Distributed Vault System Administrator and Manager Guide

Optegra® Release 6

DOC40039-009

Parametric Technology Corporation

Parametric Technology Corporation, 140 Kendrick Street, Needham, MA 02494-2714
8 January 2001

Table of Contents

Preface

Distributed Vault System Administrator and Manager Guide provides information for Vault system administrators and system managers for setting up a Distributed Vault environment.

Related Documents

The following documents may be helpful as you use the *Distributed Vault System Administrator and Manager Guide*:

- *Vault Manager Guide*
- *Vault System Administrator Guide*
- *Vault Interactive Query Facility Guide*
- *Vault command reference*

Book Conventions

The following table illustrates and explains conventions used in writing about Optegra applications.

Convention	Example	Explanation
EPD_HOME	`cd $EPD_HOME/install` (UNIX) `cd %EPD_HOME%\install` (Windows)	Represents the default path where the current version of the product is installed.
Menu selections	`Vault > Check Out > Lock`	Indicates a command that you can choose from a menu.
Command buttons and options	`Mandatory` check box, `Add` button, `Description` text box	Names selectable items from dialog boxes: options, buttons, toggles, text boxes, and switches.

User input and code	`Wheel_Assy_details`
	`-xvf /dev/rst0`
	`Enter command>` **`plot_config`**
		Enter the text in a text box or on a command line.
		Where system output and user input are mixed, user input is in bold.

System output	`CT_struct.aename`	Indicates system responses.
Parameter and variable names	`tar -cvf /dev/rst0` filename	Supply an appropriate substitute for each parameter or variable; for example, replace filename with an actual file name.
Commands and keywords	The `ciaddobj` command creates an instance of a binder.	Shows command syntax.
Text string	`"SRFGROUPA"` or `'SRFGROUPA'`	Shows text strings. Enclose text strings with single or double quotation marks.
Integer	n	Supply an integer for *n.*
Real number	x	Supply a real number for *x.*
#	`# mkdir /cdrom`	Indicates the root (superuser) prompt on command lines.
%	`% rlogin` remote_system_name `-l root`	Indicates the C shell prompt on command lines.
$	`$ rlogin` remote_system_name `-l root`	Indicates the Bourne shell prompt on command lines.
>	`> copy` filename	Indicates the MS-DOS prompt on command lines.
Keystrokes	`Return` or `Control-g`	Indicates the keys to press on a keyboard.

Online User Documentation

Online documentation for each Optegra book is provided in HTML if the documentation CD-ROM is installed. You can view the online documentation from an HTML browser or from the HELP command.

You can also view the online documentation directly from the CD-ROM without installing it.

From an HTML Browser:

1. Navigate to the directory where the documents are installed. For example,

   `$EPD_HOME/data/html/htmldoc/` (UNIX)

   `%EPD_HOME%\data\html\htmldoc\` (Windows NT)
2. Click `mainmenu.html`. A list of available Optegra documentation appears.
3. Click the book title you want to view.

From the HELP Command:

To view the online documentation for your specific application, click HELP. (Consult the documentation specific to your application for more information.)

From the Documentation CD-ROM:

1. Mount the documentation CD-ROM.
2. Point your browser to:

   CDROM_mount_point`/htmldoc/mainmenu.html` (UNIX)

   CDROM_Drive:`\htmldoc\mainmenu.html` (Windows NT)

Printing Documentation

A PDF (Portable Document Format) file is included on the CD-ROM for each online book. See the first page of each online book for the document number referenced in the PDF file name. Check with your system administrator if you need more information.

You must have Acrobat Reader installed to view and print PDF files.

The default documentation directories are:

- `$EPD_HOME/data/html/pdf/`doc_number`.pdf` (UNIX)
- `%EPD_HOME%\data\html\pdf\`doc_number`.pdf` (Windows NT)

Resources and Services

For resources and services to help you with PTC (Parametric Technology Corporation) software products, see the *PTC Customer Service Guide*. It includes instructions for using the World Wide Web or fax transmissions for customer support.

Documentation Comments

PTC welcomes your suggestions and comments. You can send feedback in the following ways:

- Send comments electronically to `doc-webhelp@ptc.com.`
- Fill out and mail the PTC Documentation Survey located in the *PTC Customer Service Guide.*

Chapter 1

Understanding Vault Transparency

Distributed Vault (DV) provides for a transparent and cooperative use of Vaults in a distributed environment. This chapter discusses the tasks and concepts associated with the transparency operation.

- Transparency and Distributed Vault
- Registering Objects for Transparent Access
- Distributed Object Directory
- Vault Commands Supporting Transparency
- Transparency Operation
- Review Administration
- Configuration Considerations for Vault Transparency

Transparency and Distributed Vault

Distributed Vault enables you to configure multiple servers (vaults) in a distributed, cooperative arrangement. This arrangement allows users to access selected files and parts on any of the configured vaults. Users do not need to know the locations of these selected objects. The transparent nature of Distributed Vault is comprised of the following elements:

- Registration: This mechanism makes a file or part accessible to users on other vaults.
- Distributed Object Directory: This repository holds registered files and parts. Vaults consult this directory in order to make a transparent item location.
- Client commands: These commands support the transparency of the vaults.
- Object Locator: This client process interacts with both the local vault and the distributed object directory to make objects accessible to users in the Distributed vault environment.
- Remote Access Protocol: This protocol enables transaction requests to remote vaults. A remote vault is a vault on which a user is not signed.

The sections that follow discuss these components in detail, and describe how the transparency of vault objects behave in the Distributed Vault environment.

Registering Objects for Transparent Access

Files and parts (also known as objects) become visible in the distributed vault environment when they are registered by authorized users. By registering an object, an authorized user declares that users from other vaults can access those objects. Because the distributed vault environment is transparent, users do not need to sign on to the vault that contains the file or part they need to access.

Please note: You cannot register a file or a part with Optegra Programming. An attempt to call the `REGISTER` command causes a 20,030 NSM error and produces additional `ADMIN_SERVERs` that are unusable.

The registration level, or the scope of usability, determines the degree of access given to users who attempt to access files and parts transparently. In order to determine this scope, the registrant of an object declares the registration level at the time of registration. Registration levels are as follows:

- Query: The object is represented in the distributed object directory (`DOD`), but users cannot access it. The distributed vault command `cilocate` will report it, however.
- Read: The object is accessible for read-only purposes. Any attempt to access the object for write purposes will fail.
- Write: The object is accessible for read-write purposes. This registration level allows all commands that support transparency to access the object.
- Ownable: The object is accessible for read-write purposes, similar to the Write registration level. In addition, the object is approved for an export-move operation. Use this special privilege only to move an object from one vault to another vault in a distributed vault environment.

Please note:

- When an object that was registered and made available in the `DOD` is reregistered as Local, it is removed from the `DOD` and is then available only to the local vault.
- You cannot register objects that have the revision as blank.
- You cannot change the registration level of a previously registered object, if the object is signed out.

Distributed Object Directory

Each vault in the distributed vault environment is assigned to a distributed object directory (`DOD`). The `DOD` is a repository of metadata for all registered objects from their originating vaults.

Using the ciregister command, the administrator can select the object's degree of availability within the distributed vault environment. For example, the following command registers revision 3 of file `test1`, with external read-access privileges.

```
ciregister selscope=f selname=test1 reglevel=r revision=3
```

Updating the DOD

The operational sequence for updating the `DOD` is as follows:

1. Whenever you register a file or a part, a `DOD` update action is inserted into the action manager action queue.
   a. The Action Manager processes all `DOD` updates in a first-in first-out manner, and ensures data delivery to the `DOD` at the earliest possible moment. Note that, because the `DOD` may not be online at all times, the action manager periodically checks for its availability before attempting an update.
   b. The `DOD` receives a replica of all metadata from the `DM_FILE_DIRECTORY` and `DM_PART_DIRECTORY` for the object, including its registration information.
2. When the `DOD` is updated, the object becomes visible to all the distributed vaults, according to its registration level.
3. Each modification to a registered file or part activates an update to the `DOD`.
4. When a registration level is set to L (local), the object is deleted from the `DOD` and becomes inaccessible to all the distributed vaults. The object is still accessible to its local vault.

Vault Commands Supporting Transparency

The following commands support the Distributed Vault transparency feature, and are described in detail in the *Vault Command Reference*.

- `cichgfa`
- `cichgfcl`
- `cichgfpw`
- `cichgfrev`
- `cichgfsc`
- `ciget`
- `ciread`
- `cireadmsg`
- `cireplace`
- `cireset`
- `cireqrvw`
- `cirsvp`
- `cisendmsg`
- `cisignout`
- `ciupdate`

For each of these commands, you must specify the vault identifier using the `vaultid` command modifier. This modifier determines how the command is directed (local, distributed or specific). Valid `vaultid` parameters are discussed in the sections that follow.

For convenience, you can place the vault identifier in the `EDM.DEFAULTS` file, and its default value can be set by the vault administrator.

Directing Commands to the Local Vault

To direct a command to the vault on which the user has signed on (the local vault), omit the `vaultid` command modifier. Optionally, you can specify the following to achieve the same effect:

```
vaultid=local
vaultid=l
vaultid=*   (in UNIX environments, vaultid=\*)
```

Directing Commands to a Specific Remote Vault

You can send any of the commands to a specific remote vault by specifying the remote vault's domain name as the vaultid parameter. For example, the following command causes the transparency feature to select myfile from the vault named `ENGINEERING`:

```
ciget selname=myfile selscope=f vaultid=ENGINEERING
```

This command has the same effect as the `vaultid=query` command but restricts the selection to the domain of the named vault. If the vault domain happens to be the local domain, then this command has the same effect as the `vaultid=local` command.

Please note: The `readmsg` and `sendmsg` commands require a vault domain name as a `vaultid` parameter. For example, the following command sends a message directly to the vault named `RESEARCH`:

```
cisendmsg userid=nick msgtext='Review your mail' vaultid=RESEARCH
```

Using Transparency to Select an Object

To locate and select an object in the distributed vault environment, specify the `vaultid` command modifier with one of the following parameters:

```
vaultid=query
vaultid=q
vaultid=?   (in UNIX environments, vaultid=\?)
```

For example, the following command causes the transparency feature to select a suitable version of the named object from the `DOD`.

```
ciget selname=myfile selscope=f vaultid=query
```

Please note: You cannot use the `sendmsg` and `readmsg` commands in this manner.

If there is no suitable item to select, the command displays an appropriate message. If a suitable item is selected, the client process connects to the appropriate vault server in the domain. The selection algorithm is further discussed in the next section.

