



# Teradata® Unity™

## User Guide

---

Release 17.00




September 2020

# Copyright and Trademarks

Copyright © 2011 - 2020 by Teradata. All Rights Reserved.

All copyrights and trademarks used in Teradata documentation are the property of their respective owners. For more information, see [Trademark Information](#).

## Product Safety

Safety type	Description
 <b>NOTICE</b>	Indicates a situation which, if not avoided, could result in damage to property, such as to equipment or data, but not related to personal injury.
 <b>CAUTION</b>	Indicates a hazardous situation which, if not avoided, could result in minor or moderate personal injury.
 <b>WARNING</b>	Indicates a hazardous situation which, if not avoided, could result in death or serious personal injury.

## Third-Party Materials

Non-Teradata (i.e., third-party) sites, documents or communications ("Third-party Materials") may be accessed or accessible (e.g., linked or posted) in or in connection with a Teradata site, document or communication. Such Third-party Materials are provided for your convenience only and do not imply any endorsement of any third party by Teradata or any endorsement of Teradata by such third party. Teradata is not responsible for the accuracy of any content contained within such Third-party Materials, which are provided on an "AS IS" basis by Teradata. Such third party is solely and directly responsible for its sites, documents and communications and any harm they may cause you or others.

## Warranty Disclaimer

**Except as may be provided in a separate written agreement with Teradata or required by applicable law, the information available from the Teradata Documentation website or contained in Teradata information products is provided on an "as-is" basis, without warranty of any kind, either express or implied, including the implied warranties of merchantability, fitness for a particular purpose, or noninfringement.**

The information available from the Teradata Documentation website or contained in Teradata information products may contain references or cross-references to features, functions, products, or services that are not announced or available in your country. Such references do not imply that Teradata Corporation intends to announce such features, functions, products, or services in your country. Please consult your local Teradata Corporation representative for those features, functions, products, or services available in your country.

The information available from the Teradata Documentation website or contained in Teradata information products may be changed or updated by Teradata at any time without notice. Teradata may also make changes in the products or services described in this information at any time without notice.

## Machine-Assisted Translation

Certain materials on this website have been translated using machine-assisted translation software/tools. Machine-assisted translations of any materials into languages other than English are intended solely as a convenience to the non-English-reading users and are not legally binding. Anybody relying on such information does so at his or her own risk. No automated translation is perfect nor is it intended to replace human translators. Teradata does not make any promises, assurances, or guarantees as to the accuracy of the machine-assisted translations provided. Teradata accepts no responsibility and shall not be liable for any damage or issues that may result from using such translations. Users are reminded to use the English contents.

## Feedback

To maintain the quality of our products and services, e-mail your comments on the accuracy, clarity, organization, and value of this document to: [docs@teradata.com](mailto:docs@teradata.com).

Any comments or materials (collectively referred to as "Feedback") sent to Teradata Corporation will be deemed nonconfidential. Without any payment or other obligation of any kind and without any restriction of any kind, Teradata and its affiliates are hereby free to (1) reproduce, distribute, provide access to, publish, transmit, publicly display, publicly perform, and create derivative works of, the Feedback, (2) use any ideas, concepts, know-how, and techniques contained in such Feedback for any purpose whatsoever, including developing, manufacturing, and marketing products and services incorporating the Feedback, and (3) authorize others to do any or all of the above.

# Contents

<b>Chapter 1: Teradata Unity Overview</b>	<b>5</b>
Welcome to Teradata Vantage	5
Teradata Analytical Ecosystem Integration	5
Unity Features	5
Basic Unity Operation	5
<b>Chapter 2: Unity Configuration</b>	<b>17</b>
Unity Operational Considerations	17
Prioritizing Unitymgmt in TASM	17
Addition of a Teradata System to the Unity Environment	18
Testing Connectivity to a Managed Teradata System	20
Synchronizing Database Content to a New System	20
Unity Data Dictionary	21
<b>Chapter 3: Unity Routing Rules</b>	<b>23</b>
Session Routing	23
Managing Routing Rules in the Unity UI	27
Managing Routing Rules Using unityadmin	27
Passive Routing for Reporting and Other Read-Only Workloads	28
Managed Routing Load and Workloads Requiring Data Synchronization	29
Routing Rule for Rejecting Unexpected Users	29
Default Routing Rule	30
Routing Rules Example	30
Error Lists in Routing Rules	30
Routing Rule with User Mappings	30
User Mappings Order	31
Default Account Strings for Groups Of Users	31
Using Time Windows	31
Dynamic Routing Rule Changes Using Query Band Requests	32
Unity Session Monitoring	32
<b>Chapter 4: Unity UI</b>	<b>34</b>
Overview	34
Managing Configurations in the Unity UI	34
Monitoring Objects and Alerts in the Unity UI	35
Changing the Unity UI Settings	35
<b>Chapter 5: Table Consistency in Unity</b>	<b>36</b>
Interrupted Tables	36

Unrecoverable Tables .....	37
<b>Chapter 6: Teradata System Outages .....</b>	<b>40</b>
Planned Outages .....	40
Unplanned Outages .....	43
<b>Chapter 7: Security and Authentication .....</b>	<b>47</b>
Providing the Required Passwords to Teradata for Change Controls .....	47
Changing the Unity Management User Password .....	48
Changing the Unity Admin Password Used by Viewpoint .....	48
Updating Passwords for Unity System Accounts .....	49
Authenticating Unity Admin Users with LDAP .....	49
<b>Chapter 8: Unity Maintenance .....</b>	<b>51</b>
Changing the State of a Table for Maintenance .....	51
Changing the State of a System for Maintenance .....	51
Failing Over to the Standby Sequencer for Planned Maintenance .....	51
Blocking Incoming User Sessions for Maintenance .....	51
Shutting Down Unity .....	51
Deadlock Detection .....	52
Best Practices for Backup and Restore in a Unity Managed Environment .....	53
<b>Appendix A: Viewpoint Unity Setup Portlet .....</b>	<b>55</b>
<b>Appendix B: Viewpoint Unity Portlet .....</b>	<b>81</b>
<b>Appendix C: Log and Diagnostic Files .....</b>	<b>109</b>
<b>Appendix D: Unityadmin Command Reference .....</b>	<b>114</b>
<b>Appendix E: Additional Information .....</b>	<b>250</b>

# Teradata Unity Overview

## Welcome to Teradata Vantage

Teradata Vantage™ is our flagship analytic platform offering, which evolved from our industry-leading Teradata® Database. Until references in content are updated to reflect this change, the term Teradata Database is synonymous with Teradata Vantage.

Advanced SQL Engine (was NewSQL Engine) is a core capability of Teradata Vantage, based on our best-in-class Teradata Database. Advanced SQL refers to the ability to run advanced analytic functions beyond that of standard SQL.

## Teradata Analytical Ecosystem Integration

Teradata® Unity™ integrates with Data Mover to enable continuously available (also known as Active-Active, or dual-load) Teradata Vantage systems.

Unity can be scripted to work with Data Mover to define jobs that copy database objects, such as tables, users, views, macros, stored procedures, and statistics between Teradata Database systems.

## Unity Features

Unity supports multi-system environments with high volume, scalable data loading to multiple Teradata systems.

Managed Routing	Passive Routing
High availability	High availability
Data synchronization	Highly scalable access without data synchronization
Disaster recovery	Low latency
User and query routing	Simple reporting
Workload balancing	Session based routing
Workload shifting	Isolated sandboxes

## Basic Unity Operation

### Unity Interfaces

There are three interfaces to manage Unity.

Unity Interface	Description
Unity UI	Graphical user interface
unityadmin command line	Simple command line tool that can be installed on any suitable Linux server
Unity Viewpoint Portlet	Viewpoint plugin to view the Unity state and configuration from Viewpoint

## Unity Processes

The Unity software stack includes four processes. You can view the state of these processes on all Unity servers from the Unity UI **Components** view.

Process	Description
Endpoint	Listens on port 1025 for incoming client connections, and processes incoming requests
Sequencer	Coordinates requests across the available Teradata systems, tracking system, and table state
Dispatcher	Executes work on a specific Teradata system
Watchdog	Monitors the other Unity processes and performs file transfers between Unity servers

## Managing Unity Processes

The Unity processes can be controlled from the Unity service script on each Unity server.

1. To manage Unity processes, do one of the following from the Unity service script:

Action	Steps
To start all Unity processes	Run <code>/opt/teradata/unity/bin # unity start</code>
To check the status of the Unity processes	Run <code>/opt/teradata/unity/bin # unity status</code>
To stop all Unity processes	Run <code>/opt/teradata/unity/bin # unity stop</code>


## Alerts


Unity alerts notify a DBA of an event that requires attention. A summary of currently open alerts appears on the **Alerts** card of the Unity UI **Overview** screen. From this view you can easily and quickly sort for alert


types and severity, and for the different types of alerts that are open. Select the alert name to see the details for each alert type.

