, NetWorker can backup and restore an OpenVMS system disk.
- **/Modified/Since**. NetWorker can use a date/time, in OpenVMS format, to specify that files modified since a specific date should be backed up.
- **/Block Size**. By default, to maximise throughput, the NetWorker Storage Node selects the best block size for the backup device. On OpenVMS, NetWorker is limited by OpenVMS, so the maximum block size is 63K.
- **/Delete**. NetWorker provides an option to the archiving function that is equivalent to this capability.
- **/Exclude**. It is possible to exclude files with NetWorker. Depending on the interface used depends on how this function is performed.
- **/ignore=interlock**. By default, NetWorker on OpenVMS behaves the same as OpenVMS Backup as it will not backup open files. However, this can be enabled via a NetWorker for OpenVMS directive.
- **/Ignore=nobackup**. By default, NetWorker will ignore files marked nobackup, and backup the file headers only. There is a directive for the NetWorker client for OpenVMS to backup these files.
- **/list**. When a backup is performed, the list of files backed up are stored in the on-line indexes. A report can be run against the on-line indexes to find out what files were backed up. For a manual save, the files are also displayed on the input screen as they are backed up.
- **/record**. NetWorker client for OpenVMS provides a directive to have the record date stamped on files.
- **/select**. NetWorker can be directed to backup anything from the entire system, down to an individual file. It supports OpenVMS wildcards for easy election
- **/tape_expiration**. NetWorker implements much of the media management capabilities of SLS, or ABS. NetWorker can manage the on-line file indexes, and the retention policy for tapes. It does this as two separate definitions.
- **/unshelve**. NetWorker co-exists with HSM for OpenVMS. By default files will remain shelved, but it is possible to force them to be unshelved

- **/verify**. By default NetWorker performs in the same way as OpenVMS backup and does not verify the data. However, data verification can be selected as a standard part of the product.

## NetWorker equivalents to SLS/ABS/Tapesys

- **Jukebox management.** NetWorker knows exactly which cartridges are in which slot in a jukebox, and automatically mounts those cartridges into free drives. It also provides the ability to manage importing and exporting of cartridges.
- **Media and File Index Database.** NetWorker stores the Media and File Index information in XDR encoded databases on the NetWorker server. NetWorker has a very powerful command set to interrogate these databases, and to export them in varying formats. Because the data is stored in an open format, you can even migrate these databases to systems that run a different Operating System.
- **Disaster Recovery, and High Availability.** NetWorker automatically backs up these databases as part of its standard backup cycle, and provides utilities to restore NetWorker in a Disaster Recovery scenario. NetWorker also supports Highly Available NetWorker servers using most of the industry standard UNIX and Windows clustering solutions, including LEGATO's own Application Availability Manager.

## Conclusion

OpenVMS continues to be the platform of choice for many important business applications. It is important to note that it increasingly co-exists with many other system platforms. Software solutions that facilitate the management and backup of OpenVMS systems in heterogeneous environments are essential to help control IT costs, decrease management complexity, and enable IT departments to meet Service Level Agreements. With the NetWorker Client for OpenVMS, NetWorker Storage Node for OpenVMS, and NetWorker Oracle Module for OpenVMS, LEGATO enables the integration of OpenVMS into a seamless enterprise backup solution so that IT departments can continue to take advantage of the unique capabilities of OpenVMS that has made it critical to so many businesses, without compromising the integrity of those systems.

LEGATO

LEGATO Software, a division of EMC

2350 West El Camino Real

Mountain View. CA  94040

USA

Tel  (650) 210.7000 • (888) 853.4286

Fax (650) 210.7032

www.legato.com

For a complete listing of LEGATO Software offices worldwide, please visit http://www.legato.com/offices

# OpenVMS Technical Journal

## Ask the Wizard

Stephen Hoffman, Senior Member Technical Staff
OpenVMS Engineering Group

The OpenVMS "Ask The Wizard" (ATW) area is an informal forum where you can ask questions about OpenVMS and HP Layered Products and Software. "Ask The Wizard" contains nearly 10,000 questions and answers on a wide range of OpenVMS topics, including IP printing topics, procedures for resetting the host name or host address, and other OpenVMS and layered product discussions.