Transparency Operation

The algorithm that the transparency feature uses when selecting an object to suit a vault command is briefly described here for your understanding. For each distributed vault command, the algorithm performs the following actions:

1. Evaluates the command to identify a transparency request and to validate the vaultDomainName parameter.
2. Invokes the object locator process and supplies it with the values of selname, selscope, revision, the command code, the value of the vaultDomainName parameter, and the name of the local vault domain.
3. Restricts the object locator search process to items that reside on the remote or local vault, as specified by the vaultDomainName parameter. If the vaultDomainName parameter refers to a local vault, then the object locator function stops and the transaction proceeds as if no vault identifier were supplied.
4. Selects the object and, using a Remote Access Protocol to execute the command, contacts the vault that owns the object and performs the requested transaction.

Consulting the DM_VAULT_CONFIG Table

The `DM_VAULT_CONFIG` table contains a list of valid configured vaults to which transparency requests can be made. This table results from the configuration of vaults and object directories. For further information, refer to the *Vault Command Reference* for descriptions of the following commands:

- `ciaddadod`
- `ciremadod`
- `ciaddvault`
- `ciremvault`

Each entry in the `DM_VAULT_CONFIG` table identifies a vault (or `DOD`) and further identifies whether the local vault considers the remote vault (or `DOD`) to be online. The object locator process consults the table for two purposes:

- To determine if the `DOD` is online. If it is not, the command terminates. When the `DOD` is offline, the object locator cannot query the `DOD`.
- To query the `DOD`, based on the selname, selscope, and revision values supplied to it. The query is restricted to only those items that reside on online remote vaults in the `DM_VAULT_CONFIG` table. This set of vaults is ordered by their sequence numbers.

Each entry in the `DM_VAULT_CONFIG` table has a sequence number, which sequences the selection set by the object locator process. In general, the local vault is configured with the sequence number 1. Other remote vaults can be assigned higher sequence numbers.

Selecting an Object

The object locator process selects the first object in the set that matches the intent of the command, from the sequenced set of objects. When a selection occurs, the vaultid value of the object is returned to the client process.

The local vault is analyzed first. Any item that resides on the local vault and satisfies the request is selected, in preference to non-local vault items. Objects cannot be selected from a local vault when any of the following conditions exist:

- The item is archived, deleted, or a placeholder.
- The item is in the process of being moved to another vault.
- For requests other than read-only, the item is either the subject (original) of a DV Export replicate-write operation or the object (copy) of a DV Export replicate-read operation.
- The revision is specified and the item revision is not an exact match.

When none of these conditions exist for an object residing on the local vault, the object is selected by the locator process, and no further location processing occurs.

When the object locator does not find a suitable object on the local vault, it attempts to select one from a remote vault. The following conditions apply:

- The remote vault that contains the object must be online. Objects residing on remote vaults must appear as online from the perspective of the local vault.
- The registration scope must be compatible to the request. A read-only request requires a registration level of R, W, or O. A write-request requires a registration level of either W or O.
- For a GET request, the object cannot be a placeholder, archived, or deleted.
- For non-GET requests, if the object is released, a placeholder, archived, or deleted, the revision must be specified.
- The object cannot be the subject (original) of a DV Export move operation.
- For requests other than read-only, the object cannot be the subject (original) of a DV Export replicate-write operation or the object (copy) of a DV Export replicate-read operation.
- If revision is specified, the object revision must be an exact match.

If no item is selected by the object locator, the process displays the following message, accompanied by a status for each object that was rejected by the process:

```
CDMGET792E No suitable version of fileName is found:
```

When the object locator process completes its function, it hands the client process the vault identifier associated with the object. The client then connects directly to that vault, and the transaction is complete.

Completing the Transaction

To complete the transaction, the `DD` (data distribution) or `ADMN` (data administration) process that receives the transparency request (the receiving vault) performs the following operations:

1. Validates the originating vault identifier. This ensures that the originating vault is configured to the receiving vault, and that the originating vault has the requested command in its public command list. If these checks are unsuccessful, a security violation is logged and the transaction fails.
2. Processes the request. From this point, no distinction is made to whether the request arose from a transparency request. The user-ID of the transaction is the same as for the user that made the request, and all user, file, and project validation is performed in the same manner as a local transaction.

Review Administration

The Request for Review (`cireqrvw`) and Respond to Review (`cirsvp`) commands honor the vaultid command modifier, therefore they can perform operations that include users from many vaults. The following conditions apply:

1. Place the Reviewer List for an object on the same vault as the file or part.
2. Ensure that all users in the Reviewer List are valid users in that vault.
3. Register an object with a level of at least W for transparent access.
4. Set the user's home vault to the vault where the user typically signs on. This may not be necessary with the Email Trigger, because the reviewer receives a review notification by email.

   A user (or an authorized administrator) can set the home vault with the `ciaddalias` command. This action redirects messages to the user's home vault. The reviewer on a remote vault can then read the file or part and use the cirsvp command to make a review response. The user need not be signed on to that vault to do this.

Configuration Considerations for Vault Transparency

When configuring your Distributed Vault for vault transparency, you need to make specific considerations regarding users and projects.

Vault Users

A vault user-ID always represents the same vault user. The administration of users among the distributed vault environment must be consistent with respect to the assignment of user-IDs. If a user attempts to access a file on a remote vault (that is, not on the vault to which the user is currently signed on), then that user's ID must be valid on the remote vault, and must refer to the same user.

Vault Add Command

You should typically specify `TYPE=VAULT` in the Vault Add command. `TYPE=SELF` implies configuring the Vault on which the command is being entered as a distributed vault. This occurs silently during the initial DV installation.

Projects

For projects, including the default public project, the distributed vault environment assumes a consistent project definition. This holds true for any project of registered files and parts.

Project Names and Revision Code Sequences

The project name and revision code sequence must be consistent among participating vaults. The project's life cycle need not be identical, but you need to assign status codes carefully if this is not the case.

Status Levels

In general, order the set of status levels consistently. However, the relationship of a status level to an authority number need not be identical. For example, one could assign the IN-WORK-2 level an authority code of 10 on one vault and 20 on another. However, the placement of the status level in the authority scheme must be consistent.

User Configurations

Vault users assigned to a project may need to be configured with the `ciaddup` and `cichgup` commands on all vaults that participate in the project.

Chapter 2

Administering Distributed Vault

This chapter provides information for setting up a distributed Vault environment. It also describes how to diagnose and correct certain problems that may occur.

- Project Administration
- Troubleshooting
- Registered Files and Parts Not Visible to Remote Users from the Server Side
- Registered Files and Parts Not Visible to Remote Users from the Client Side
- Autoregistration on Distributed Vault

Project Administration

There exists significant administrative overhead when defining projects, status codes, revision code sequences, review lists, command lists, and so on. However, if you use transparency to give project users access to files and parts in a project, then the job of project definition and control is restricted to a single Vault, even though users in any connected Vault will have access to your project's files and parts.

When Vaults are connected by a fast network, transparency is the choice of reason for administering projects. Transparency affords fewer complications than the Export-Import procedure, and it allows the project administrator to extend the capability of the project without increasing the complication of its administration, and without compromise to security.

For a user working on several projects, transparency provides a way to select items from any connected Vault, without regard to location. Each Vault's project administrator maintains complete control over users access to files on that vault.

Project administration, for any particular project, is localized to the vault that owns the project. If it becomes necessary to transfer the project contents to another vault, the Export-Import procedure can be used for this purpose. However, because there is currently no single function to do this, some scripting would facilitate the procedure.

Although a user can access a file transparently, that action does not mean he or she can sign on to the Vault where the file resides. The Vault administrator might not include a signon value in that user's public command list or might withhold or invalidate the user's password. Transparency in no way jeopardizes the security of vault from outside access, outside the realm of objects specifically registered for access.

The Distributed Vault transparency feature offers simplicity of administration, and this method is recommended for controlling file and part access when the network is adequate for anticipated file transfers.

Setting Up the Project

You must establish a new user-ID, such as `DVADMIN`, for executing administrative commands related to distributed Vault. Use this user-ID to execute commands, and to receive messages such as those from Export-Import operations and Action Manager notifications. The `DVADMIN` user-ID should have the same privileges as `EDMADMIN`.

The Vault user-ID can execute the following commands:

- `REPLACE`
- `READ`
- `GET`
- `STORE`
- `CHGFA`
- `CHGFCL`
- `UPDATE`
- `CHGFREV`
- `CHGFSC`
- `MARKD`
- `PURGE`
- `SIGNOUT`
- `RESET`
- `REGISTER`
- `REQRVW`

The Vault containing the project is the central controlling Vault, where you set up all project tables and users. On the central Vault, set user aliases for all users who would not typically sign on directly to this Vault. If you need to restrict sign-on access to this Vault, then ensure that those restricted users do not have a password, and that they do not know the password. Alternatively, you can ensure that the `SIGNON` command is not included in restricted users public command lists.

Each user should be assigned a home Vault, the Vault to which he or she would most likely sign on. This ensures that users need to sign on only to the home Vault and that all review messages are routed to their home Vault.

Troubleshooting

This section discusses the communication problems and the solutions related to the Distributed Vault transparency feature. It is assumed that all appropriate installations have been made and are verified as complete and correct.

Except for the problem that Vaults and `DOD`s are online or offline, there should be no other problems of communication with the transparency feature. It is identical to nontransparent client-server transactions, with the addition of the various `DOD` interactions.

Registered Files and Parts Not Visible to Remote Users from the Server Side

Although a file is registered, occasionally a complaint may arise that it is not visible to users on remote Vaults. This section lists potential causes and solutions to this problem.

Remote DOD

Vault Down

If the distributed object directory (`DOD`) is remote to the Vault from where the registration was made, then it is possible that the Vault where the `DOD` resides is down. Check the `DM_VAULT_CONFIG` table on your local Vault. This will help you determine whether that Vault is down, or whether there are network difficulties between your site and the remote site where the `DOD` is located.

The Action Manager, a server process running in the Vault, has, as one of its duties, the job of determining the current state of all Vaults and `DOD`s, based on the content of the `DM_VAULT_CONFIG` table. The action manager process attempts, every thirty minutes, to connect to the process manager process on each Vault identified in the `DM_VAULT_CONFIG` table. A failure to connect causes the action manager to mark that Vault as down (`DM_VAULT_STATUS=1`). This condition persists until the action manager makes a successful connection to the process manager. When this occurs, the Action Manager sets the Vault status to 0 (online).

Using SQL*Plus, you can set `DM_VAULT_STATUS` to `0`, forcing the `DOD` online and enabling the action manager process to attempt `DOD` updates. If this action fails, the `DM_VAULT_STATUS` value automatically reverts to `1` (offline), and there are no harmful consequences.

When a remote `DOD` is offline, then any `DOD` updates arising from a Vault are held until the `DOD` is back online. Consequently, this is one way that a registered file might not be visible to the rest of the world.