### Alerts

Critical (0)
Warning (3)
Normal (1)


**MISMATCHED ACC. STRINGS (1)**  
Mismatched account strings have been detected.


**TOO MANY LOAD CHECKPOINTS (1)**  
A load job has been aborted for having too many checkpoints.


**LOAD TOO BIG (1)**  
A load job has exceeded the allowed size limit.

[VIEW MORE](#)

You can also view alerts using the ALERT LIST command on the unityadmin command line. The following is an example of open alerts summary from the unityadmin command line.

```
unityadmin> alert list;
Count   Type   Severity   Description
-----
      1     1     Normal   Start - Unity sequencer has started.
      2    20     Critical  Mgmt Login Failed - The Unity Management User was
unable to log in to a Teradata System
      1   175     Warning   Mismatched Acc. Strings - Mismatched account
strings have been detected.
      3   225     Warning   Server Space - A process has passed a warning
threshold for available disk space.
      6   226     Critical  Server Space - A process has passed a critical
threshold for available disk space.
```

The following is an example list of all occurrences of a particular type of alert.

```
unityadmin> alert list 225;
Alert Type   : 225
Description  : A process has passed a warning threshold for available disk space.
Severity     : Warning
Recommended Action: Recover any tables or systems that are out of service or
interrupted. Then check the /data directory and remove any old files. Consider
adding disk space or expansion servers to accommodate higher load volumes.
This alert will be automatically cleared when the disk space has reached a
safe threshold.
```

ID	Initial Raised Time	Rep.	Details
-----			
9	2019-10-23, 11:33:33	340	The process identified as 'region2_dsp_db1', on Unity server 'unity2', has 4.2 GB of available disk space remaining, below the warning threshold of 10.0 GB (34%).
11	2019-10-23, 11:33:33	340	The process identified as 'region2_dsp_db2', on Unity server 'unity2', has 4.2 GB of available disk space remaining, below the warning threshold of 10.0 GB (34%).
13	2019-10-23, 11:33:33	340	The process identified as 'region2_ept', on Unity server 'unity2', has 4.2 GB of available disk space remaining, below the warning threshold of 10.0 GB (34%).

## Forwarding Alerts to Email

It is important to be aware of Unity alerts. For best practice, have all Unity alerts forwarded to a list of email addresses of database administrators or operations support personnel.

Configure the Notification Service bundled with the Unity UI to forward email messages to an appropriate corporate SMTP server (under the 'Settings > SMTP Config' screen).

You can also configure a set of email addresses to forward specific types of alerts.

## Notification Service Configuration for Alert Emails

The Notification Service is the SMTP email service provided in the Unity UI docker container stack along with UI and REST. Once configured, you receive emails when an alert is raised.

### Adding the SMTP Server Details in the Notification Service

Add the SMTP Server details in the notification service to specify which server to send the emails. SMTP server details must be provided by your company. You can set up the notification service in the Unity UI.

1. Log on to the Unity UI and select **Settings**, then **SMTP Config**.
2. Select **Edit Config** to enter the details.

### Adding the Notification Service Details on the Unity Server

Provide the notification service details and default recipient email address for alerts.

1. From the Unity UI, select **Configuration**, then **Notification**
2. Click **Add Config**.

The Notification Server and Authentication Server are hosted on the same server as the UI and REST docker containers. If you have installed the Unity UI on the unity server, you can use the



address 127.0.0.1 for both the notification service and the authentication server. To get the token, provide the username and password. You can also provide the LDAP credentials or the root user credentials for the user service. The admin user setup was provided during the Unity UI installation and configuration.

Edit Notification Config
X

Notification Server \*

127.0.0.1

Port \*

9390

Authentication Server \*

127.0.0.1

Port \*

5000

Username \*

admin

Password

Email Alerts To \*

jsmith@company1.com X

Reason \*


Initial Configuration

21/1000

EDIT

## Adding an Email Address to an Alert

All emails for alerts are sent to the default email addresses provided in the config section of the Unity UI. If needed, you can configure each alert type to send to a different set of email addresses.

1. From the Unity UI, select **Alert Types**.
2. Select  to edit an alert type.

## Enabling and Disabling Alerts Using unityadmin

You can enable and disable alerts directly using unityadmin.

1. From unityadmin, enable or disable alerts using the following example commands:

Alert Action	Example
Enable single alert	ALERT ENABLE 40034;

Alert Action	Example
Enable multiple alerts	ALERT ENABLE 40034, (40043, 40023, 40012);
Disable single alert	ALERT DISABLE 40034;
Disable multiple alerts	ALERT DISABLE 40034, (40043, 40023, 40012);

## Change Data Multicast (CDM)

Change Data Multicast (CDM) is a method to replicate changes between multiple systems. When enabled, CDM replicates SQL changes that are non-deterministic and can produce different results if run in parallel on different systems. CDM sends a non-deterministic SQL to one of the managed Teradata systems and then captures all the resulting data changes. It then multicasts those data changes to the rest of the participating Teradata systems. This allows Unity to replicate changes that involve non-deterministic functions like RANDOM and SAMPLE and writes to tables that include Identity columns.

### Note:

The replication of IDENTITY columns adversely affects performance. CDM requires that you redefine IDENTITY columns so they have different ranges on different systems.

Unity uses CDM across managed systems to keep non-deterministic objects and triggers synchronized. Non-deterministic objects and triggers include:

- Identity Columns
- Dynamic SQL in Stored Procedures
- Non-deterministic ordering or sampling (i.e. Sample, TOP, ROWNUMBER)
- Analytic functions

When non-deterministic objects or triggers are identified, Unity processes the action on the first Teradata system, then uses CDM to send and override the values on the other systems.

## Handling of Non-Deterministic Elements

Unity overrides certain non-deterministic functions, such as current time and random. The following table shows which SQL functions are automatically rewritten if used in a SQL request which performs a write.

SQL Function	Replacement
RANDOM	If CDM is disabled, active sequencer replaces this function with a single randomly generated value. A single write request involving multiple rows results in the same random number. If CDM is enabled, Unity does not replace the RANDOM clause. The write requests are replicated using CDM.
CURRENT_DATE	Active sequencer generates a date to replace this function.
CURRENT_TIME	Active sequencer generates a time to replace this function.

SQL Function	Replacement
CURRENT_TIMESTAMP	Active sequencer generates a timestamp to replace this function.
DATE	Active sequencer generates a date to replace this function.
SESSION	Not supported.
TIME	Active sequencer generates a time to replace this function.
HASHAMP	Not supported.
HASHBAKAMP	Not supported.
HASHROW	Not supported.
HASHBUCKET	Not supported.
ROW NUMBER	If CDM is disabled, not supported. If CDM is enabled, write replicated using CDM.
Sample Clause	If CDM is disabled, not supported. If CDM is enabled, write requests using SAMPLE clause are supported and replicated using CDM.
INSERT into table with identity column	If CDM is disabled, not supported. If CDM is enabled, write replicated using CDM.
Non-deterministic User-Defined Function	If CDM is disabled, write replicated using SQL Multicast. If CDM is enabled, write replicated using CDM.
Stored procedure call	If CDM is disabled, writes are replicated using SQL Multicast. Traditional Unity SP limitations apply. If CDM is enabled, writes are replicated using the enhanced stored procedure feature. Traditional limitations do not apply.
Write using READ_NOS table operator	If CDM is disabled, not supported. If CDM is enabled, write replicated using CDM.
Write using foreign tables	If CDM is disabled, not supported. If CDM is enabled, write replicated using CDM.
Updatable Cursors	If CDM is disabled, write replication is not supported. If CDM is enabled, writes are replicated using CDM.

Unity must be part of the secure database cluster to verify override. The secure database cluster includes Unity and all Teradata systems it manages. Systems in the secure database cluster are identified through shared proxy certificates that must exist on all database nodes and on all Unity server nodes.

For more information on Teradata Proxy certificates, see *Teradata Vantage™ - Advanced SQL Engine Security Administration*, B035-1100.

## Enhanced Stored Procedure

When a stored procedure (SP) call is made through Unity with CDM enabled, Unity uses enhanced SP handling for that SP call. With enhanced SP handling, the database engine forwards each statement from the SP body to Unity, and Unity SQL replicates or CDM replicates each statement.

Each individual statement from the SP body is sent to Unity for replication purpose, lifting all traditional Unity SP limitations. This allows Unity to support SP containing dynamic SQL, conditional execution patch, multiple DDL, multiple transaction, and so on.

## CDM and Structured User-Defined Types

To replicate tables with structured User-Defined Types (UDT) columns successfully, the structured UDT must satisfy the following requirements:

- The CAST function must be defined for the default TO-SQL transform.
- The default transform must be symmetrical. In other words, casting UDT > base type > UDT must yield the exact same UDT.

The above requirements should be best practice when using structured UDT in general. They are required for proper CDM replication of structured UDT.

## CDM Streaming

When running Unity 17.00 with Teradata 17.00 or later, the CDM performance is improved with the CDM Streaming and CDM Multi-Stream functionalities.

With CDM Streaming, the source system sends the change data information to Unity as the AMP executes the query. Unity forwards that change data information to the target systems, which stores the data in a staging table. Once all the data is received on the target system, the change data is applied as a single bulk operation.

