VMS Software

VSI Data Dictionary (VDD) and Oracle CDD/Repository on OpenVMS

User-Facing Differences Guide

Publication Date: July 2026

Operating Systems: VSI OpenVMS x86-64 Version 9.2-3 or higher
VSI OpenVMS IA-64 Version 8.4-2L1 or higher

VMS Software, Inc. (VSI)
Boston, Massachusetts, USA

VSI Data Dictionary (VDD) and Oracle CDD/Repository on OpenVMS User-Facing Differences Guide

Legal Notice

Table of Contents

1. Introduction

VSI Data Dictionary (VDD) is a native, modern data repository for OpenVMS designed as a replacement for the Oracle Common Data Dictionary (CDD) to ensure seamless application compatibility across modern hardware platforms. VDD is supported on OpenVMS x86-64 and IA-64 systems. Because VDD implements the same user-visible interfaces and parameters as CDD, the majority of behavior is identical by design.

For users familiar with CDD, this document outlines the key user-facing differences between CDD and VDD. The following sections describe areas where both products provide comparable functionality, but differ in behavior, implementation, or usage.

Note

CDD features not yet implemented in VDD are deliberately excluded from this document. New features and improved functionality will be included in future releases of VDD.

Similarly, internal implementation details will not be covered, except for where they surface as user-visible differences.

For more information on VDD, refer to the *VSI Data Dictionary Installation and User Guide* [https://docs.vmssoftware.com/vsi-data-dictionary-install-and-user-guide/] and *VSI Data Dictionary X1.2-03 Release Notes* [https://docs.vmssoftware.com/vsi-data-dictionary-version-x1-2-03-release-notes/] or contact VSI.

2. Installation

The installation and environment setup procedures for VDD mirror those of CDD, but use VDD$-prefixed command procedures. The steps are as follows:

1. Install the PCSI kit with the following command:

   ```
   $ PRODUCT INSTALL VDD
   ```

2. Set up the system environment by running the following command procedure:

   ```
   $ @SYS$STARTUP:VDD$STARTUP.COM
   ```

3. Set up the user environment by running the following command procedure:

   ```
   $ @SYS$MANAGER:VDD$SETUP_USER_ENV.COM
   ```

3. Library Linking

The redirection of the CDD logical names is automatic. After VDD$SETUP_USER_ENV.COM is run, the CDD logical names will already point to the VDD images. For example:

```
$ SHOW LOG CDDSHR
"CDDSHR"    = "SYS$LIBRARY:VDDSHR.EXE;"     (LNM$PROCESS_TABLE)
$ SHOW LOG CDDLIBSHR
"CDDLIBSHR" = "SYS$LIBRARY:VDDLIBSHR.EXE;" (LNM$PROCESS_TABLE)
```

- VDDSHR.EXE replaces CDDSHR.EXE (the runtime shared library).

- VDDLIBSHR.EXE replaces CDDLIBSHR.EXE (the compiler stub library).

Layered products and custom applications that use CDD APIs transparently use the new VDD library, which exports the same set of symbols and implements equivalent runtime behavior. A user does not need to re-link CDD-built applications, as they pick up VDD through the redirected CDDSHR/CDDLIBSHR logical names.

4. Logical Names for Paths

VDD defines several VDD$-prefixed logical names and also recognizes a subset of the corresponding CDD$ logical names for compatibility. These logical names control how path names are resolved during dictionary operations.

When both VDD$ and CDD$ versions of a logical name are defined, the VDD$ logical name takes precedence.

The following logical names are supported:

VDD$ANCHOR

Defines the repository-root anchor. In CDD, this role is performed by CDD$COMPATIBILITY. CDD$COMPATIBILITY is **not supported** by VDD and is ignored if defined.

If VDD$ANCHOR is not defined, the dictionary file defaults to SYS$COMMON:[SYSTEMREPO].

VDD$DEFAULT

Defines the default directory prefix. It is equivalent to the CDD logical name CDD$DEFAULT.

Both VDD$DEFAULT and CDD$DEFAULT are recognized. If both are defined, VDD$DEFAULT takes precedence.

CDD$TOP

Defines the symbolic origin (repository root). It is recognized by VDD and retains its role from CDD. No VDD-specific equivalent is defined.

Note

VDD's Datatrieve setup separately points VDD$DEFAULT at a "system repository" located in SYS$COMMON:[SYSTEMREPO]. This location is a specific deployment's repository location, not the anchor-resolution default, which is SYS$COMMON:[CDDPLUS]. These locations serve different purposes and should not be considered equivalent.

For single-repository configurations, VDD behavior is comparable to CDD. The primary difference is the use of VDD$ logical names in place of unsupported or lower-priority CDD$ logical names.

5. Included Utilities

CDD includes the following three utility tools:

- **CDO (Common Dictionary Operator)**

A command language and interactive utility used to define, manage, and query metadata objects within the data dictionary repository.

- **DMU (Dictionary Management Utility)**

  A utility used to maintain and manage the contents and structure of the data dictionary repository, including performing administrative and batch operations.

- **CDDL (Common Data Dictionary Data Definition Language)**

  A data definition language used to declare structured metadata definitions (such as fields and records), which are processed by the CDDL compiler and loaded into the repository.

VDD includes the following three utility tools:

- **VDO (VDD Operator)**

  A VDD operator utility, analogous to CDO.

- **VDMU (VSI Dictionary Management Utility)**

  A dictionary management utility used to manipulate dictionary directories.