You should verify that the action manager process is running. In Vault, this server is called `EDMAMAN`, and there should always be one of these servers running. You can, at any time, issue the `nsmstop` command, to stop the server (if it is running), and then issue the `nsmstart` command to start a new instance. Refer to the *Vault System Administrator Guide* for complete information about these and other network services commands.

DOD Online

Updates Not Dispatched

If the DOD appears to be online, it is possible that updates are not being dispatched. Check the DM_VAULT_CONFIG table for the DM_VAULT_CONNECT flag that pertains to the row that identifies the DOD. This flag is set by the action manager process whenever the action manager starts a service application entity (AE). The service AE extracts data from the Vault, and issues Insert, Update and Delete directives to the DOD. When a service AE is dispatched by the action manager, the action manager notes this by setting DM_VAULT_CONNECT = 1, indicating that a service AE is in process, and preventing action manager from continuously spawning service AEs for each new action.

Occasionally, the service AE terminates quietly. When this occurs, the DM_VAULT_CONNECT flag remains set. Consequently, the action manager process ceases to spawn a service AE for DOD updates. To clear this condition, use the nsmstop and nsmstart commands on the action manager program, called EDMAMAN. Review the syslog message file to find any errors that were encountered by either the service AE or the action manager.

Actions, including DOD updates and other subscribed actions, reside in the DM_ACTMANAGER table and are dispatched by the action manager process to an appropriate service AE. Each row in this table contains a column titled DM_ACTION_JOINKEY. This 10-character key displays as a prefix to all messages that are generated by the service AE, so it is possible to correlate a message to a specific action. In addition, this same key is used to join the DM_ACTMANAGER table to a table called DM_ACTION_DATA, where all of the action tokens are stored. You can fetch these tokens from DM_ACTION_DATA, using a query such as the following:

```
select DM_ACTION_KEYWORD, DM_ACTION_DATA from DM_ACTION_DATA
where DM_ACTION_JOINKEY =’action_joinkey’;
```

The response will indicate the nature of the failing action.

Action Manager Not Running

If the DOD appears online, and the DM_ACTMANAGER table looks reasonable, then verify that the action manager process is in fact running. To do this, issue the nsmquery -pm command to search for the EDMAMAN AE. If it is not running, start it by issuing nsmstart, or restart the Vault entirely and it will start up automatically.

When the Action Manager is not operating, any action (such as a Registration) may be residing in the table called `DM_ACTION_QUEUE`. This table is the input queue to the Action Manager. If this table has entries and they do not clear within 30 seconds, then the Action Manager is either not running or it is not operating correctly and should be restarted.

Event Manager Server Not Running

If everything appears correct so far, check the Event Manager server. This server is responsible for populating `DM_ACTION_QUEUE`. It derives its input from a table called `DM_EVENT_QUEUE`. This server's identity is `EDMEMGR` and should always be running. If it is not, you need to start it. This server periodically queries `DM_EVENT_QUEUE`. The frequency of its query depends on the timestamp of the last query that found work. The Event Manager server increases its query interval to about 5 minutes if it continues to find no events in the `DM_EVENT_QUEUE`, so a 5-minute wait may be necessary to observe that it is operating correctly.

Configuration Errors

If you cannot resolve the problem, you need to validate the installed configuration. Possible problem areas could include the following:

- Your `PM.CONFIG` file is incorrect. This could lead to false indications that the `DOD` is offline.
- Your Vault-ID is not known to the `DOD`. This should be verified by the administrator of the `DOD`.

Registered Files and Parts Not Visible to Remote Users from the Client Side

After verifying that the server side (the Vault that is supplying the registered objects to the `DOD`) is in reasonable shape, take a look at the client side.

Vault Not Visible to Client

If the Vault from which the user issues the command considers the Vault being called to be offline, it issues the following message:

```
CDMGET792E No suitable version of filename found:
VaultName:Rev1:OffLine
```

When this occurs, the administrator must determine if this status is valid.

Similarly when problems occur at the server side, check to ensure that the action manager process is active and running. If it is not any prior status `(DM_VAULT_STATUS)` will remain.

If the action manager process is running and the Vault is considered offline, then you need to determine if that Vault is visible to the client's Vault. To do this, attempt to sign on to the server Vault from the client Vault host machine, using a well known user-ID (for example, `DVADMIN`). If sign-on is not possible, then verify the configuration. If sign-on is possible, then try restarting the action manager process.

Having performed these actions, if the client Vault still does not consider the server Vault online, try forcing `DM_VAULT_STATUS` for the server Vault to `0` (online), and retry the test.

DOD Not Visible to Client

If the user receives an error indicating the `DOD` is offline, then the problem is much the same as the server Vault being considered offline, except that you now need to be concerned about the `DOD` instead. Follow the same troubleshooting procedure described for that problem.

Autoregistration on Distributed Vault

Autoregistration is enabled during the `STORE/GET/COPY` operations to Vault. All Files/Parts should be registered at store/get/copy time. This maintains DOD up to date with all Distributed Vaults.

Please note: If the registration fails, then the operation is carried out successfully, displaying a warning.

Installation for Distributed Vault is done by `EDMDVINSTALL.`Refresh is done by `EDMDVREFRESH.`

To implement automatic registration on Distributed Vault, the `EDMDVINSTALL` and `EDMDVREFRESH` scripts have been modified to accept your choice.

The Vault Administrator can enable/disable automatic registration in any one of the following ways:

- Running `edmdvinstall` (for a system with Vault installed for the first time).
- Running `edmdvrefresh` (for a system whose Vault is being refreshed)
- Manually editing the `nsm.config` file and adding/modifying the relevant lines in it.

To confirm the respective settings, do the following:

1. Locate the following lines of code in the `nsm.config` configuration file located in the `$EDM_HOME/data` directory.

```
#  Application Entity for PDM Data Distribution Facility.

   AE(PDMDD,edmgrp,3)
      PATH($EDM_HOME/scripts/DD.STARTUP)
      OWNER(edm)
      WORKDIR(/tmp)
      CLOSE
      SERVER
      CONCURRENCY(1,1)
      MAXINST(6)
      GRPCTL(1,1,2)
```

2. Insert the following lines after `WORKDIR(/tmp)` to enable autoregistration.

```
     USER(AUTOREGISTER=YES)
      USER(REGLEVEL=W)
```

   The Registration levels can be:

   - W — Writable: This level allows remote access to the file/part for Vault file access commands which honor the VAULTID parameter.

- `R` — Read only: This level restricts remote access to the file/part to the Vault `READ` command.
- `Q` — Query only: The level enables file/part metadata to be sent to the DOD and is available for query, but no Vault operations are valid.
- `O` — Ownership: Beyond `W`, this level also allows exporting (EXPORT) of this file/part for a move [tag=move] operation.

3. To discontinue autoregistration replace the value `YES` with `NO`.

Please note: Private Files/Parts will be stored without registration.

Limitations

The automatic registration uses the VAULTID for logging in for the registration procedure. Hence, even if you do not have registration in the user list of available commands, registration will still go through successfully.

Chapter 3

Remote Exporting and Importing of Parts or Files

This chapter describes the facility of the Import and Export services which replicate registered files from one Vault to other, in the Distributed Vault environment.

- Introduction
- Exporting Parts and Files
- Importing Parts and Files
- Tags
- Setting Registration Scope
- Setting the Distributed State of an Object
- Useful Tables and Files
- Performing InterVault Communication
- Using Subscriptions
- Import Manager
- Graphical User Interface

Introduction

Export and Import services replicate the registered files and parts from one Vault to other, in the distributed Vault environment. This is a two-stage process:

1. Exporting Vault*:* The user selects a file or part and exports it.
2. Importing Vault*:* The user imports the exported object.

The Export-Import feature is useful in the following situations:

- A file or part has a high read-only access rate, particularly from remote vault users, and the interconnecting network is overburdened.
- You need to move a file to another location.
- You need to subcontract the file to a remote vault for updates, with the expectation that it be returned to the originating vault in its updated state.

In order to successfully export and import files between Vaults, the Vault user-ID of both the sending and receiving Vaults must be included, as users in the project(s) to which exported/imported files/parts belong.

Please note: The user-IDs must have the privileges to override certain pre-existing conditions.

You can use the Export/Import commands on files and parts belonging to rulebases other than LOCAL and CADDS.

Following is a list of terms that apply to the export-import process.

- Exporter*:* The name of the Vault where an export operation was performed. The exporter is identified by the name of the domain of the vault.
- Importer: The name of the Vault that performs the import operation. The importer is identified by the name of the domain of the Vault.
- Vault List: A list containing names of vault domains that are configured for distribution to the current Vault. The Vault list is a convenience list to create lists of eligible importers when issuing an export command. When an object is exported to a Vault list, all members of the Vault list are eligible to import the object. When all members of the specified vault list have imported the exported object, the import process is considered complete.
- Tag: A name used to define an export queue, and also to define the purpose, or intent, of the export operation. Tags are described in detail in a later section.

Exporting Parts and Files

The `ciexport` command copies registered objects from their source Vault to the export directory. An exported object is kept in this directory until all the Vault IDs identified in the Vault list (VAULTLST) imports it.

When exporting an object, specify a:

- tag to determine the object access privileges (`COPY`, `MOVE`, `REPLICATE_READ`, `REPLICATE_WRITE`, `UPDATE`, and `REPLACE`).
- VAULTLST parameter to determine the importer Vault list. This identifies the Vault destinations.

Please note: The `EXPORT` command does not support files with a password. Exporting a file with a password displays an error message.

Settings for Export Command

To export files and parts belonging to the other rulebases specify one or more of the following values in the `$EPD_HOME/data/EDM.DEFAULTS` file.

- EXPORT_PREFIX
- EXPORT_SUFFIX
- EXPORT_FILEDIR

To set the parameters for each rulebase, use the following syntax in the `EDM.DEFAULTS` file:

- `EXPORT_PREFIX(rulebase_name)=` suffix_values (if any)
- `EXPORT_SUFFIX(rulebase_name)=` prefix_values (if any)
- `EXPORT_FILEDIR(rulebase_name)=` OS level representation of the files and parts.

  The values are:

  - F — If the object is stored as a file.
  - D — If the object is stored as a directory.

Please note: You need not set these parameters in the `EDM.DEFAULTS` for the files and parts with representations similar to CADDS and LOCAL.

- If both a SUFFIX and a PREFIX exist for an object, specify both the values in the `EDM.DEFAULTS` file.
- If SELSCOPE=F and the OS level representation of the object is a file, then do not specify the EXPORT-FILEDIR value in the `EDM.DEFAULTS` file.
- If SELSCOPE=P and the OS level representation of the object is directory, then do not specify the EXPORT-FILEDIR value in the `EDM.DEFAULTS` file.