With CDM Multi-Stream, multiple connections are used to retrieve the change data from the source system and to apply the data to the target systems.

## CDM Staging Tables

The staging tables used by CDM Streaming, CDM Multi-Stream, and CDM for CREATE TABLE AS (CTAS) are managed by Unity. They are automatically created in the current user default database, and are automatically dropped when Unity has completed the transaction.

In order to avoid any name collision with user tables, the CDM staging tables use a unique naming convention:

```
_U<sid>_<tid>_<dbqlid>_<stmtno>_<orderno>_<oid>_ISTG_<tablename>
```

Where:

- *sid* is the Unity primary session ID for the original request.

- *tid* is the Unity primary transaction ID for the original request.
- *dbqlid* is the DBQL Query ID from the source system for the original request.
- *stmtno* is the statement number from the source system for the original request.
- *orderno* is the step order number within each statement from the source system for the original request.
- *oid* is the Unity dictionary ID of the base table for which this staging table is used.
- *ISTG*, *DSTG*, or *TMP* are staging table types.
- *tablename* is the name of the base table for which this staging table is used.

If Unity is shut down improperly while CDM staging tables are in use, it is possible for some staging tables to remain after the Unity restart. These staging tables can be identified by the above naming convention, and the fact that the associated session has moved passed the associated transaction. If that is the case, the orphaned staging table can be dropped through Unity. This should only be done if you are certain that the staging table is orphaned. Dropping a staging table that is still in use will cause the change data replication to fail on the target system.

## Secondary Sessions

CDM Streaming and CDM for a non-deterministic CREATE TABLE AS query uses a secondary session to create and populate the staging table. CDM Multi-Stream uses multiple secondary sessions to load the staging table in parallel.

The use of secondary sessions increases the overall session count on the target Teradata system. You can limit the number of secondary sessions using the **CDMMaxDataConnectionsPerGroup** configuration parameter.

## Unity @xplain

The Unity @xplain feature provides information about an executed query. The output of a SQL statement prefixed with @xplain returns the following result sets.

---

### Note:

The output of a query prefixed with @xplain(#) returns only the specified result set.

---

## Resultant Query / Rewrite Conditions

The resultant query result set provides the query after all rewrite rules have been applied.

```
*** Query completed. One row found. One column returned.
*** Total elapsed time was 1 second.
```

Resultant Query

```
-----
INS t0 SEL TOP 1 * FROM v0;
```

Column	Details	Possible Values
Resultant Query	The query processed by Unity once all rewrite rules have been applied.	The query as processed by Unity

## Parsing Details

The parsing details result set provides the overall query details from the Unity parser.

```
*** Query completed. One row found. 4 columns returned.
```

```
Parse Status  Query Class  Primary Table  Query Type
-----
successful    Insert        -              Write          -
```

Column	Details	Possible Values
Parse Status	Indicates whether the parsing was successful	Successful
Query Class	Displays the statement type of the query	Any valid statement type
Primary Table	Name of the target table for single-table queries	Name of the primary table, or "-" if unknown
Query Type	Identify whether the query is a read or write operation	read, write

## Unflattened Object Details

The unflattened object details result set contains the list of objects being directly referenced by the query.

```
*** Query completed. 2 rows found. 5 columns returned.
```

```
Object Type Access Dictionary ID Name Full Name
-----
view        read          1003 v0    db0.v0
table       write         1001 t0    db0.t0
```

Column	Details	Possible Values
Object Type	The type of object	table, view, macro, function, procedure, trigger, reference, UDT, other
Access	The type of access	read, write, call
Dictionary ID	The Unity dictionary ID for the object	Valid dictionary ID
Name	Unqualified object name	Name of object
Full Name	Fully qualified object name	Name of object, including database name

## Flattened Object Details

The flattened object details result set contains the list of object and tables being accessed by the query once all views/macros/procedures are expanded.

```
*** Query completed. 3 rows found. 4 columns returned.
```

```
Table ID Name Full Name Access
```

```
-----
1002 t1    db0.t1    read
1003 v0    db0.v0    read
1001 t0    db0.t0    write
```

Column	Details	Possible Values
Table ID	The Unity dictionary ID for the object	Valid dictionary ID
Name	Unqualified object name	Name of object
Full Name	Fully qualified object name	Name of object, including database name
Access	The type of access	read, write

## Lock List

The lock list result set contains the list and type of locks obtained by Unity for the query.

```
*** Query completed. 3 rows found. 6 columns returned.
```

```
Lock ID Full Name Type Scope Level Row Hash
```

```
-----
1001 db0.t0    write transaction table -
1002 db0.t1    read  transaction table -
1003 db0.v0    read  transaction table -
```

Column	Details	Possible Values
Lock ID	The Unity dictionary ID for the object	Valid dictionary ID
Full Name	Fully qualified object name	Name of object, including database name
Type	The lock type	access, read, write, write escalate, exclusive, exclusive escalate
Scope	The lock scope	transaction
Level	The lock level	table, row
Row Hash	For row lock, the corresponding row hash for the lock.	Row hash value, or "-" for table lock

Replication Type

The replication type result set provides details on whether a query will be SQL replicated or CDM replicated.

```
*** Query completed. One row found. One column returned.
Replication Type
-----
-----
Write request CDM replicated because query uses a TOP clause in a write query
```

Column	Details	Possible Values
Replication Type	Provides details on whether the query is a read/write/DDL query, whether it will be SQL or CDM replicated, and details for the replication choice	Text detailing the replication type



# Unity Configuration

## Unity Operational Considerations



To avoid any operational issues, a DBA must be aware of the following aspects of the unitymgmt user. This user helps Unity monitor connectivity to the managed Teradata systems.


- This unitymgmt user must be excluded from all backup jobs running on the Teradata system. It has no data to store in backups. Executing a backup on this database can result in locks that block incoming requests and result in an unintended outage.
- The unitymgmt user must be prioritized in the TASM rule set as a high priority tactical user (never queued). Failing to do so could result in database locking issues that lead to unintended outages.
- The dictionary scanner requires the unitymgmt user to have SHOW permission on scanned databases to retrieve the object text and default database for stored procedures.

## Prioritizing Unitymgmt in TASM

Connections from the sequencer and dispatcher query the unitymgmt.processHeartbeat table and need to take priority. Set a Teradata Active System Management (TASM) rule to prioritize logins and queries from unitymgmt.

1. In the **Workload Designer** portlet, select a system, then do one of the following:

Option	Description
Create a new ruleset for the prioritization rule	<ol style="list-style-type: none"> <li>a. Select .</li> <li>b. Enter a ruleset name and <b>Save</b>.</li> </ol>
Use an existing ruleset for the prioritization rule	<ol style="list-style-type: none"> <li>a. In the local Working section, select the ruleset , then select <b>View/Edit</b>.</li> </ol>

2. Select **Workloads**, then .
3. Name the workload unitymgmt\_tactical, select **Tactical**, then select **Save**.
4. Select **Classification**.
5. From **Add Classification Criteria**, select **Request Source**, then select **Add**.
6. In **Edit Request Source Criteria**, from **Source Type**, select **Username**.
7. In **Username** type unitymgmt, then select **Include**.

### Note:

Though unitymgmt is the default username suggested by the setup script, customers can specify their own username.

8. Select **OK**, then **Save**.
9. To the left of **Workload Designer**, select the return arrow, then select **unitymgmt\_tactical**.

## Addition of a Teradata System to the Unity Environment

A new Teradata system must be added to the Unity environment before you can use it. This can be done using the SYSTEM ADD command at the unityadmin command prompt or from Unity UI.

When the system is added to the Unity environment, the following happens:

- The required processes and config settings are added to connect to the Teradata system.
- If a DBA provides dbc credentials for the Teradata system being added, the unitymgmt user is added to the new Teradata system.

---

### Note:

This action can be deferred or completed manually. If the required dbc credentials are not provided, it is skipped. See [Manually Adding the Unitymgmt User to a Teradata System](#).

---

- If a DBA includes the required root name and password, proxy authentication is configured on the Teradata system. This action triggers a reset of the Teradata system, in order to complete the proxy configuration.
- 

### Note:

This action can be deferred or completed manually. If the required root credentials are not provided, it is skipped. See [Manually Configuring Proxy Authentication on a Teradata System or Single Node](#).

---

- A full dictionary scan of the newly added system is executed so its objects are available in the Unity data dictionary. Any newly added objects are unmanaged by default.
- The newly added processes are started.