- **VMU (VDD Migration Utility)**

  A CDD-to-VDD conversion tool for migrating legacy repositories (IA-64 Only).

See *Section 6, "Differences Between VDO and CDO Utilities"* and *Section 7, "Differences Between VDMU and DMU Utilities"* for a comparison between the VDD utilities that have CDD counterparts.

6. Differences Between VDO and CDO Utilities

This section describes key differences in functionality between the VDD Operator utility (VDO) and the Common Dictionary Operator utility (CDO). These differences are important for users migrating existing scripts or operational procedures.

6.1. Supported Input Dialects

VDO supports a broader range of input formats than CDO. Specifically, VDO accepts:

- Legacy DDL (CDDL) syntax used by DMU/CDDL tools.
- CDO-style record definitions.

These dialects are supported concurrently within a single VDD repository. This capability allows consolidation of legacy and modern definitions within a single environment.

In contrast, CDD associates different syntactic dialects with distinct tools and repository formats.

6.2. Command Parsing and Keyword Handling

CDO provides flexible command-parsing capabilities. Command keywords may be abbreviated, provided that the abbreviation is unique. A spell-checking mechanism automatically corrects misspelled keywords when possible, and command processing continues without error.

In contrast, VDO currently enforces strict command syntax:

- All command keywords must be entered in full.
- Command abbreviation is not supported.
- No spelling correction is performed.

Additionally, the following command-parsing features provided by CDO are not currently implemented in VDO:

- Wildcard object-name matching.
- Comma-separated object lists.

As a result, existing command procedures that rely on keyword abbreviations, spelling correction, wildcard expansion, or object lists may require modification when migrating from CDO to VDO.

Note

Improvements to command parsing are planned for future VDD releases.

6.3. Transaction Command Semantics

In CDO, transaction control commands provide full transactional behavior:

- **START_TRANSACTION** begins a transaction scope.
- **COMMIT** completes the transaction, making all changes permanent.
- **ROLLBACK** aborts the transaction.

These commands ensure that changes are committed as a single logical unit, locks are held and released appropriately, and repository and database transaction overhead is minimized by batching multiple operations.

In VDO, the same commands are accepted for compatibility but *have no operational effect*:

- **START_TRANSACTION**, **COMMIT**, and **ROLLBACK** are *no-ops*.
- Each command is executed as an independent, auto-committed operation.

As a result, existing scripts using transaction commands will execute without error, but the commands have no effect and provide no transactional guarantees.

6.4. Error Reporting and Diagnostics

CDO returns specific, named diagnostic messages such as `CDO-E-ERRDEFINE`, `CDO-E-INVNAME`, and `CDO-E-KWSYNTAX`. These messages supply precise error classification and context.

VDO currently implements a simplified error reporting model. Errors are typically reported using generic status codes, such as:

- `%VDO-E-PERR` (parse error)

- `%VDO-E-UNRECOGNIZED` (unrecognized command)

As a result, contextual detail is limited compared to CDO.

Note

Enhanced diagnostic reporting is planned for a future VDD release.

---

7. Differences Between VDMU and DMU Utilities

This section describes the key differences between the VDD Management Utility (VDMU) and the CDD Dictionary Management Utility (DMU). These differences primarily affect command parsing and error reporting behavior.

7.1. Command Interface and Abbreviation

VDMU retains the command interface model used by DMU. Specifically, VDMU:

- Is CLD-based, consistent with DMU.
- Supports standard DCL-style command abbreviations.
- Supports invocation as a foreign command, allowing commands to be executed directly from the DCL prompt. For example:

```
$ VDMU LIST
```

VDMU accepts standard DCL-style command abbreviations such as **`CRE`**, **`LI`**, **`DEL`**, and **`SH DEF`**. The behavior is consistent with DMU.

However, the following command-parsing features provided by DMU are not currently implemented in VDMU:

- Wildcard object-name matching.
- Comma-separated object lists.

As a result, existing command procedures that rely on object lists or wildcard expansion may require modification when migrating from DMU to VDMU.

Note

Improvements to command parsing are planned for future VDD releases.

7.2. Error Reporting and Status Codes

The primary functional difference between VDMU and DMU lies in error reporting behavior.

While both utilities report errors using condition codes, the specific codes returned by VDMU differ from those produced by DMU for equivalent operations.

For example, when attempting to delete a non-empty directory:

- **DMU** returns `DMU-E-CDDERROR` accompanied by `CDD-E-NODHASCHI`.
- **VDMU** returns `%DMU-E-DELFAIL`.

Similarly, certain syntax errors may produce different `DMU-E-*` status codes in VDMU. In some cases, a corresponding DMU diagnostic may not be returned at all.

As a result, scripts that branch on specific DMU error codes may behave differently against VDMU.

8. ACMS Support

This section describes ACMS-related differences between VDD and CDD.

8.1. ACMS System Workspace Initialization

Under CDD, ACMS system workspace definitions are created as part of the ACMS installation procedure.

VDD exhibits equivalent behavior when ACMS is installed on a system where VDD is already present.

However, if VDD is installed after ACMS has already been installed against CDD, the ACMS system workspace definitions are not initially present in the VDD repository. In this case, the VDD installation procedure creates the required definitions automatically.

8.2. ACMS Menu Definitions

CDD supports ACMS menu definitions. This support is not currently implemented in VDD.

As a result, ACMS applications that contain menu definitions cannot be built using VDD.

Note

Support for ACMS menu definitions may be added in a future VDD release.