Exporting PS File

To export the PS file, set the following in the `EDM.DEFAULTS` file:

- On UNIX:

```
EXPORT-SUFFIX(PS)=/_ps
EXPORT-FILEDIR(PS)=D
```

- ON WINDOWS NT:

```
EXPORT-SUFFIX(PS)=\_ps
EXPORT-FILEDIR(PS)=D
```

To store a PS file in Vault, it requires `lfname` as directory name, followed by suffix value (/_ps for Unix and _ps for Windows NT) for EXPORT-SUFFIX (`rulebase_name`) variable.

`lfname` is the localFileName which is name of the local file or directory in your local area. The filename length can be up to 80 characters.

Please note: Do not set the EXPORT_PREFIX variable in the `EDM.DEFAULTS` file, if the prefix does not exist.

Exporting MEDSHT File

To export the `MEDSHT` file, set the following in the `EDM.DEFAULTS` file:

- On UNIX:

```
EXPORT-PREFIX(MEDSHT)=s.
EXPORT-FILEDIR(MEDSHT)=F
```

- ON WINDOWS NT:

```
EXPORT-SUFFIX(MEDSHT)=s.
EXPORT-FILEDIR(MEDSHT)=D
```

MEDSHT part is represented at the OS level as a file starting with `s.partname`.

When a `MEDSHT` file is stored in Vault, it requires `lfname` as part name, with `.s` as the prefix value for EXPORT-PREFIX(`rulebase_name`) variable. In this case SUFFIX does not exist.

Please note: Do not set the EXPORT_SUFFIX variable in the `EDM.DEFAULTS` file, if the suffix does not exist

Using Export

An object can be exported by using:

- `EXPORT` (CI: Command Line Interface)
- `DISTRIBUTE` (EPD.Connect)

1. Select View > Distributed > Files or Distributed > Parts.

   A `DOD` view is displayed. This is the view from which you can use the `Export` command.
2. From the Admin menu:
   a. Select Admin >Distribute Now. A list is displayed. Select an Intent and a Vault list.
   b. Select Distribute On Event. Enter a subscription for exporting an object when an event occurs. A pop-up selection list is displayed.
3. Select the required Vault list.
4. Choose Apply or OK. The selected object is exported.

Input Data

Object details such as type, name, revision, tag specification, and exporter list are the same for `ciexport` command. The additional information required is:

- If the object is represented as file or directory at the OS level
- The prefix/suffix of the object to be exported.

The `EXPORT` command takes the input from the specified command line. Additional required information is picked up from `EDM.DEFAULTS` file.

The server that process the `EXPORT` command, retrieves the object in the export directory and adds the additional information to the relevant files in the export directory. The `EXPORT` command generates a message in the standard Vault message format. The message is written to the audit log `EXPORT.EDMAUDIT`.

Example:

```
ciexport selscope=P selname=testpart.asm revision=1
tag=replicate_read vaultlst=LOCALDOD
```

Importing Parts and Files

The `ciimport` command transfers objects from the export directory of a remote Vault to the importing Vault.

You can issue the `ciimport` command to import objects, however, each Vault automatically imports objects from known Vaults (see `ADDVAULT`).

You can import an object by using `IMPORT` (CI). Object details such as type, name, revision, tag specification, and importer list are the same for `ciimport` command.

Please note: The `ciimport` command must be in the public command list.

The server that processes the `EXPORT` command, retrieves the object from the export directory.The `IMPORT` command generates a message in the standard Vault message format. The message is written to the audit log `IMPORT.EDMAUDIT`.

Example:

```
ciimport selscope=P selname=testpart.asm revision=1
tag=replicate_read exporter=VAULTNAME <name of the exporter vault>
```

For more information on Exporting and Importing files, refer to *Vault Command Reference.*

Tags

A tag is a name used to define an export queue, which contains exported objects queued up to be imported by their importing Vaults. A tag defines the purpose, or intent of export operations that use the tag.

The export queue of a tag exists at `$DATA_DIRECTORY/oaxis/xd/tagName`

This directory contains the objects that have been already exported to the tag but are not yet imported. Each tag is configured by the placement of a file named after the tag in the `$DATA_DIRECTORY/oaxis/tags` directory. This file contains the tag information and also defines the purpose of the export-import operation.

There are six types of Vault tags. They are:

- Copy — Specifies that the exporter exports the object as a copy to another Vault. After the object has been imported, the exporter and the importer have fully usable copies of the object.
- Move — Specifies that the exporter exports the object for the moving to the importer. After the object has been imported, it is marked for deletion at the exporter. A fully usable copy is then available at the importer.
- Replicate_read — Specifies that the exporter exports the object as a read-only copy to the importer.

  At the completion of the import, the importer has the object signed out as the exporting Vault and therefore the user can only perform a read operation on the object. The exporter would have a fully usable copy.
- Replicate_write — Specifies that the exporter exports the object for the purpose of providing a writable copy to the exporter.

  The object is signed out to the export user at the exporting Vault. At the completion of the import operation, the importer has a fully usable copy.
- Update — Specifies that the exporter wants to update the original of a writable copy. This tag is used to export objects that have been previously imported with replicate_write.

  The object remains usable at the exporter and remains signed out at the importer after the import is done. This tag allows you to update the source version while leaving the replica available for further modification.
- Replace — Specifies that the exporter wants to return a copy to its owning Vault. This tag is used to export objects that have been previously imported with replicate_write or replicate_read.

The object is marked for deletion at the exporter. The importer's object becomes fully usable and is updated. This tag allows you to return the replicated object to the source Vault, while removing the replica.

The relationship between tag and Vault list is shown in the table that follows:

Table 3-1

Tag	Who Can Import
copy	Only one importer can be specified.
move	Only one importer can be specified.
replicate_read	Many importers can be specified.
replicate_write	Only one importer can be specified.
update	Only the source of the replica can be specified.

Please note: All importers are specified by including them in the Vault list. You can create and update the Vault list using the UNIX commands `ciadvlst`, `ciremvlst`, `ciaddvlmem`, and `ciremvlmem`. Alternately, you can select View>Admin>Vault List from the GUI. Refer to the *Vault System Administrator Guide* for more information.

Setting Registration Scope

All objects must be registered before they can be exported**.** The registration command accepts a registration-level argument that is stored as the registration scope. The registration scope determines to what extent the exporter can export the object.

The following table illustrates the relationship between registration scope and permissible tags for export.

Registration Scope	**Description**	**Permissible Tag**
R	Read	replicate_read
W	Write	replicate_write copy replicate_read
O	Ownership	move copy replicate_read replicate_write

Setting the Distributed State of an Object

During the process of export and import, objects can be put into a distributed state depending on the intent of the export. In a distributed state, certain operations might not be permissible on the object. The following table illustrates the meaning of the distributed states.

Table 3-2

Distributed State	Description
IR	Inbound read replica
IW	Inbound write replica
OM	Outbound move
OW	Outbound write replica

The next table illustrates the legal registration scope and distributed state for an object to be imported. It also illustrates the relationship between a distributed state and the various stages of export-import. Distributed states are stored in the registration table as registration reason.

Table 3-3

Import Operation	Registration Scope	Distributed State
copy	—	—
replicate_read	—	—
	R	R
replicate_write	—	—
move	—	—
update	W	OW
replace	W	OW

The following table illustrates the permissible use of objects in various distributed states.

Table 3-4

Distributed State	Permissible Operations
IR	READ operation. The object is a read-only copy of an original. The object is signed out to the source of the replica.
IW	All Optegra operations except deleting and archiving.
OM	No Optegra operations because the object is in the process of being moved.
OW	READ operation. The object is the source of a writable copy. The writable copy exists in another Vault, and has a distributed state of IW.

The following table illustrates the export-import operations that can be performed on an object with a distributed state.

Distributed State	Permissible Export	Permissible Import
IR	replace (only to peer Vault)	replicate_read (only from peer Vault)
OW	None	None
IW	update (only to peer Vault) replace (only to peer Vault)	None

The following table illustrates the state of the object after an export or import operation is performed.

Distributed State	Export Operation	Object State in the Exporter
—	copy	Fully usable
IW	update	Fully usable
IW	replace	Object deleted when imported
Distributed State	**Import Operation**	**State of Object in Vault**
IR	replicate_read	None
OW	update	Read only
OW	replace	Fully usable

Useful Tables and Files

Distribution Control Table

All exported objects that have not yet been imported are recorded in a table called `DM_DISTRIBUTION_CONTROL`. This table has one row for each exported object under a tag for each importer. When all intended importers have imported an exported object, the entries are deleted from this table. This table contains useful information, such as `IMPORTER, XD` (the path name where the exported object is located), and the state of the exported object. Refer to Appendix A, “Distributed Vault Database Tables”.

Distribution History Table

The distribution history table, called `DM_DISTRIBUTION_HIST`, contains one entry for each export-import operation. This table contains the date and time stamp of the export-import along with the peer Vault name, object name, type, revision, and the tag.

Export Audit File

For each export command, the export manager process creates an export audit file, named `EXPORT.EDMAUDIT.yymmdd`, under the `/tmp` directory. This file contains detailed results of all operations that were performed while exporting the object. If you perform the export operation using `ciexport`, an audit file called `EXPORT.EDMAUDIT` is created in the directory from which the command is issued.

Import Audit File

For each import command, the import manager process creates an import audit file, named `IMGR.AUDIT.yymmdd`, under the `/tmp` directory. This file contains detailed results of all operations that were performed while importing the object. If you perform the import operation using `ciimport`, an audit file called `IMGR.AUDIT` is created in the directory from which the command is issued.

Performing InterVault Communication

This section discusses how to accomplish interVault communication using the Export-Import feature. InterVault communication allows you to maximize accessibility to remote users, while maintaining varying degrees of control at the owning Vault.

When importing or exporting files between two Vaults of a shared project, the project needs be available on both the Vaults.

If the project does not exist on both the Vaults, the following functions will fail:

- the export operation of a file that is in a project
- the import operation of a file that is in a project

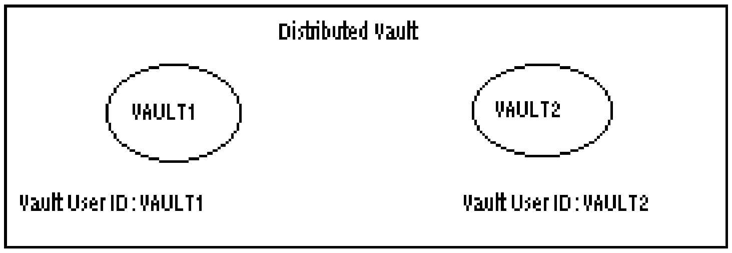
  Such a failure is recoverable by performing appropriate project administration on the importing Vault.

Please note: In order to Export a file, it must be registered. The Export facility uses and modifies a file’s registration when that file enters into the distributed state. Also note that files and parts are imported to the importing Vault using the importing Vault’s domain name as the user-ID. This special user-ID must be appropriately configured for any project in which a file is to be imported.