## Adding a Teradata System Using Unityadmin Command

---

### Prerequisite:

- The system must be accessible through the network from all Unity servers.
  - To create the unitymgmt user, you must have DBC or equivalent access to create a new user and to grant the required authorization to the user.
  - To configure proxy authentication, you must have root access to the Teradata system nodes using SSH.
-

1. [Optional] Retrieve the name of the existing region to which you want to add the new system by using the REGION LIST command.
2. Log in as root to the unity server with the active sequencer.
3. Run the SYSTEM ADD command using one of the options below.
  - If the username and password was supplied using the unityPasswd tool: `unityadmin> system add <db1> <region1>;`
  - If the username and password were not supplied using the unityPasswd tool: `unityadmin> system add <db1> <region1> user root password <PASS123> dbcuser <dbc> dbcPassword <dbc>;`

## Manually Adding the Unitymgmt User to a Teradata System

The `addUnityMgmtUser.sh` script creates the user and updates the password to the current value for the `unitymgmt` user in the `tdwallet`. If the user already exists, the password is updated to the current value in `tdwallet`.

1. Make sure you have DBC or equivalent access to create a new user and then run the required grants.
2. Add the `unitymgmt` user.
  - a. Enter the following if the customer used the `unityPasswd` tool:
 

```
# /opt/teradata/unity/scripts/addUnityMgmtUser.sh -s [system];
```
  - b. If the customer did not use the `unityPasswd` tool, specify the user and password to use to create the `unitymgmt` user (typically DBC).
 

```
# /opt/teradata/unity/scripts/addUnityMgmtUser.sh -s [system] -du dbc -dP dbc;
```

## Manually Configuring Proxy Authentication on a Teradata System or Single Node

1. Do one of the following:
  - Run the `systemProxyConfig.sh` script using the `unityPasswd` tool with required credentials. With this option, you do not have to specify required credentials as arguments.
 

```
Host:/opt/teradata/unity/scripts #./systemProxyConfig.sh -s tdpid
```
  - If the username and password were not supplied using the `unityPasswd` tool, you must specify the user (typically root) and user password used to configure proxy on the Teradata system:
 

```
Host:/opt/teradata/unity/scripts #./systemProxyConfig.sh -s tdpid -u root -P [password]
```
  - To repair the proxy on all Teradata system nodes on all Unity servers in parallel, use the PROXY PAIR command at the `unityadmin` prompt:
 

```
unityadmin> proxy pair [tdpid] password <system password>;
```

---

### Note:

The PROXY PAIR command does not support the `unityPasswd` tool for supplying a password.

---

**Note:**

To use this command, the root password must be synchronized on all system nodes.

## Adding a Teradata System Using the Unity UI

1. In the Unity UI, select **Systems**.
2. Select **Add System**.
3. In the window that appears, complete the required information.

**Note:**

It may take several minutes for a system with a large number of objects because adding a system triggers a data dictionary scan of that system. For example, for a system with 1 million objects, it may take up to 1 hour. You can skip this dictionary scan using the NO SCAN option in the UI or command line.

## Testing Connectivity to a Managed Teradata System

Unity tool conntest.sh helps comprehensively test the connectivity to a managed Teradata system. This tool includes the proxycheck tool that was previously used.

1. Run the following command to test connectivity.  
`<unity1>:/opt/teradata/unity/scripts # ./connTest.sh -s <db1>`

The following is checked after the test is run.

Basic connectivity	<ul style="list-style-type: none"> <li>• Teradata system is reachable (on all nodes).</li> <li>• tdwallet is ready to use.</li> <li>• tdwallet has the right keys.</li> <li>• unitymgmt user can connect with TD2 authentication using the password in tdwallet.</li> <li>• Correct tdgss version is installed.</li> </ul>
Local proxy configuration	<ul style="list-style-type: none"> <li>• disableProxyConnection setting is set.</li> <li>• unitymgmt user can connect using proxy authentication.</li> <li>• tdgss xml configuration is correct.</li> <li>• Local tdgss certs exist and are in the right place.</li> <li>• tdgss binary matches the settings in user xml file.</li> </ul>
Remote proxy configuration	<ul style="list-style-type: none"> <li>• System timezone matches the unity timezone.</li> <li>• tdgss xml configuration is correct.</li> <li>• Gateway password digest is correct.</li> <li>• tdgss version is installed on the remote node.</li> <li>• Unity server and remote node have copies of each other's public certs.</li> </ul>

## Synchronizing Database Content to a New System

These commands may take a long time to complete depending on the number and size of the tables within the database. While an individual system is being synchronized, a table-level read lock is placed on the table of the source system. This prevents any writes to the table while the table is being copied.

1. From unityadmin, run the following command to activate a new system without activating its tables:  
unityadmin> system activate system2 NO TABLES;
2. Run the following command to mark all the tables in the specified database as being available on the second system:  
unityadmin> database manage dbtest on system1, system2;

---

**Note:**

Those tables start in the unrecoverable state on the new system.

---

3. Run the following command to synchronize and activate all the tables in the specified database:  
unityadmin> database resync dbtest from system1 to system2;

## Unity Data Dictionary

Passive routing sessions can access objects on the Teradata systems to which they are routed. It is not necessary to add or manage objects for passive routing sessions.

Managed routing sessions can only access objects that are managed on the systems to which they are routed. This means the objects must exist in the Unity data dictionary, which monitors the object definitions and placement.

Unity data dictionary is maintained automatically. As clients create new objects, the objects are added to the Unity data dictionary. As objects are dropped, they are removed from the data dictionary.

When working with a new object or database created directly on one or more Teradata systems, you can manually run a data dictionary scan. When a new system is added to the Unity environment, a complete scan of the system and its objects is run automatically, to inventory all the objects that exist on the system. When you add a system, you can skip the full system scan using the NOSCAN option with the SYSTEM ADD command from unityadmin.

---

**Note:**

If you do not see the database you want to scan on the **Databases** screen, perform a full system scan from the **Systems** screen.

---

After executing a data dictionary scan to find where an object exists, you can manage the object on all the systems on which the object exists or on a single system of the set, if the objects are not yet synchronized. Controlling where an object or database is managed can be done using Unity UI or the unityadmin command line. When deploying applications to use managed routing, you can change where the application database is managed.

For best practice, start with managing the lowest level of a database hierarchy by selecting where to manage the databases containing staging and base tables. Then progress upwards to any dependent databases containing application views.

You can choose where to manage objects individually or at the database level. When deploying a new application to Unity, work at the database level.

You can also change where a database is managed at the unityadmin prompt:

```
unityadmin> help database manage;
Command: DATABASE MANAGE {<database> [ON <tdpid> [, <tdpid>, ...]]}
[REASON '<reason>']
Description: Changes the managed location of all objects in the specified
database to the specified systems.

unityadmin> DATABASE MANAGE dbtest on db1;
```

## Managing Unity Data Dictionary Objects

1. Do one of the following to manage Unity data dictionary objects.

Action	Steps
Unity UI	<ol style="list-style-type: none"> <li>a. Select <b>Databases</b>.</li> <li>b. Select the appropriate database, then select <b>Manage Databases</b>.</li> <li>c. In the <b>Manage All Objects in Database</b> window, select the systems on which you want to manage this database.</li> </ol>
unityadmin	<pre>unityadmin&gt; DATABASE MANAGE dbtest on db1;</pre>

# Unity Routing Rules

Unity routing provides access and synchronization options for a set of Teradata systems using different routing rules. A routing rule defines how a session is connected to a set of Teradata systems. Choosing appropriate routing rules for different workloads is important for achieving business goals with a Unity environment. When chosen correctly, routing rules can provide such benefits as minimizing network latency, isolating workloads to particular systems, or providing additional scale across multiple systems.

Routing rules are selected using user mappings that match incoming connections based on defined criteria. Avoid updating routing rules once they are first designed. This best practice leads to a consistent behavior and understanding of what the routing rule is set to accomplish.

## Session Routing

Session routing determines which system to send a request. Unity applies session routing rules to each session that successfully connects to the Unity server.

Unity uses the following routing characteristics for each session:

### User Mappings

User mappings are sets of rules that define the logon requirements of a user or client.

### Routing Rules

Routing rules specifies the Teradata system and how read and write requests are routed.

### Error Profiles

Error profiles are lists of error codes in a single profile that specifies the actions if an error occurs.

### Time Windows

Time Windows let you set automated start and stop times for routine updates, planned outages, or a User Mapping. Time Windows can run a script when opening or closing, function as an on off switch, or block sets of users from production for specified periods.

Unity includes a set of default routing rules during installation. You can add and edit routing rules to meet specific business and ecosystem requirements. You can add or modify user mappings as needs change.

## Managed and Passive Routing Overview

Unity supports two types of routing rules, Managed Routing and Passive Routing. The standard routing rule from versions prior to 15.00 is managed routing.

Use managed routing when you need data synchronization. Use passive routing when running reports or when you do not need data synchronized.

The following table highlights the key differences between managed routing and passive routing.