Although questions arrive continuously at ATW, the answers are posted in batches, with a general goal of posting once a month. Longer intervals between postings are certainly possible.

If you require the absolute certainty of an answer, a quick answer, or a response by telephone, contact the Customer Support Center directly; use the more formal problem escalation channels that your question deserves. ATW is intended as an informal resource and not as a replacement for the expertise of the Customer Support Center.

The ATW area is available at:

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

You will know that you are on the right page if you see the photo of the Wizard (the Wooly Mammoth) busily answering questions.

On the main ATW page, you will find a list of links to topics like "OpenVMS information", "Evolving business value" and "Test Drive OpenVMS" on the left-hand Navigation bar. Directly beneath the "Overview" in the middle of the page is a link to the OpenVMS Frequently Asked Questions (FAQ) document. Under the FAQ link is an area to ask questions and search the previous answers. Beneath this area is a long list of previously-asked questions and answers. These resources taken together will answer most common questions.

As of this writing, there are four keystone topics listed at ATW:

- (1020) general IP printing, and HP printer blank pages
- (1661) memory management, application synchronization, and common coding errors
- (6776) resolving the access violation (ACCVIO) error
- (7552) general application debugging

These questions arise regularly, so they are highlighted on the ATW page.

On the ATW page, you will typically find several months worth of questions and answers, and links to archives of older questions and answers. Each of the questions and answers has a searchable title and an associated unique identifying topic number in parenthesis. There is a third number visible with each question, the posting order.

You will also find three search functions— one that uses an HP search engine, one that uses a local HP OpenVMS "Ask The Wizard" search function, and a third that performs a topic-number-specific search which finds a topic by its topic number, with or without parenthesis. You can use any or all of these searches, or you can choose to search the ATW site with Google or another external search tool.

At the bottom of the page, you will find a zip file containing all of the questions and answers, with current updates. Older questions are occasionally updated, so the entire zip archive is rebuilt each time a batch of questions and answers is posted.

http://h71000.www7.hp.com/wizard/wizard.zip

Within answers to some questions you will see references to existing discussions. For example, printing-related questions often cite topic (1020). You can locate topic (1020) by finding it in the roughly-monthly batches of questions, by the topic number search engine mentioned earlier, or by noticing the format of the URLs used for topics. For instance:

http://h71000.www7.hp.com/wizard/wiz_1020.html

refers to topic 1020.

The OpenVMS Wizard is well aware of hyperlinks and href tags and such, and would certainly welcome the addition of this support at the website. That written, the Wizard also has code to write and features to develop for OpenVMS releases.

If you wish to ask the Wizard a question, there is a link in the middle of the ATW page for this purpose. As a prolog to asking the question, you will see an explanation of the ATW area, as well as a list of general rules for asking your question.

As an example of a question that occasionally arises, assume that you want to find the log specification of a batch job, from within the batch job. You can use either of the two keyword search engines to look for queue or log, or you can search for titles containing queue-related discussions. In this latter case, one such match you might find is topic (9130), $GETQUI LOG_SPECIFICATION?, which provides you with the following DCL:

```
$ log_specification=f$getqui("display_job","log_specification",,"this_job")
$ IF log_specification.EQS."".AND.-
        NOT.f$getqui("display_job","job_log_null",,"this_job")
$ THEN
   $   log_specification=f$parse("",-
       f$getqui("display_job","job_name",,"this_job"),"sys$login:.log")
$ ENDIF
```

This code generates the batch log specification for the currently-executing batch job.

In a topic such as (9130), you will find pointers to other existing discussions of the F$GETQUI DCL lexical function, the SYS$GETQUI system service, and related features, including topics: (813), (1240), (2159), (3951), (4546), (4568), (4903), (5188), (5471), (5567), (5651), (5793), (5982), (6315), (6877), and (9130).

The OpenVMS Wizard looks forward to your questions, and hopes that you find OpenVMS and "Ask The Wizard" valuable, and that this article helps you more effectively use ATW.

# OpenVMS Technical Journal

## Effective Uses for the SYSGEN USER Parameters

Mark 'Jilly' Jilson

Proprietary Solutions Support Group

OpenVMS PCC Team

### Overview

The OpenVMS SYSGEN utility includes parameters that a system manager can use to increase a system's maintainability and increase application configurability. Integrating these parameters into system startup procedures may hasten resolution of system problems and can increase the flexibility of application startup.