Configuration of Vault User ID’s when using Distributed Projects

If you want to share files or parts (that is; export/import) within an Optegra Vault project, you must ensure that the Optegra Vault ID's are configured, as users, in that project. This must be done on all Vaults that participate in the distributed Vault environment.

Let us consider a Vault named `VAULT1`. When Distributed Vault is installed on `VAULT1`, a user ID of the Vault domain name (that is; `VAULT1`) is automatically created for the corresponding distributed Vault. The same is true for other distributed Vaults, like `VAULT2.` Distributed Vaults have a User ID, corresponding to their Vault domain name.



All the Vault User ID's of the Vaults participating in the distributed Vault environment, need to be included in the user list of the shared projects. For more information on assigning a user to a project, refer to the `ciaddup` command in the Vault Command Reference.

For authorization, each Vault User ID must have a command list which includes:

- `read`
- `get`
- `store`
- `replace`
- `update`
- `reset`
- `chgfa`
- `chgfcl`
- `chgfrev`
- `chgfsc`
- `markd`
- `purge`
- `signout`
- `sendmsg`
- `register`
- `reqrvw`
- `rsvp`

The Vault User ID, must have Administrative privileges and read/write authority numbers. This ensures that the Vault User ID can modify any distributed file or part.

Please note: The Vault User ID’s are invoked only during Export and Import operations. No end-user process or person needs to access these user IDs, thus their passwords may be kept confidential.

Using Export-Import

To create a read-only version of a file that is available to many remote Vaults, specify an export command. The syntax follows:

```
ciexport tag=replicate_read selname=objectName
selscope=fileOrPart vaultlst=listOfVaults
```

The Vault list specified by listOfVaults contains the Vault identifiers of all the Vaults that should receive the file. The export operation ignores the local Vault if it is encountered in this list. Therefore it possible to create enterprise-wide Vault lists.

Similarly, you can copy a file using the following command:

```
ciexport tag=copy selname=objectName selscope=fileOrPart
vaultlst=listOfVaults
```

Please note: A copy has no persistent relationship to its original, unlike `replicate_read` or `replicate_write`. To make a write-replica, issue the following command:

```
ciexport tag=replicate_write selname=objectName
selscope=fileOrPart vaultlst=singleVaultName
```

If you execute this command as `DVADMIN`, the import manager process sends you a message indicating whether or not the file imported successfully. This allows you to track when a file is received at each remote Vault site. The import manager is a constant running Vault server process that runs on each Vault.

At the importing site the import manager server process enters the file or part into the Vault using standard Vault commands. Following an import operation, the import manager sends a message indicating the disposition of the import operation. This message is sent to the export user-ID at the exporting Vault.

Remember to use `DVADMIN` for export operations if you want to receive notifications of imports. Further, using an alias you can cause all messages sent to any administrator (`DVADMIN`) to be routed to a particular Vault, allowing you to centralize your messages to your home Vault.

As an alternative, you can declare a special user-ID for each project, accomplishing the same thing at the project level.

Please note: The `EXPORT` command does not support files with a password. Exporting a file with a password displays an error message and produces no effect on the database. Although you can set up a subscription to files with passwords, if the action is for an export of the file, the processing done is incorrect.

Export-Import Operational Recommendations

Using the Export Directory

Export functionality extracts and queues files and parts from the Vault in a staging area called the export directory, or `XD`. This directory is located in `$DATA_DIRECTORY/oaxis/xd`. You must ensure that this space is large enough to hold all of the files and parts that are exported but not yet imported. Note that if a file is exported to multiple Vaults, only one copy of that file is retained in the export directory. If you want, you can place this directory in its own file system and use a link, rather than allowing it to remain in its default location.

Using the Distribution Control Table

Each export operation produces one or more rows of data in the `DM_DISTRIBUTION_CONTROL` table. There is one row for each importer. As imports are successfully completed, the corresponding row is deleted from the table. Therefore, this table always contains an updated list of all outstanding imports.

In the `DM_DISTRIBUTION_CONTROL` table, the column named `DM_DIST_IMPSTATUS` is set to `READY` if the import has not been performed. It is set to `ERR` if an import was unsuccessful. When a file has an error status, the import manager process stops attempting to import it. Therefore, you must perform the import operation manually. To do this, sign on to the importing Vault, perhaps as `DVADMIN`. Then, attempt an import by supplying the necessary parameter information as follows:

```
ciimport tag=tagName exporter=exportingVaultID
selname=objectName selscope=fileOrPart revision=revNumber
```

This command generates an import audit file that can help you to determine the nature of the problem. Note that you must supply all of the parameters in order to perform a direct import. The import manager process issues the following command to each known Vault, for each known tag:

```
ciimport tag=tagName exporter=vaultID
```

If you omit the selection name, scope and revision, the exporting Vault selects the next available file or part (labeled `READY`) from the tag.

Please note: The import manager transfers files in uppercase or lowercase as entered. However, files are transferred in uppercase, if they are imported manually.

Keeping the Audit Files

The import manager server process writes one audit file each day, called `IMGR.AUDIT.yymmdd`, to the `/tmp` directory. This file contains a history of all attempted imports. You can copy this file (and optionally store it in the Vault) as a record of import operations.

Export and import operations are logged to a table called `DM_DISTRIBUTION_HIST`. The information in this table is not as complete as it is in the `IMGR.AUDIT` file, but it does provide a persistent record of all export and import activity. The information in `DM_DISTRIBUTION_HIST` consists of one row each for export and import, naming the objects and the Vaults.

Exporting a Write Replica

When you export a write-replica, the receiving Vault has the writable version of that file. To return this file, enter the following command:

```
ciexport tag=replace selname=objectName selscope=fileOrPart
```

No Vault list parameter is necessary because the export and import functionality tracks the source of all replicas. This data is stored in the `DM_REGISTRATION` table, alongside the other registration information.

Vault code does not allow anyone to delete read- or write-replica files from the system while the file is in a distributed state. Use the Export-Import feature to return the file to its originating Vault.

Please note: When a write-replica is returned to its home Vault, the import manager process at the home Vault issues `cireqrvw` (request review) for this item. Depending on how you have set up your project life cycle, the returned file could transition to a released state without any approvals. You can establish a specific status code sequence to account for this procedure. Provide the following transitional status code to allow the write-replica to go out as IW and be subject to review prior to attaining `RL: IW -> SUBCONTRACT -> RL`

If you want to continue making updates and need to update the source of the replica without returning the copy, issue the following command:

```
ciexport tag=update selname=objectName selscope=fileOrPart
```

This command sends the updated version to the home Vault. Be sure to monitor progress of exports and imports by routing messages from the import manager process to `DVADMIN`.

Using Subscriptions

Currently, distributed Vault supports the following subscribable actions:

- `sendmsgu`: sends a message to a user that a selected event occurred on a selected file or part.
- `sendmsgl`: sends a message to a user list that a selected event occurred on a selected file or part.
- `export_read_replica:` invokes an export action to export a read-only replica of a file to a list of Vaults.
- `sync_replica_source`: invokes an export action to refresh (update) the original of a previously write-replicated file.

Following are the events to which a user can subscribe:

```
change_object_metadata
change_object_binary
change_object_revision
change_object_status
```

You can, for example, use the `sendmsgu` and `sendmsgl` actions to send messages to automated clients that might issue occasional `readmsg` commands to obtain event notifications and then take site-specific action. You can use a user-ID (such as `DVADMIN)` to automatically receive these notifications and to make processing decisions. Following is an example for establishing a subscription:

```
ciaddsub subname=mysub, notifyid=dvadmin, notify=b,
selname=myfile, selscope=f,
eventid=change_object_status,
actionid=sendmsgl, userlist=projectlist
```

The action indicated by this subscription (`sendmsgl`) is executed when the status of `myfile` is changed, perhaps due to a review procedure. All of the users in `projectlist` will be notified as follows:

```
”from myvault: subscription from user dvadmin: the event
change_object_status has occurred to object: type f, name myfile,
revision a”
```

`DVADMIN` is notified as well. Having received this message, an automated process could, for example, issue `ciexport` to create read-only replica in other distributed Vaults.

Actions that are subscribable are typically repeatable in nature. An action that would create a write-replica is not repeatable because only one write-replica of an object can exist at any time.

After you enter a subscription, the following events occur:

1. An entry is made in `DM_SUBSCRIPTION`, identifying the subscription.
2. The Vault database triggers monitor all database changes against the item named in the subscription. Any event that correlates the subscription to the specific event is logged in `DM_EVENT_QUEUE`.
3. The event manager process queries `DM_EVENT_QUEUE`, and for each subscription creates an entry in `DM_ACTION_QUEUE`, places all of the pertinent parameters of the action into `DM_ACTION_DATA`, which is joined to `DM_ACTION_QUEUE` by `DM_ACTION_JOINKEY`.
4. When the action manager process queries `DM_ACTION_QUEUE` and discovers this new action, it moves the action to `DM_ACTMANAGER`, and spawns a service `AE` to execute the action.

Please note: When you create a subscription to notify a user list of a particular event, the subscription name does not appear in the Notify on Event window. However, the notification is created and works as designed.

A subscription on `CHANGE_OBJECT_REVISION` cannot be used on objects with `NONE` revision sequence.

Import Manager

The import manager is a process that runs on every distributed Vault, and has an AE name of `EDMIMGR`. This process extracts a list of possible exporters from the `DM_VAULT_CONFIG` table. This table contains the domain names of all other Vaults for which a given Vault has knowledge for the purpose of distribution.

The `EDMIMGR` process cycles through this list of Vaults issuing import commands for each possible tag name. If there is anything to import, the process would import it, or else go to the next tag name or to the next Vault.

The frequency of the `EDMIMGR` process can be controlled by setting up the `IMGR_EXECUTION_FREQUENCY` keyword for a specific value in the `nsm.config` file.

Terms

Audit: The `EDMIMGR` import manager process creates an audit file with a date stamp on a daily basis. This audit file contains the results of all imports, successful or failed, that take place.

Rules*:* All rules concerning import that apply to the command line import apply to the Import Manager as well.

Notification to Exporter: The import manager process sends a message to the user in the peer Vault who actually initiated the export after the success or a failure of an import operation.

Import Error

If an import fails at the importer because the object is not in a required state or the importing domain is not an authorized user or for any other internal reason, the exporter is informed and the exported object is put on an ERR state.

Under these conditions, if the reasons for failure are correctable at the importer, then the corrections can be done and the object can be imported. The import manager will not import an object whose tag is in the ERR state at the exporter. But command line import will succeed if the exact name, type, and revision of the required object are specified.

Graphical User Interface

The Export-Import services can be used throughout the graphical user interface (GUI). There is no selectable command in the GUI to import an object. The import manager process imports the exported objects in each of the importing Vaults.