Managed Routing	Passive Routing
Sessions are opened on all available systems, per routing rule definition. The session open command is logged in the Recovery Log.	A session is opened on exactly one system, per passive routing rule definition. The session open command is not logged in the Recovery Log.
Sessions can only read and write to those objects that are managed and deployed in data dictionaries.	<ul style="list-style-type: none"> <li>• A session can read all objects, whether or not they are deployed in the data dictionary.</li> <li>• A session can write to all objects that are not in data dictionaries.</li> <li>• [Optional] A session can write to objects in the Deployed Data Dictionary if the <b>Allow Managed Writes</b> check box or the keyword is specified.</li> </ul>
Requests are parsed by the Endpoint.	Requests are parsed by the Endpoint (so Unity can control writes to managed objects).
Requests go through Unity internal locking mechanism, and are run by the Dispatcher after all locks have been granted. Requests are logged in the Recovery Log.	Requests skip Unity internal locking and are sent to the Dispatcher, which then runs them against the database. Requests are not logged in the Recovery Log.
Error code-based retry/resubmit logic only works for read queries (with the limitation that query be the first query in an implicit transaction). Resubmit tries the query on another system where the connection is already open.	Error code-based retry/resubmit logic works for both read and write requests, regardless of transaction status and query count within transaction. Resubmit closes the session on current system, opens it on another and tries the query on that system.
A session is no longer available on a system if the system is no longer in the Active state in Unity. The session is recovered at a later time when the system is Active.	Once a Passive session is opened on a system, it remains Active on the system regardless of the system's state change within Unity. If a request's return code indicates that the session is invalid (for example, due to a system down), the connection is moved to the next system per the routing rule/error code definitions.
Unity performs deadlock detection on managed sessions before the query is sent to the system.	Unity performs deadlock detection for passive sessions after the query is sent to the associated system.
Requests update the Transaction Table on the managed systems where the connection is open.	Requests do not update the Transaction Table on the managed system where the connection is open.
Loads and exports are allowed.	Loads and exports are allowed.
A session close request is logged in the Recovery Log.	As with all passive requests, session close requests are not logged in the Recovery Log.

## Managed Session Routing Requirements

Unity authenticates the user on all systems listed in the session routing rule.



If any of the Teradata systems in the session routing list return a user authentication error, such as a wrong password, Unity rejects that client session on all Teradata systems.

The default Teradata database for the user must be the same on all Teradata systems. If the default user is not the same, Unity rejects the session.

If startup text is defined, it is chosen randomly from any one of the systems where the logon succeeded. Make sure that if a startup text is defined, it is the same across all systems where the user logs in.

---

**Note:**

Startup queries only run during normal logons and not when connections are opened during recovery processes.

---

## Managed Routing Session Considerations

For managed routing sessions, do the following:

- Enable Proxy authentication with Teradata Database 14.10 or later. This is not required for passive routing if using TD2 authentication.
- Use NTP to synchronize all time zones. All systems act as one logical system.
- Do not change DBQL and related configuration settings with Unity in managed sessions. Run those changes directly on each managed Teradata system.

For managed routing sessions, the following options are not supported:

- XQuery.
- Teradata (QueryGrid) Foreign Server, including Foreign Server definitions and Foreign Server access within SQL code.
- SQL-H.
- Writes that read from system-specific information.

---

**Note:**

Writes are supported in passive routing, but are not synchronized.

---

- Usernames and database objects containing periods, single quotation marks, spaces, or special characters as managed objects.
- 

**Note:**

In managed sessions, DDL requests using ETL routing may not achieve expected performance. When processing multiple, simultaneous requests, use Passive routing.

---

## Read Requests for Managed Routing

Unity routes session read requests to a single database system. If no routing rules are defined for a client logon, Unity provides a default routing rule (DefaultRouting).

When the requested data resides in tables on a subset of database systems, the Sequencer determines which database systems have all of the tables requested in the read operation.

The algorithm selected for routing depends on both the routing rule specific to the session client logon and current system and table states. Only tables in active or read-only state are available for reading. If there are no tables in active or read-only state, but one or more tables are in standby state, then Unity uses one of the tables in standby state for the read operation.

In a managed routing session, Unity load balances incoming read requests across all systems that have the needed tables. Unity keeps track of the number of outstanding requests on each system and sends the next read to the system with the least number of outstanding requests. Unity includes the **EnableIncludeReadsInLBQ** configuration setting. If set to true, the count of outstanding reads is included in the count of outstanding requests for load balancing in managed sessions. If set to false, only the outstanding writes are counted.

## Write Requests for Managed Routing

For write requests, Unity checks the first response for a successful operation or error code, such as communication failures. If the first response is successful, Unity notifies the client application. If the first response is an error, all active Teradata systems must return a response before the client application is notified.

The active sequencer determines which response is the most common successful response and sends that response back to the client application. If no successful response is available, the most common error response is sent to Unity. If responses come at the same time, the first response that Unity receives returns to the client application.

If the error code specifies the action Exit, Unity returns the error and exits the session on all Teradata systems. If the response indicates a successful operation, Unity compares the results returned from all Teradata systems to determine consistency across the systems. All responses returned by the Teradata systems must match to verify data consistency.

Unity uses the activity count (number of rows) to indicate data consistency for all INSERT, DELETE, UPDATE, and MERGE statements. The activity count from all Teradata systems must match the activity count of the response sent back to the client. The activity count is not checked for any other types of writes. For these type of statements, Unity only checks for statement success or failure. If the activity count or status of the write operation does not match, Unity changes the table state to Interrupted or Unrecoverable depending on the type of statement.

## Passive Routing Session Requirements

A passive routing session is opened on one system only.

If more than one system is specified in the passive routing rule, a system is chosen based on least used (LU), round robin (RR), weighted round robin (WRR), or first available. If RR is specified, the session open is attempted on the first session in the list of systems associated with the rule.

**Note:**

When running a load utility that opens multiple data sessions, only a routing rule that directs those sessions to a single system is allowed. Routing rules that load balance sessions across multiple systems are not allowed, because they would attempt to connect and load data across those specified systems. If a user attempts to run a load job using a LU, RR, or WRR PASSIVE routing rule with multiple systems, it is rejected with an error.

## Read and Write Requests for Passive Routing

According to the routing rule definition, passive sessions are opened on only one system. All read and write requests are directed to that system. If a request contains write requests to one or more objects, the objects are compared with the objects in the data dictionary. If a write object exist in the data dictionary and the session associated routing rule does not allow write requests to managed objects, the query is rejected with error code 4530, "Query rejected because associated routing rule prohibits writing to managed objects."

For write requests, because there is only one system, Unity does not perform data consistency checks or keep write requests to managed objects in sync. When a read or allowed write request is done against a Unity-managed object (object exists in data dictionary), access to the object is allowed regardless of the state of the objects. This allows passive sessions (sessions associated with a passive routing rule) to access objects that are unrecoverable or in an otherwise inaccessible state in the data dictionary.

## Managing Routing Rules in the Unity UI

1. Do one of the following to manage routing rules in the Unity UI.

Action	Steps
View a routing rule	a. Select <b>Routing Rules</b> .
Edit a routing rule	a. On the <b>Routing Rules</b> page, double click an existing routing rule. b. In the window that appears, update the settings.
Add a routing rule	a. On the <b>Routing Rules</b> page, select <b>Add Routing Rule</b> . b. In the window that appears, complete the settings.

## Managing Routing Rules Using unityadmin

1. Do one of the following to manage routing rules from unityadmin.

Action	Unityadmin Command
View a routing rule	unityadmin> routing list;
Edit or add a routing rule	<pre>unityadmin&gt; help routing update; Command: ROUTING UPDATE &lt;RoutingName&gt; (REJECT LOGON)   (READ (ALL   &lt;tdpid&gt;[, &lt;tdpid&gt;, ...]) [PREFERRED] [WRITE [ALL   &lt;tdpid&gt;[, &lt;tdpid&gt;, ...] [CREATE {PREFERRED   BALANCED}]] [OPTIONS &lt;OptionName&gt;[, &lt;OptionName&gt;, ...]] [ERRORLIST &lt;ErrorListName&gt;]]  (PASSIVE [ALLOW MANAGED WRITES] ([LU] &lt;tdpid&gt;[, &lt;tdpid&gt;, ...]   RR &lt;tdpid&gt;[, &lt;tdpid&gt;, ...] [WEIGHTED &lt;n1&gt;[, &lt;n2&gt;, ...]]) [ERRORLIST &lt;ErrorListName&gt;]) Description: Updates the system lists and routing modes for the given routing name.Acceptable &lt;OptionName&gt;: 'ACCESS UPGRADE' 'NO CDM'</pre>

## Passive Routing for Reporting and Other Read-Only Workloads

Passive routing is the most basic form of connectivity provided with Unity. It allows a client session to connect to a single specified Teradata system. Requests are then sent to that system. Optionally, the routing rule can also define a set of failover servers, so that if the first system is not available, the session reconnects to the next available system in the set.

Reporting applications and other workloads often use volatile tables for temporary storage of selected data. This data is used only for the lifetime of the workload and is discarded when the workload completes. These types of workloads generally use a passive routing rule. The passive routing option provides the following benefits.

- Volatile tables are only created on one system.
- Queries that access the tables are sent only to the system.
- With only one set of data generated (on a single system), it does not matter if the data generated is deterministic across multiple systems.
- Requests from passive routing rules are never recorded in the recovery log, never affect the Unity data dictionary, or update the unitymgmt transaction table, processing time.

There are five types of passive routing rules.