### SYSGEN USER Parameters

SYSGEN has four USER parameters that can be used in many ways, as determined by the system manager.

As shown in **Example 1**, there are two dynamic parameters, USERD1 and USERD2, and two non-dynamic parameters, USER3 and USER4. They all have the same defaults and limits and are numeric parameters. None of them can be used to store text values. These parameters are not referenced by any other part of OpenVMS. The parameters can be maintained like any other parameter using the MODPARAMS.DAT and the AUTOGEN command procedures. These parameters also can be set at the SYSBOOT> prompt during a conversational boot. You can determine the current active value of the parameter by examining the DCL lexical F$GETSYI, as shown in **Example 2**.

```
$ RUN SYS$SYSTEM:SYSGEN
SYSGEN> SHOW USER
Parameter Name      Current    Default     Min.      Max.     Unit  Dynamic
--------------      -------    -------    -------   -------   ----  -------
USER3                     0          0         0        -1
USER4                     0          0         0        -1
USERD1                    0          0         0        -1             D
USERD2                    0          0         0        -1             D
SYSGEN> HELP SYS_PARAMETERS USER
Sys_Parameters
  USERD1
       USERD1 is reserved for definition at the user's site. The
           reserved longword is referenced by the symbol SGN$GL_USERD1.

       On Alpha systems, this symbol is in the
       SYS$LOADABLE_IMAGES:SYS$BASE_IMAGE module.

       On VAX systems, the symbol is in the SYS$SYSTEM:SYS.STB module.

       USERD1 is a DYNAMIC parameter.
```

**Example 1**

```
$ WRITESYS$OUTPUT F$GETSYI("USER3")
$ USERD1 = F$GETSYI("USERD1")
```

**Example 2**

## Controlling System Startup

There are times during system maintenance when the options OpenVMS provides for system startup (STARTUP_Pn parameters and SET/STARTUP) are limited in their functionality. Incorporating the SYSGEN USER parameters into a startup procedure can increase system startup flexibility and allow a system manager to complete maintenance operations in less time. **Example 3** shows the USER3 parameter being used to control which parts of SYSTARTUP_VMS.COM get executed. In this example, if USER3 equals 1, then only the network stacks will be started and all the disks mounted. None of the application startup procedures will be executed and DECWindows will not be started.

```
$ IF F$GETSYI(“USER3”) .EQ. 1 THEN GOTO No_Application_Boot
$ ! Normal startup procedure would follow here
.
.
$ EXIT 1    ! End of normal startup procedures
$ No_Application_Boot:
$ START/NETWORK DECNET
$ @SYS$MANAGER:TCPIP$STARTUP
$ @SYS$MANAGER:MOUNT_DISKS.COM
$ DEFINE DECW$IGNORE_DECWINDOWS TRUE
$ STARTUP$INTERACTIVE_LOGINS :== 0
$ EXIT 1
```

**Example 3**

This is a fairly simple example, but a system manager could implement many different startup configurations all based on the values to which the USER parameters are set. One USER parameter could be used to control which network ports are configured into the system while another USER parameter could be used to control whether a system would participate as a server for a TCPIP-based service. Adding these parameters into your system startup command procedures can significantly enhance the maintainability of a system.

## Controlling Application Startup

There are times when a system manager would like to boot a system and start up the system applications in a non-default manner. The USER parameters provide an excellent way to do this. You can edit SYLOGICALS.COM to employ the USER parameters to determine to which values the application logicals should be set. In a large cluster this strategy allows a system manager to have one subset of nodes act as the main application servers and another subset act as backup servers --- or the USER parameters can be used to disable individual parts of an application. These parameters can also be used to enable database verification procedures before normal application startup. The possibilities are limited only by how much control the application startup procedures provide.

## Documenting USER Usage

The last -- and very important! -- step in implementing the USER parameters is to document how they have been employed. The best place to document their usage in a system or a cluster is directly in the file where they are used, whether it be MODPARAMS.DAT, SYSTARTUP_VMS.COM, or any other command procedure. You should also print and keep a hardcopy document with other system documentation such as boot profiles and system configuration profiles.