DOD View

The Export command can be used under the distributed object directory (DOD) view in the GUI. To reach the DOD view, select the Distributed > Files or Distributed > parts command from the View menu.

Distribute Now

In the DOD view, the Distribute Now action can be selected under the Admin menu. This action displays a menu of intents for the object and a Vault list. When you click on Apply or OK after selecting the intent and the Vault list, the export command acts on the selected object.

Following are the two commands that achieve the Distribute Now action:

```
ciexport selname=objectName selscope=fileOrPart
revision=revisionNumber Vaultlst=listOfImportingVaults tag=tagName
ciimport selname=objectName selscope=fileOrPart
revision=revisionNumber exporter=exportingVault tag=tagName
```

Please note: `revision` is optional.

Following is an alternative `ciimport` command:

```
ciimport exporter=exportingVault tag=tagName
```

The latter form of the import command imports the next available object for this given Vault under the given tag name.

Chapter 4

Locally Importing Parts or Files

This chapter describes the local Import feature in the Distributed Vault environment.

- Introduction
- Configuration
- Importing Automatically

Introduction

The local import feature imports local (nonnetwork) files and parts between the participating Vaults in a Distributed Vault environment.

Local import fetches importable files or parts from a local import directory and not from the export directory on the exporting Vault.

The local import process is as follows:

1. The Distributed Vault administrator exports the files or parts.
2. The Distributed Vault administrator extracts the entries of files from the export directory for a local import. These entries are delivered to the importing Vault by means of a tape.
3. The Distributed Vault administrator at the import site extracts these items from the tape and places them in a predefined local import directory.
4. At the import Vault site, the Import Manager checks the exporter Vaults for entries to be imported.
5. The Import Manager receives confirmation of files or parts to be imported from the exporting site and checks the locally configured import directory for these items. If available, the files or parts are copied into the working directory of the Import Manager.

Please note: Files or parts not found locally are fetched from the exporting site.

6. The `IMGR.AUDIT` file includes messages whenever an item is used from the local import directory. The `ViSiTeD` file is updated in the local import directory with the names of all locally imported files or parts.

   The `ViSiTeD` file helps the Distributed Vault administrator determine when these entries are cleared from the local import directory.

Configuration

To perform local import operations, you must configure the local import directory. When performing import operations, both Import Manager and OAXIS look for a directory at `$OAXIS_PATH/id`.

By default, the local import directory is located at `$OAXIS_PATH/id.`

Please note: The `$OAXIS_PATH/id` path can be changed by altering both the `OAXIS.STARTUP` and `IMGR.STARTUP` scripts.

During the import operation, `IMGR.STARTUP` and `OAXIS.STARTUP` scripts checks for a subdirectory with the name of the exporting Vault.

To perform local import operations, configure a local import directory `$OAXIS_PATH/id/EXPVAULT` at the importing Vault site. Here, `EXPVAULT` is a Vault domain. A subdirectory of this type must exist for each exporting Vault from which you can perform the local import operation. The locally importable files and parts reside in this subdirectory.

The export directory (`replicate_read`) is organized as follows:

```
$OAXIS_PATH/xd/replicate_read/<n>
```

where `<n>` is a subdirectory containing a single exported item. Each `<n>` subdirectory is uniquely numbered with a `_HEADER` file. The `_HEADER` file holds the next available number to be used in this subdirectory.

To enable local import operations, configure a subdirectory for each export queue in the `EXPVAULT` local import directory. By default six sub-directories are available within `$OAXIS_PATH/xd`. They are:

- `copy`
- `move`
- `replicate_read`
- `replicate_write`
- `update`
- `replace`

Please note: It is not necessary to:

- Configure all export queues locally. Configure only the required tags. The others operate in the normal mode.
- Make all items within a tag available for local import.

For example, set the following path to configure local import on the `replicate_read` tag of the `EXPVAULT` domain:

```
$OAXIS_PATH/id/EXPVAULT/replicate_read
```

This completes the configuration of the local import directory. The other subdirectories can be added later.

Example

If `EXPVAULT` and `IMPVAULT` are Vault domains, enter the following at the exporting site to export the `testcase` file from `EXPVAULT` to `IMPVAULT` as a read-only replica:

```
ciregister selname=testcase selscope=f reglevel=r

ciexport selname=testcase selscope=f
tag=replicate_read Vaultlst=IMPVAULT
```

After the export operation, combine the `testcase` file with the Distributed Vault control files and place them on a tape. Send this tape to the `IMPVAULT` site. The Distributed Vault administrator places the files in the predefined local import directory. To do so, determine the exact location of the `testcase` export subdirectory.

If the `testcase` export subdirectory is located within `$OAXIS_PATH/xd/replicate_read/<n>` and you have performed the last export operation to the `replicate_read` queue, then `<n>` is the highest number entry in the `$OAXIS_PATH/xd/replicate_read` subdirectory.

You can obtain this information by performing the following SQL query:

```
select dm_dist_local_name from dm_distribution_control where
dm_dist_selname = testcase
dm_dist_dist_selscope= FILE
dm_dist_importer = IMPVAULT
dm_dist_tagname = replicate_read;
```

This query results in the exact location of the subdirectory that contains the exported file. In this case, the result is:

`$OAXIS_PATH/xd/replicate_read/381/`

where `381` is the subdirectory containing the `testcase` exported file.

Copy the `381` subdirectory to the tape, and send it to the importing site. If you want to locally import several items, copy all the subdirectories as well.

Please note: Do not move the `381` subdirectory to the tape. Copy it.

At the importing site, the extracted subdirectory is copied to the local import directory. Therefore the resulting path is:

`$OAXIS_PATH/id/EXPVAULT/replicate_read/381/`

When the Import Manager checks `EXPVAULT` for the `replicate_read` queue, it receives an indication that the `381` subdirectory is ready to be imported. The Import Manager checks the local import directory and derives the `381` subdirectory locally if it exists. Import Manager then copies the files containing the `381` and creates the `ViSiTeD` file.

The `ViSiTeD` file contains a list of all the elements of the `381` subdirectory that are derived locally. The `ViSiTeD` file is located at:
`$OAXIS_PATH/id/EXPVAULT/replicate_read/381/ViSiTeD`

The `IMGR.AUDIT` file indicates the use of the local versions in its report.

Please note: The Import Manager does not remove the locally imported files or parts from the local import directory. That is the responsibility of the Distributed Vault administrator. The corresponding files at the export side are however cleaned up.

Importing Automatically

An exported item can be imported by the Import Manager.

If an item is imported so that it can be combined with the Distributed Vault control files, the Import Manager must import that item from the remote tag before the tape is mailed to the importing site. To avoid this situation, do one of the following:

- Allow the Import Manager to install the items on the local import directory of the importer. Doing this prevents all automated importing activity because the Import Manager is required for automated import.

  You can perform manual imports by using the `ciimport` command line interface. The `ciimport` command is identical to the Import Manager except that `ciimport` imports only one item.

  For more information on importing parts and files remotely refer to page 3-7 of Chapter 3, "Remote Exporting and Importing of Parts or Files."
- Create a new tag or queue. A tag is a name used to define an export queue, which contains exported objects queued up to be imported by their importing Vaults. A tag defines the purpose, or intent, of export operations that use the tag.

Creating a New Tag

This section describes how to create a new customized tag.

On the Exporting Site

The `$OAXIS_PATH/tags` directory contains a set of tag files, one file for each configured queue. The files are `replicate_read`, `replicate_write`, `copy`, `move`, `update`, and `replace`. Each file contains a single line that specifies the intent of exports which occurs on that tag.

For example, if you display the file `$OAXIS_PATH/tags/replicate_read`, you will see the following:

```
INTENT=REPLICATE_FOR_READ
```

Items exported to this tag are processed according to this intent. A read-only replica can be read at the receiving site. There can be multiple receivers for a read-only replica. Select one of the existing tags, depending on the intent for which you want assign a custom queue, and create a new tag with similar intent.

For example, to create a new tag called `batch_replicate_read` whose export or import semantics are the same as the existing `replicate_read` tag, do the following:

1. Copy the following tags:
   - `$OAXIS_PATH/xd/tags/replicate_read`
   - `$OAXIS_PATH/xd/tags/batch_replicate_read`
2. Create the `batch_replicate_read` directory

   ```
   mkdir $OAXIS_PATH/xd/batch_replicate_read
   ```
3. Define the `_HEADER` file as follows:

   ```
   echo 0 > $OAXIS_PATH/xd/batch_replicate_read/_HEADER
   ```

   These steps create the export directory structures required to enable a Distributed Vault user on the exporting Vault.
4. Run the following:

   ```
   ciexport selname=testcase selscope=f
   tag=batch_replicate_read Vaultlst=IMPVAULT
   ```

On the Importing Site

To manually configure a new customized tag, do the following:

1. Perform steps 1 to 3, as shown in the preceding section to configure the importing site.
2. Configure the local import directory so that it can perform local import operations. Enable only `EXPVAULT` for local imports by creating the `batch_replicate_read` directory:

   ```
   mkdir $OAXIS_PATH/id/EXPVAULT/batch_replicate_read
   ```

   Doing this enables you to do the following:
   - Export items to the `batch_replicate_read` queue, at the exporting side
   - Manually import items from the `batch_replicate_read` queue of the exporting Vault.

Please note: Automated import requires the following additional configuration step.

3. Perform the following query so that the Import Manager can automatically import from the `batch_replicate_read` queue:

   ```
   insert into dm_dist_tag values
   ('batch_replicate_read', 'REPLICATE_READ');
   ```

Performing Automated Import

Import Manager performs automated import as follows:

1. The Import Manager determines if the exporting Vault is accessible by performing the following query:

```
select dm_Vault_id from dm_Vault_config
where dm_Vault_type = 'V' and dm_Vault_status = 0;
```

Please note: The exporting Vault should be accessible to the importing Vault at the time of import.

DM_Vault_STATUS is maintained by the Action Manager, which periodically checks the remote Vaults and sets the `dm_Vault_status` accordingly.

2. The Import Manager then determines the set of tags to be imported automatically by performing the following query:

```
select dm_dist_tagname from dm_dist_tag;
```

This query produces a list of tags or queues against which import functions are performed. The list of tags is as follows:

```
SQL> select * from dm_dist_tag;
DM_DIST_TAGNAME DM_DIST_INTENT
--------------------------------
copy COPY_OBJECT
move MOVE_OBJECT
replicate_read REPLICATE_READ
replicate_write REPLICATE_WRITE
update UPDATE_ORIGINAL
replace RETURN_ORIGINAL
6 rows selected.
```

Please note: Do not modify the existing content of `dm_dist_tag`. You can, however, add other rows if required.