Routing Rule	Description
Single-System	<p>Specifies only a single system</p> <p>If that system is not available, the clients using this routing rule cannot connect. Use this option for business users that do not require any failover capability or for Database Administrators that need to connect to a specific system to perform maintenance.</p>

Routing Rule	Description
Preferred	Specifies that the first system in the routing list is the preferred system It is used if available. If it is not available, the session connects to the next available system in the routing rule system list.
Least-Used	Allows the client to connect to a set of two or more Teradata systems When the client connects, it is sent to the system that currently has the least number of passive connections. Use this option to makes sure the systems are evenly balanced when the sessions are running different loads.
Round Robin	Allows the client to connect to a set of two or more Teradata systems When the client connects, it is sent to the first system. The next session is sent to the second system, and so on. Use this option when each session is expected to execute the same amount of work.
Weighted Round Robin	Allows the client to connect to a set of two or more Teradata systems, but in a weighted ratio specified by the routing rule Use this option when working with systems that are sized differently and can support a different number of connected sessions.

## Managed Routing Load and Workloads Requiring Data Synchronization

Managed routing is a form of Unity connectivity recommended for data synchronization.

Use a managed routing rule for load, ETL, or ELT processes that write permanent data to the Teradata systems to synchronize it across a set of Teradata systems.

Managed routing sessions are logged to the Unity recovery log, so if any system is unavailable or encounters an unexpected error Unity can automatically replay the request when the system becomes available.

Managed routing also performs continuous consistency checking on the responses to any write performed to the database systems, to make sure all systems produce the same results.

Changes made in managed routing sessions can be synchronized in two ways. They can run using SQL multicast or Change-data-multicast (CDM), if it is enabled on the Teradata systems. Managed routing parses the incoming request to identify what tables and rows are affected by the request, and whether the query is deterministic. If the query is non-deterministic, it can run the request on one system first, and then replicate the change using CDM.

---

### Note:

Changes performed on DBC user are never replicated using CDM.

---

## Routing Rule for Rejecting Unexpected Users

It is a best practice to match an application or user to an appropriate use case and to determine who is allowed to connect to a production Unity environment. Use Unity reject routing rule assigned as the default rule in the user mappings when a user does not match any other user mapping.

```
unityadmin> routing update reject REJECT LOGON;
Successfully updated routing 'reject'
```

## Default Routing Rule

Teradata recommends that all users are mapped to a defined routing rule using a user mapping. Unity includes a default routing rule that is used when a user is not mapped to a defined routing rule. After Unity is installed, the 'DefaultRouting' rule is set to Reject. This is a best practice to make sure that any user not mapped to a user mapping is blocked.

The Default Routing rule can be updated to choose a different default behavior. For example, you can have users default to a passive, single-system routing rule.

When a user is mapped to the default routing rule, an alert is triggered to inform the DBA. This alert can be disabled, if this is intended behavior. However, this is not a recommended best practice.

## Routing Rules Example

Here is an example of a set of routing rules for most environments. These base rules are configured by default in all new environments. They can be customized.

```
unityadmin> routing list;
DefaultRouting  READ WRITE
Reporting      PASSIVE db1, db2
ETL            READ db1, db2 WRITE db1, db2
Reject         REJECT LOGON
db1_Only       PASSIVE db1
db2_Only       PASSIVE db2
```

## Error Lists in Routing Rules

A routing rule can be used to define special responses to specific error codes from the Teradata systems. See [Error Codes](#).

## Routing Rule with User Mappings

After a set of routing rules is designed to meet the variety of business needs in an environment, map incoming users to those routing rules through user mappings.

You can create a user mapping based on the following parameters.

Parameter	Description
Username	User name of the incoming user connection The * wildcard is allowed for pattern matching.
Client specified account string	Account string as specified by an incoming client connection
Default account string	Default account string as specified on the Teradata systems The list of available account strings for each user is loaded every 10 minutes (by default) by the active unity sequencer.
LDAP specified profile	Profile specified by an external LDAP server
LDAP specified role	Role specified by an external LDAP server
Unity region	Region to which the client connected Use to direct passive sessions to a preferred local Teradata system.
Time window	Pre-defined time expression that can be used to implement different user mappings at different times of the day, week, month, or year Use to shift user workloads during the business day to accommodate activities like load processes, critical reports, or daily backups. Also use for managing outage events.

## User Mappings Order

The first user mapping that matches an incoming connection is used to route the user. In general, the most selective rules should be ordered so they appear at the top of the user mapping list.

If necessary, alter the order of routing rules from Unity UI or at the unityadmin prompt.

## Default Account Strings for Groups Of Users

Use the default account string when mapping groups of users to a shared routing rule. Normally this rule is already present to provide workload control within the Teradata system. The rule provides an effective way to group users and workloads according to the business need.

### Note:

There is a delay between the time when the Unity sequencer starts and when the account strings are loaded. This may take 10 minutes depending on the setting in AccountStringScanFreq. Assign an appropriate default routing such as Reject to handle the appropriate users if the account strings were not loaded.

## Using Time Windows

The Time Windows feature allows you to select different routing rules based on a time expression.

1. From unityadmin, use one of the following commands.

Task	Command
Create a new time window	<pre>unityadmin&gt; time window update &lt;name&gt; [YEAR &lt;Start&gt; [ to &lt;End&gt;]] [MONTH &lt;Start&gt;[ to &lt;End&gt;] [MDAY &lt;Start&gt;[ to &lt;End&gt;] [WDAY &lt;Start&gt;[ to &lt;End&gt;]] [HOUR &lt;Start&gt;[ to &lt;End&gt;]] [MIN &lt;Start&gt;[ to &lt;End&gt;]] enable;</pre>
List available time windows	<pre>unityadmin&gt; time window list;</pre>
Update user mapping	<pre>unityadmin&gt; user map update TW '&lt;name&gt;' Rule &lt;rule&gt; TOP;</pre>

## Dynamic Routing Rule Changes Using Query Band Requests

You can use a query band statement to change a sessions routing rule dynamically. The query band rule makes sure that particular reporting or load processes use a specific routing rule.

If the client application is already using the routing rule, then the query band has no effect. If the client specifies a routing rule different from the currently used, it disconnects its Unity sessions and reconnects using the specified routing rule.

The example below demonstrates a report that is intended to run only on the first DB1 system, using the DB1\_Only routing rule.

```
SET QUERY_BAND = 'UNITYROUTING=DB1_Only;' For Session;
```

## Unity Session Monitoring

After you implement routing rules and create user mappings, you can monitor the results from Unity session list in the Unity UI or at the unityadmin prompt.

All session related actions, such as stopping and aborting a session, must be performed from the Unity environment and not directly on the managed Teradata system. If you attempt to manage a session on one of the managed Teradata systems, Unity treats the action as a system failure and automatically restores the affected session to its previous state.

The Unity UI and the unityadmin command line allow you to stop sessions in either passive or managed mode. Additionally, you can stop all sessions using the same routing rule from unityadmin.

You can also abort the current transaction in managed sessions from the Unity UI or the unityadmin command line. The abort command may take longer time because it must wait for a roll back of the transaction on the Teradata systems.



## Stopping Idle Unity Sessions Automatically

Set an idle session timeout for passive or managed sessions to automatically stop the client sessions that are connected but have been idle for a long time.

1. From unityadmin, do one of the following to stop an idle session automatically:

Session Type	Command Example
For passive sessions	<pre>unityadmin&gt; config update PassiveSessionIdleTimeout 1200 reason 'Idle Session timeout'; Requested config added/updated successfully. Config changes will be applied dynamically.</pre>
For managed sessions	<pre>unityadmin&gt; config update ManagedSessionIdleTimeout 1200 reason 'Idle Session timeout'; Requested config added/updated successfully. Config changes will be applied dynamically. unityadmin&gt;</pre>

2. Set the value to 0 to disable these timeouts.

# Unity UI

## Overview

Use the Unity UI to view the health of systems and clusters in real time. The Unity UI tracks the following Unity services:

- System Health
- Table Health
- Alerts
- Sessions
- Unity Storage


You can also set up Unity environments for different uses, such as testing or development and switch between overviews for each cluster.

The screenshot displays the Teradata Unity UI interface for the 'large\_tables\_cluster'. The interface is divided into four main sections:

- Managed Systems Health:** A table showing the health of two systems: 'bermuda' and 'sawgrass'. Both are active with 150,000 records. The 'Interrupted', 'Restore', and 'Unrecoverable' columns show zero counts with circular icons.
- Alerts:** A summary of alerts showing 0 Critical and 2 Warning alerts. A 'VIEW MORE' link is provided.
- Table Health:** A section showing 'No tables' with a 'VIEW MORE' link and navigation arrows.
- Sessions:** A section showing session status with 0 Root Cause, 0 All, and 0 Blocked sessions. A 'VIEW MORE' link is provided.

## Managing Configurations in the Unity UI



You can add systems and clusters from Unity to the Unity UI to view in real time.

1. Click  and do one of the following:

Option	Action
Adding a System	Select <b>Systems &gt; Add System</b> .
Adding a Cluster	Select <b>Settings &gt; Add Cluster</b> .

## Monitoring Objects and Alerts in the Unity UI

You can switch views between clusters and systems to monitor system and table health, alerts, and sessions.

1. Click  and select an object to monitor.
2. [Optional] Click  to select a different cluster to view.

## Changing the Unity UI Settings

You can add User Mappings and Routing Rules and Time Windows to the Unity UI.

1. Select , then select one of the following:

Option	For More Information
User Mappings	<a href="#">User Mappings Tab</a>
Routing Rules	<a href="#">Routing Rules Tab</a>

# Table Consistency in Unity

Unity Managed Routing can enforce data consistency across multiple Teradata systems operating asynchronously in parallel. This provides true active-active availability. Each Teradata system can run the same request in parallel, providing faster response times and zero RTO (Recovery Time Objective) and RPO (Recovery Point Objective), so that if a system fails the other systems are already up-to-date and ready to continue processing requests.

The tables written to use Managed Routing must be consistent, with the same data and table definition on all systems.

## Interrupted Tables

When Unity detects a request that failed on one system but succeeded on another, the session and tables with the failed request become interrupted. Lack of spool or perm space, missing user grants, and missing database objects can cause a request to fail.

The interrupted state means that Unity can still automatically recover the session and tables from the recovery log after the initial problem is resolved on that system.

When there is an interrupted request, Unity raises an alert to inform the DBA. The DBA can then fix the problem identified on the failing system. For example, the DBA can add more spool or perm space, correct user permissions, and create or drop required objects.

It is important to respond to the alert immediately because any subsequent work depending on the work done by the initial interrupted session also becomes interrupted. These secondary or cascade interrupts are filtered in the Unity UI and at the `unityadmin` command line so you can resolve the initial session that caused the issue.

If ignored, an interrupt alert can escalate from a small isolated issue, involving a single session and a small set of tables, to a larger problem that can cause the entire system to become interrupted.

## Skipping Interrupted Requests

In some cases, the DBA is unable to resolve an interrupted request directly on the Teradata system. For example, this may happen if a user was mapped to the wrong routing rule and accidentally executed a request that requires objects that do not exist on a given Teradata system. For these situations, it is possible to skip requests, allowing Unity recovery process to continue recovering any other workloads that were dependent on the request.

---

**Note:**

Skipping a request can result in tables becoming unrecoverable. For Unity the copy of the table is no longer synced with the copies of the table on other systems. It is not recommended to override this. The best practice is to allow the table to become unrecoverable, and then use Unity validation and resync features to synchronize the table if needed.

---

1. From the **Interrupted Sessions** screen of the Unity UI, select the session you want to skip.
2. Select **SKIP** in the **Interrupted Sessions Details** window.
3. In the **Skip Interrupted Session** dialog box, select **SKIP** again.

## Unrecoverable Tables

If a write request succeeds on a set of systems in a managed routing session but the systems return a different number of affected rows, Unity makes tables involved in the request unrecoverable on the second system to respond. Unity assumes the first successful response is correct, and relays that request to the client. After the client receives that response, the rest of the systems must agree with it.

After a table becomes unrecoverable, re-synchronize and validate the table to return to the active state. Several Unity features can be used to do this.

## Validating Unrecoverable Table

To confirm a table is in sync and all data is consistent across the systems, Unity can validate whether the table is in sync and all data is consistent across the systems. During the validation, Unity puts a read-lock on the source table to make sure the table does not change while being validated.

Use the following best practice to validate an individual unrecoverable table.

1. In the Unity UI, select **OBJECTS**.
2. Select an object to validate.
3. In the **Validate object data** window, select **VALIDATE**.

If the table is in sync, the validation request automatically activates the unrecoverable copy of the table.

## Self-Healing Unrecoverable Tables

Unity self-healing feature can automatically validate, re-synchronize, and re-activate unrecoverable tables. Tables may become unrecoverable when a session encounters data inconsistency with multiple tables involved in the request. Only one of the involved tables may be out of sync. The rest of the tables can be automatically re-synchronized after they are validated.

If the self-healing feature is enabled, Unity runs it automatically, with default intervals of 10 minutes, to re-activate unrecoverable tables found to be in-sync. The Unity table synchronization service automatically re-synchronizes other out-of-sync tables.

Enable this feature with the following unityadmin command:

```
config update AutoSelfHealingInterval 600 reason 'Enable Self-healing';
config update AutoTableValidateAttempts 1 reason 'Enable Self-healing';
```

You can optionally set this self-healing feature to validate only tables or to validate and then synchronize tables using the configuration setting **AutoSelfHealingOptions**.

- If set to 0, self-healing validates and activates the out-of-sync table.
- If set to 1, self-healing validates, synchronizes, and then activates the out-of-sync table.

```
config update AutoSelfHealingOptions 1 reason 'Enable Self-healing table sync';
```

You can specify replication options for individual tables. If not specified on an individual table, the **DefaultObjectResyncOption** is used to determine whether the table is synchronized.

```
config update DefaultObjectResyncOption 2 reason 'Enable Self-healing
table sync';
```

You can also specify a timeout for each table for the replication option. If not specified, the default value set by the configuration setting is used.

```
config update DefaultObjectResyncTimeout 600 reason 'Setting global
resync timeout';
```

## Re-Synchronizing Unrecoverable Tables

Unity provides a way to re-synchronize tables that are genuinely out of sync. Unity issues a resync request to re-synchronize a system.

1. Use one of the following methods to re-synchronize an unrecoverable table.

Method	Steps
Unity UI	<ol style="list-style-type: none"> <li>a. From the Unity UI, select <b>OBJECTS</b> or <b>DATABASES</b>.</li> <li>b. Select an object or a database to synchronize.</li> <li>c. Do one of the following: <ul style="list-style-type: none"> <li>• In the object data window, select <b>Sync</b>.</li> <li>• In the database data window, select <b>Sync Database</b>.</li> </ul> </li> </ol>
unityadmin	<ul style="list-style-type: none"> <li>• <code>unityadmin&gt; help object resync;</code>  Command: <code>OBJECT RESYNC &lt;tablename&gt; id:&lt;object id&gt; FROM &lt;tdpid1&gt; TO &lt;tdpid2&gt; [TIMEOUT '&lt;timeout (seconds)&gt;']</code>  Description: Synchronizes the specified table from the source system to the destination system   <code>unityadmin&gt; OBJECT RESYNC table1 FROM db1 TO db2 TIMEOUT 600;</code> </li> <li>• <code>unityadmin&gt; help database resync;</code>  Command: <code>DATABASE RESYNC &lt;database name&gt; FROM &lt;tdpid1&gt; TO &lt;tdpid2&gt; [TIMEOUT '&lt;timeout (seconds)&gt;']</code>  Description: Synchronizes all the tables in the specified</li> </ul>

Method	Steps
	<pre>database from the source system to the destination system  unityadmin&gt; DATABASE RESYNC db1 FROM 1 TO 2 TIMEOUT 600;</pre>

## Manually Re-Synchronizing Unrecoverable Tables

When it is clear how a data inconsistency happened, it is sometimes possible to re-synchronize a table faster by manually updating the unrecoverable table. For example, a DBA may manually delete data that was accidentally loaded directly to a system, remove incorrect time ranges, or undo changes that happened outside of Unity. In these situations, once the changes have been made, it is the best practice to run a table validation from Unity to activate the table.

# Teradata System Outages

## Planned Outages

Unity has the ability to sustain long planned outages without taking applications offline. This ability can be used in situations from short outages for simple maintenance work or restarts to longer outages for Teradata system expansions, upgrades, or full replacements.

Unity tracks the Teradata system state which can be changed to control access to the system. The system state is employed for passive and managed routing. Using the HALT operation, you can close down and stop all sessions on a Teradata system. If a user routing rule allows the sessions to fail over to the second system, the sessions can fail over when the system becomes out of service. Unity recovery of managed sessions is the second mechanism that allows Teradata systems to be re-synced automatically.

Unity can manage outages on three levels – the entire system, a database, or an individual table.


## Allowed Time for a Planned Outage

For passive routing, there is no limit to how long a system can be out of service. For managed routing sessions, the amount of time a Teradata system can be out of service for a planned outage is limited by the volume of data and SQL requests executed in managed sessions on the other active Teradata systems. The data is stored in the data file system on the Unity servers. Each Unity server has 10-15 TB of storage space available. Adding Unity servers as expansion servers can increase this space.

## Shifting Passive Users Off the System

To gradually shift passive users off a system ahead of a system outage, fail over the session.

1. Use one of the following methods to perform a failover.

Method	Steps
Unity UI	<ol style="list-style-type: none"> <li>a. From the Unity UI, select <b>Sessions</b>.</li> <li>b. Click , then select <b>Failover</b>.</li> </ol>
unityadmin	<pre>unityadmin&gt; help session failover; Command: SESSION FAILOVER { &lt;unity session id&gt;   &lt;routing name&gt; } Description: Moves the preferred system for the session/routing to the next available system in the ordered list. unityadmin&gt; SESSION FAILOVER &lt;unity session id&gt;   &lt;routing name&gt;;</pre>

The best practice is to use a Time Window to shift users automatically ahead of the outage.