3. Perform the following query so that the Import Manager can automatically import from the `batch_replicate_read` queue:

```
insert into dm_dist_tag values
batch_replicate_read, REPLICATE_READ;
```

The result of this query is as follows:

```
SQL> select * from dm_dist_tag;
DM_DIST_TAGNAME DM_DIST_INTENT
--------------------------------
copy COPY_OBJECT
move MOVE_OBJECT
replicate_read REPLICATE_READ
replicate_write REPLICATE_WRITE
update UPDATE_ORIGINAL
replace RETURN_ORIGINAL
```

```
batch_replicate_read REPLICATE_READ
7 rows selected.
```

4. The Import Manager processes the `batch_replicate_read` tag.

Please note: If the `batch_replicate_read` row is removed from the `dm_dist_tag`, automated import will not be available for that queue.

Appendix A

Distributed Vault Database Tables

Appendix A lists database tables required for Distributed Vault.

- DM_OBJECT_DIRECTORY
- DM_VAULT_CONFIG
- DM_VAULT_LIST
- DM_ACTION_TEMPLATE
- DM_ACTTEMPLATE_SUB
- DM_EVENT_CODE
- DM_ACTION_QUEUE
- DM_ACTION_DATA
- DM_ACTMANAGER
- DM_EVENT_QUEUE
- DM_REGISTRATION
- DM_SUBSCRIPTION
- DM_SUBSCRIPT_SUB
- DM_USER_ALIAS
- DM_DIST_TAG
- DM_DISTRIBUTION_HIST
- DM_DISTRIBUTION_CONTROL

DM_OBJECT_DIRECTORY

This table resides in the Distributed Object Directory. The fields below are the same as those described in the Database manual for the `DM_FILE_DIRECTORY` or `DM_PART_DIRECTORY`. Exceptions are noted.

Table A-1 The DM_OBJECT_DIRECTORY Table

Field	Type	Description
`DM_OBJ_VAULT_ID`	VARCHAR(32)	source vault (domain)
`DM_OBJ_TYPE`	VARCHAR(2)	P=part, F=file
`DM_OBJ_NAME`	VARCHAR(80)	
`DM_OBJ_REV`	NUMBER(5)	
`DM_OBJ_CHAR_REV`	VARCHAR(20)	
`DM_OBJ_VERSION`	NUMBER(10)	
`DM_OBJ_PARENT`	VARCHAR(10)	part_rowid from dm_part_file or 0
`DM_OBJ_SCOPE`	VARCHAR(1)	registration for Q (query), R (read), W (write), O (ownership)
`DM_OBJ_CLASS`	VARCHAR(3)	
`DM_OBJ_OWNER_ID`	VARCHAR(12)	
`DM_OBJ_STATUS_CD`	VARCHAR(8)	
`DM_OBJ_SUB_TYPE`	VARCHAR(8)	dm_file_type or dm_part_type
`DM_OBJ_SYS_TYPE`	VARCHAR(8)	
`DM_OBJ_USER_TYPE`	VARCHAR(8)	
`DM_OBJ_PART_NO`	VARCHAR(20)	
`DM_OBJ_DESC`	VARCHAR(25)	
`DM_OBJ_CLASS_CD`	VARCHAR(32)	
`DM_OBJ_NAME_ROWID`	VARCHAR(10)	
`DM_OBJ_ROWID`	VARCHAR(10)	
`DM_OBJ_SYSTEM_CD`	VARCHAR(1)	
`DM_OBJ_XCTN_ID`	VARCHAR(10)	
`DM_OBJ_USER_ID`	VARCHAR(12)	
`DM_OBJ_NODE_NAME`	VARCHAR(32)	
`DM_OBJ_SIZE`	NUMBER(10)	if part = sum of file's sizes
`DM_OBJ_CR_DATE`	VARCHAR(6)	
`DM_OBJ_CR_TIME`	VARCHAR(6)	
`DM_OBJ_CR_ID`	VARCHAR(12)	
`DM_OBJ_UPDT_DATE`	VARCHAR(6)	

Table A-1 The DM_OBJECT_DIRECTORY Table

Field	Type	Description
DM_OBJ_UPDT_ID	VARCHAR(12)	
DM_OBJ_DATE	VARCHAR(6)	
DM_OBJ_TIME	VARCHAR(6)	
DM_OBJ_UPDATER	VARCHAR(12)	
DM_OBJ_UPDT_TIME	VARCHAR(6)	
DM_OBJ_REP_TYPE	VARCHAR(2)	blank=not a replica; IR: read replica; IW: write replica; (available for updates) OW: source of a write replica (the original from which the write replica was made) OM: transient type which indicates this file/part is in the process of being moved
DM_OBJ_SRC_VAULT	VARCHAR(32)	if DM_OBJ_REP_TYPE = IR or IW this field identifies source of the replica if DM_OBJ_REP_TYPE = OM or OW this field identifies target of write-replica or move
DM_OBJ_SRC_NAME	VARCHAR(80)	name on source vault
DM_OBJ_SRC_CHAR_REV	VARCHAR(20)	char rev on source vault
DM_OBJ_VAULT_ID	VARCHAR(32)	source vault (domain)
DM_OBJ_TYPE	VARCHAR(2)	P=part, F=file
DM_OBJ_NAME	VARCHAR(80)	
DM_OBJ_REV	NUMBER(5)	
DM_OBJ_CHAR_REV	VARCHAR(20)	

DM_VAULT_CONFIG

The `DM_VAULT_CONFIG` table stores information about an object's source vault, the Distributed Object Directories, and remote vaults that are known to the source vault.

The `DM_VAULT_CONFIG` table must have one entry for the source vault (S=self), the Distributed Object Directory (E=event manager), and any other vault from which the source vault can send and/or receive data.

Table A-2 DM_VAULT_CONFIG Table Entries

Field	Type	Description
`DM_VAULT_ID`	VARCHAR(32)	the vault's id - as in pm.config

`DM_VAULT_TYPE`	VARCHAR(1)	S=Self, i.e. this vault
		(only one of this type allowed),
		V=(full) remote vault,
		D=Distributed Object Directory
		to use for locate function. Only one of this type allowed)
		O = Other DOD to which registered objects can be distributed for future use

`DM_VAULT_CONNECT`	VARCHAR(1)	R=read W=write B=both 0=No actions are currently in progress with respect to the indicated VAULTID 1=Actions are currently in progress for the indicated VAULTID
`DM_VAULT_STATUS`	VARCHAR(1)	0=On-line, 1=offline >1=unavailable
`DM_VAULT_DISTANCE`	NUMBER(5)	Reserved for future use
`DM_VAULT_SEQNO`	NUMBER(5)	Search order (asc) of vaults when processing transparent file/part access
`DM_VAULT_UPDT_DATE`	VARCHAR(6)	Date of the action manager's last successful update of this remote vault.
`DM_VAULT_UPDT_TIME`	VARCHAR(6)	Time of the action manager's last successful update of this remote vault.
`DM_VAULT_DATE`	VARCHAR(6)	Date of the action manager's last attempted update of this remote vault.
`DM_VAULT_TIME`	VARCHAR(6)	Time of the action manager's last attempted update of this remote vault.
`DM_VAULT_RETRYTIME`	NUMBER (38)	TimeStamp of next ONLINE/OFFLINE check to this remote VAULT/DOD This value indicates a time of not more than 30 minutes from last ONLINE/OFFLINE check

DM_VAULT_LIST

The DM_VAULT_LIST table provides a method of grouping several vaults together. The list may include any vault or Distributed Object Directories as identified in the DM_VAULT_CONFIG.

Table A-3 DM_VAULT_CONFIG Table Entries

Field	Type	Description
DM_LIST_ID	VARCHAR(24)	a unique vault list name
DM_VAULT_ID	VARCHAR(32)	a valid vault id in the DM_VAULT_CONFIG table

DM_ACTION_TEMPLATE

The DM_ACTION_TEMPLATE table stores actions to which you can subscribe. The SUBSCRIBE command uses this table to validate the actions to which you have subscribed.

Table A-4 DM_ACTION_TEMPLATE Table Entries

Field	Type	Description
DM_ACTION_NAME	VARCHAR(32)	Name of action template

DM_ACTION_TYPE	VARCHAR(8)	DOD=Distribute to DOD;
		EDM=EDM CI command in DM_ACTION_TEXT field

DM_ACTION_TEXT	VARCHAR(240)	Text of action (e.g. EDM ci command). The text may contain substitutions, e.g. &USERID for a userid. All substitutions are either system defaults (like reserved words) or user-supplied at the time of subscription. User substitutions are defined in the DM_ACTTEMPLATE_SUB table. The system defaults are: EVENTYPE = I/U/D EVENTID = name of triggering event ACTIONID = this action name OBJTYPE = triggering object's type OBJNAME = triggering object's name CHARREV = triggering object's character revision, if applicable SUBNAME = subscription name SUBOWNER = subscriber's name
DM_ACTION_DESC	VARCHAR(240)	Optional description of the action template.

DM_ACTTEMPLATE_SUB

The DM_ACTPLAN_SUB table stores user-supplied substitutions required for action templates. The SUBSCRIBE command checks this table to see if all the required substitutions have been entered for the chosen action template.

Table A-5 DM_ACTTEMPLATE_SUB Table Entries

Field	Type	Description
DM_ACTION_NAME	VARCHAR(32)	Name of action template.
DM_ACTION_SUB_NAME	VARCHAR(32)	Name of a user-supplied substitution parameter required for a subscription when using this action template. If the action is an EDM command, the substitution will appear in the message text as, for example, &USERID. The'&' that appears in the action text is not stored here.

DM_ACTION_SUB_APPLY	VARCHAR(1)	When is substitution applied:
		E=when event occurs;
		S=when subscription entered

DM_ACTION_SUB_TYPE	VARCHAR(32)	Type of data the parameter is -- usually, the same as the DM_ACTION_SUB_NAME. For example, an action template substitution &USERID is of type USERID. A substitution of &NOTIFYID is also of type USERID.
DM_ACTION_SUB_REQ	VARCHAR(1)	Is parameter required? Y=yes, and key s/b in database N=no, and key must not be in database R=yes (required) but no database checks blank = not required

DM_EVENT_CODE

The DM_EVENT_CODE table stores the names of events that can be subscribed to. This table is used to verify the event code supplied with the SUBSCRIBE command.

Table A-6 DM_EVENT_CODE Table Entries

Field	Type	Description
DM_EVENT_NAME	VARCHAR(32)	Name of event code. The same event can apply to multiple object types, and the event name appears once for each object type to which it applies.

DM_OBJ_TYPE	VARCHAR(2)	Type of object that can use this event:
		F=file, P=part,
		U=user, M=user list member

DM_EVENT_DESC	VARCHAR(240)	Description of event code.

DM_ACTION_QUEUE

This table stores the actions to perform as a result of logged events. It is created by the event manager and by certain other transactions.

Table A-7 DM_ACTION_QUEUE Table Entries

Field	Type	Description
DM_ACTION_NAME	VARCHAR(32)	Joins to DM_ACTION_NAME in DM_ACTION_TEMPLATE
DM_ACTION_TYPE	VARCHAR(8)	DOD=Distribute to DOD; EDM=EDM CI command in DM_ACTION_TEXT field
DM_ACTION_XCTN_ID	VARCHAR(10)	Event Manager transaction id
DM_ACTION_JOINKEY	VARCHAR(10)	Joins to DM_ACTION_DATA

DM_ACTION_DATA

This table stores miscellaneous data items associated with the action in the DM_ACTION_QUEUE table.

Table A-8 DM_ACTION_QUEUE Table Entries

Field	Type	Description
DM_ACTION_JOINKEY	VARCHAR(10)	Joins to DM_ACTION_QUEUE
DM_ACTION_KEYWORD	VARCHAR(32)	Name of a data item pertaining to subscription, event, etc.
DM_ACTION_DATA	VARCHAR(240)	Value of keyword.

DM_ACTMANAGER

This table maintains the action manager control information. Table entries are deleted when action (and follow-up) is completed or when retry count exceeds the permissible limit.

Table A-9 DM_ACTION_QUEUE Table Entries

Field	Type	Description
`DM_ACTION_NAME`	VARCHAR(32)	Joins to name in `DM_ACTION_TEMPLATE`

`DM_ACTION_TYPE`	VARCHAR(8)	DOD=Distribute to DOD;
		EDM=EDM CI command in
		`DM_ACTION_TEXT` field

`DM_ACTION_JOINKEY`	VARCHAR(10)	Joins to `DM_ACTION_DATA`
`DM_ACTION_VAULTID`	VARCHAR(32)	Vault initiating the action
`DM_ACTION_TIME`	VARCHAR(6)	Time action query
`DM_ACTION_DATE`	VARCHAR(6)	Date action query
`DM_ACTION_STATUS`	NUMBER(5)	0= action is dispatchable 1= action has been launched, i.e. service AE started 2= restartable 3= completed 4=retry count exceeded
`DM_ACTION_RETRIES`	NUMBER(5)	Number of retries performed
`DM_ACTION_RETRYTIME`	NUMBER(5)	Next retry time (not in user-visible form)

DM_EVENT_QUEUE

This table stores the queue of logged (recorded) events that are subscribed to. The database triggers populate the event queue for Event Manager use.

Table A-10 DM_EVENT_QUEUE Table Entries

Field	Type	Description
`DM_SUB_NAME`	VARCHAR(24)	Which subscription initiated event logging
`DM_OBJ_JOINKEY`	VARCHAR(24)	FILE/PART NAME_ROWID
`DM_OBJ_TYPE`	VARCHAR(2)	F=file, P=part, U=user
`DM_EVENT_TYPE`	VARCHAR(1)	I=insert, U=update, D=delete
`DM_OBJ_XCTN_ID`	VARCHAR(10)	EDM transaction which causes event. Correspond to `DM_XCTIN_ID` column of `DM_XCTN_CONTROL` table

DM_REGISTRATION

The DM_REGISTRATION table records information for registered objects.

Table A-11 DM_REGISTRATION Table Entries

Field	Type	Description

DM_OBJ_JOINKEY	VARCHAR(24)	Parts/Files= name_rowid
		Users = userid
		Project = project id

DM_OBJ_TYPE	VARCHAR(2)	F=file,
		P=part

DM_OBJ_SCOPE	VARCHAR(1)	Scope of external access granted:
		For parts and files:
		Q=query
		R=read
		W=write
		O=own
		By default, any object with no entry in the registration table has a scope of L=local only.

DM_REG_LIST	VARCHAR(24)	Name of vault list containing id's of DOD's that are to list this object. Default value is "LOCALDOD"
DM_VAULT_ID	VARCHAR(32)	the Exporting Vaultid when dist_reason is IR, IW. the Importing Vaultid when dist_reason is OW or OM.
DM_DIST_REASON	VARCHAR(2)	The object referred to here is; IR: Imported as Read Replicate IW: Imported as Write Replicate OW: Exported as Write Replica OM: Exported - Moveownership blank: not under distribution control

DM_SUBSCRIPTION

The DM_SUBSCRIPTION table stores subscriptions made to object events.

Table A-12 DM_SUBSCRIPTION Table Entries

Field	Type	Description
DM_SUB_NAME	VARCHAR(24)	A unique name assigned by the subscription user.

DM_OBJ_TYPE	VARCHAR(2)	Type of object the subscription is on:
		P=part
		F= file
		J = project definition
		A=all parts and files in a project
		U= user
		M=user member of userlist

DM_OBJ_NAME	VARCHAR(80)	Object name (or *)
DM_OBJ_CHAR_REV	VARCHAR(20)	Object Character revision (or *)
DM_EVENT_NAME	VARCHAR(32)	Event code (as in event table)
DM_ACTION_NAME	VARCHAR(32)	Action template (as in action template table)
DM_SUB_EXPDATE	VARCHAR(6)	Expiration date of subscription. Format is YYMMDD. It is blank if no expiration date
DM_SUB_EXPTIME	VARCHAR(6)	Expiration time of subscription. Format is HHMMSS
DM_SUB_EXPCNT	NUMBER(5)	Expiration count of subscription. Zero if no expiration count
DM_SUB_USECNT	NUMBER(10)	Number of uses this subscription has had

DM_SUB_NOTIFY	VARCHAR(1)	Notify an EDM user if:
		S = subscription succeeds;
		F = subscription fails;
		B = both success or failure
		N = neither (no notify done)

DM_SUB_NOTIFYID	VARCHAR(12)	EDM user id to get notification.
DM_SUB_DATE	VARCHAR(6)	Date that subscription entered in YYMMDD format.
DM_SUB_TIME	VARCHAR(6)	Time that subscription entered in HHMMSS format.
DM_SUB_OWNER	VARCHAR(12)	User who entered the subscription

DM_SUBSCRIPT_SUB

The DM_SUBSCRIPT_SUB table stores the substitutions needed for subscriptions made to object events.

Table A-13 DM_SUBSCRIPTION Table Entries

Field	Type	Description
DM_SUB_NAME	VARCHAR(24)	Joins to subscription.

DM_SUB_TYPE	VARCHAR(1)	Type of substitution:
		A = action template
		E = event code (unused)

DM_SUBSUB_NAME	VARCHAR(32)	The name of a substitution parameter for the action template.
DM_SUBSUB_VALUE	VARCHAR(240)	The value of the named action parameter substitution.

DM_USER_ALIAS

The `DM_USER_ALIAS` table specifies the home vault for a user . The user messages are sent to the home vault.

Table A-14 DM_USER_ALIAS Table Entries

Field	Type	Description
`DM_USER_ID`	VARCHAR(12)	A valid user id in the `DM_USER` table.
`DM_VAULT_ID`	VARCHAR(32)	A valid vault id in the `DM_VAULT_CONFIG` table.

DM_DIST_TAG

The `DM_DIST_TAG` table contains the set of valid tags for import and export operations.

Table A-15 DM_DIST_TAG Table Entries

Field	Type	Description
`DM_DIST_TAGNAME`	VARCHAR(32)	Tagname: The set of valid tagnames. The tags are loaded at install. Fixed for first release
`DM_DIST_INTENT`	VARCHAR(32)	Purpose of export or import

DM_DISTRIBUTION_HIST

The `DM_DISTRIBUTION_HIST` table maintains a history of all the export and import operations.

Table A-16 DM_DISTRIBUTION_HIST table entries

Field	Type	Description
`DM_EXP_OBJ_TYPE`	VARCHAR(2)	Object type: F (file), P (part), U (user), etc.
`DM_EXP_OBJ_NAME`	VARCHAR(80)	Source object name
`DM_EXP_OBJ_REV`	VARCHAR(20)	Object revision
`DM_EXP_OBJ_ROWID`	VARCHAR(10)	Row id
`DM_EXP_OBJ_CLASS`	VARCHAR(3)	Object classification of PUB (public), PRI (private), or PRO (project)
`DM_EXP_OBJ_OWNER_ID`	VARCHAR(12)	User id, project id, or blank
`DM_EXP_OBJ_STATUS_CD`	VARCHAR(8)	Object's status code in the source vault
`DM_IMP_OBJ_NAME`	VARCHAR(80)	Destination object name
`DM_IMP_OBJ_REV`	VARCHAR(20)	Destination revision
`DM_IMP_OBJ_ROWID`	VARCHAR(10)	Destination row id
`DM_IMP_OBJ_CLASS`	VARCHAR(3)	Destination classification PUB (public), PRI (private), PRO (project)
`DM_IMP_OBJ_OWNER_ID`	VARCHAR(12)	Destination user id, project id, or blank
`DM_IMP_OBJ_STATUS_CD`	VARCHAR(8)	Destination status code
`DM_DIST_TAGNAME`	VARCHAR(32)	Tagname used when export performed
`DM_DIST_TYPE`	VARCHAR(2)	X (export) or I (import)
`DM_DIST_REASON`	VARCHAR(2)	The object referred to here is being: IR: Imported as Read Replicate IW: Imported as Write Replicate OW: Exported as Write Replica OM: Exported - Moveownership
`DM_DIST_VAULT_ID`	VARCHAR(32)	the Exporting Vaultid when dist_reason is IR, IW; the Importing Vaultid when dist_reason is OW or OM.
`DM_DIST_USER_ID`	VARCHAR(12)	User id performing the import or export
`DM_DIST_DATE`	VARCHAR(6)	Date and time of import or export
`DM_DIST_TIME`	VARCHAR(6)	

DM_DISTRIBUTION_CONTROL

The `DM_DISTRIBUTION_CONTROL` table contains an entry for each importer of an exported object. It indicates the name of the exporter, importer, and import status.

Table A-17 DM_DISTRIBUTION_CONTROL entries

Field	Type	Description
`DM_DIST_TAGNAME`	VARCHAR(32)	User-defined tagname
`DM_DIST_SELNAME`	VARCHAR(80)	Selection name of this aggregate
`DM_DIST_SELSCOPE`	VARCHAR(32)	Selection scope of this aggregate — badly named
`DM_DIST_REVISION`	VARCHAR(20)	A character revision code — inconsistently named
`DM_DIST_IMPORTER`	VARCHAR(32)	A vault id of an importing vault
`DM_DIST_IMPSTATUS`	VARCHAR(5)	How the import is progressing: need explicit values here.
`DM_DIST_TAPE_ID`	VARCHAR(6)	Reserved for future use
`DM_DIST_TAPE_SEQNO`	NUMBER(5)	Reserved for future use
`DM_DIST_NODE_NAME`	VARCHAR(32)	Reserved for future use
`DM_DIST_LOCAL_NAME`	VARCHAR(240)	Holding area for object metadata.

Index