---

**Note:**

You do not need to fail over Managed Sessions, which stop on the Teradata system automatically when it is moved to the out-of-service state.

---

## Time Monitoring during an Outage

While a Teradata system is in the disconnected, interrupted, or unrecoverable state, Unity monitors how far behind the system is and the time left to return the system to the active state before it can no longer be recovered and falls to the unrecoverable state.

## Time to Recover System

When a planned or unplanned outage is complete, a DBA can initiate a full system recovery to bring the system back to the active state by re-running all the requests sent in managed sessions. The time to recover depends on such factors as the size of the Teradata system and the concurrency of the workloads. In a typical environment that includes a mixture of applications performing reads and writes, the recovery of a system takes less than the total time the system was down. During the recovery, system only needs to re-run the write requests that were missed.

In a workload with numerous write requests, with little or no read traffic, the time to recover a system may be longer and may equal the time of the outage. For example, it happens if the client workload continues to drive the active systems at their full throughput while the recovery system is trying to catch up. While the system is recovering, Unity recovery log and data file system provide the recovering system extra capacity to sync. The log and data files store incoming write requests at the top of the recovery queue as the recovering system is re-running them from the bottom of the queue.

## Monitoring System Recovery after Long Outage

After a long outage, monitor the progress of the system recovery and respond to alerts or interrupted sessions. The IDs assigned to client sessions are used to sequence requests during recovery. If a request on a session ID fails during recovery with an interrupt, it blocks later sessions in the recovery that re-use the same session ID. To avoid these problems, address the issues that appear in the recovery process immediately.

Use the interrupted session screen to monitor issues that appear during recovery. The interrupted sessions are divided into **Root causes** and **Secondary**. It is normal for sessions to become interrupted if they are waiting on other sessions to finish their recovery first. These sessions appear as **Secondary** and no actions need to be taken to address them.

If a request repeatedly fails to recover, you can skip the request. If there are tables involved, you should use the option to make the tables unrecoverable.

1. Use one of the methods below to monitor a system recovery.

Method	Steps
Unity UI	Select <b>Systems</b> , then select a system for which you want to view the recovery status.
unityadmin	<pre>unityadmin&gt; status;</pre> <p>OR</p> <pre>unityadmin&gt; system recovery status;</pre>

## Planned Unity Server Outage


For best practice, perform Unity maintenance in a rolling fashion. Start on the Unity server with the Standby sequencer and finish with the Active sequencer.

### Note:

Shutting down the Active sequencer triggers a brief outage to fail over the sequencer.

## Shifting Off Client Connections

Before starting maintenance on a Unity server, clients that are connected through that server must shift to the other Unity servers. For best practice, stop new connections on the Unity server endpoint several hours before the planned maintenance.

1. In the Unity UI, select **Components**, then select **Endpoints**.
2. Click  to disable the endpoint.

### Note:


Monitor the number of sessions connected through a particular endpoint using the **Sessions** screen in the Unity UI or using the STATUS command at the unityadmin prompt.

### Postrequisite:

Enable the endpoint after the planned maintenance.

## Using Failover for the Unity Sequencer

When it is time to proceed to the Unity server with the active sequencer, you can fail over the sequencer to the second Unity server. This causes an interruption because the standby sequencer must become active, wait for the other components to re-connect, and then initiate a recovery of the managed Teradata systems.

1. To put all Teradata systems in an out-of-service state, select **Components**, then select **Endpoints**.
2. Click  to fail over the sequencer process.

**Note:**

You can also use the `unityadmin` command `SEQUENCER FAILOVER`.

## Unplanned Outages

### Unexpected Teradata System Failure

If a Teradata system fails unexpectedly, Unity automatically re-routes client applications using passive or managed routing if the routing rules allow the applications to run on another system.

When a system fails unexpectedly, managed sessions continue without noticeable interruption because the managed sessions are already connected to multiple systems in parallel. If a managed session is reading a large multi-packet response from the failing system, the current request fails, with a limited impact.

Passive sessions automatically reconnect to the other Unity system. In such cases, passive sessions lose inflight volatile data but are able to continue processing.

The failed Teradata system is marked Interrupted and new connections are automatically diverted from connecting to it until it reconnects to the Teradata system using the `unitymgmt` user. Unity automatically re-processes requests that the failed Teradata system missed while it was down. When the system is restored, Unity returns it to the Active state and resumes online work on the system.

### Unexpected Unity Server or Sequencer Failure

If the active sequencer or the Unity server on which it resides fails, the standby sequencer automatically assumes the role of the active sequencer after a specified timeout period (with a default of 5 minutes). This happens if the standby sequencer can connect to the database running on all managed Teradata systems in both data centers. If the standby sequencer cannot connect to all Teradata systems, it does not attempt to take over, in case there is only a network issue and not an actual failure of the Unity server.

Use the Component view in the Unity UI to determine which component or Unity server is down.

A `SEQUENCER STATUS` command issued on the standby sequencer displays the time remaining before the standby sequencer automatically assumes the active role. It also displays the checks performed by the standby to confirm the other Unity server is down.

```
unityadmin> sequencer status;
HA Status           : Down/Unknown
Log Replication Mode : Asynchronous
Failover Mode       : Automatic

Automatic Failover Status
-----
Sequencer           : region2_seq
```

```

Time Remaining Before Failover      : 290 seconds
Failover Conditions                  :
    Check standby sequencer can connect to all managed systems (done)
    Check active sequencer has no heartbeat on all managed systems (running)
    Check active sequencer has no transaction on all managed systems (running)

Individual Sequencer States
-----
    unity2:5344                      : STANDBY (SYNCHRONIZED)
    unknown host/port                : DOWN
...

```

## WAN or Data Center Failures

In an environment with two Unity servers, each Unity server is located in a different data center to protect against the general failure of the data center. The following are the most common causes of a data center failure.

- Air conditioner or cooling problems
- Power outages
- Network connectivity problems

It is **extremely** important for the operator to assess the situation and determine if there are data center wide issues or if the failure is isolated to the network connecting the two data centers.

The SEQUENCER STATUS command at the unityadmin prompt reports the current status and location of the active sequencer. This command can be issued while connected to either of the Unity servers.

---

### Note:

In the event of a WAN or data center outage between two data centers, **before taking any action**, an operator in the second data center must confirm the primary data center is down for an extended outage because of a serious disaster.

---

## WAN Outage

If the issue is a network WAN outage, it is most likely that the initially active sequencer is still active and the work in the primary data center was not interrupted. In this situation, the work in the second data center with the standby sequencer is halted because it cannot connect to the active sequencer. However, any applications connecting to the active sequencer are still online. For this reason, there is no automatic failover to the standby sequencer.

**Note:**

If the issue is a WAN outage, **do not fail over to the standby sequencer**. Doing so is unnecessary and can result in a split-brain failure leading to irresolvable data inconsistencies. The correct course of action is to work on resolving the network outage.

## True Primary Data Center Outage

When the entire primary data center, which houses the Unity server with the active sequencer, is down and the issue cannot be resolved in a short time, the standby sequencer located on the second Unity server (usually located in a second data center) can take over the role of the active sequencer. Use the SEQUENCER FAILOVER command at the unityadmin prompt for the second Unity server.

```
unityadmin> sequencer failover;
W Warning: SEQUENCER FAILOVER can cause client connection errors (but
transactions will be completed).

W READ CAREFULLY! This command should only be executed when you have confirmed
the active sequencer on the other unity server has been shutdown or is known to
be DOWN.
W Do NOT use this command if the WAN between the two unity servers is down,
(because the other unity server might still be up and processing connections).
W Have you:
W      1. Confirmed the other ACTIVE sequencer is DOWN, or other unity
server is DOWN?
W      Or... 2. Halted all systems for a planned failover?

Takeover has succeeded. This sequencer is now ACTIVE.
unityadmin>
```

**Note:**

The SEQUENCER FAILOVER command helps to make sure the other Unity server is really shut down.

After the connectivity to the failed data center is restored, before allowing the Unity servers to reconnect to each other, make sure the old recovery log on the formerly active Unity sequencer is deleted.

If the connectivity to the failed data center is restored before the sequencer in that data center is reset, it may trigger a split-brain shutdown. Both sequencers must be shut down so an operator can choose an active sequencer. This helps to avoid missed transactions in the recovery log of the now active sequencer. The sequencer roles can be switched later during a planned outage.

Following the failover of the Unity sequencer, the endpoints on the standby Unity server start listening on their primary and standby IP addresses. Clients can then reconnect through this Unity server.

## True Secondary Data Center Outage

If the secondary data center, which houses the Unity server with the standby sequencer, fails there is a low availability impact. Applications connected through the primary data center are largely unaffected. There may be a small number of request errors in limited situations, but the majority of users in the primary data center do not notice the failure. Users connected through the Unity server in the second data center are disconnected. These users can immediately reconnect using standby IP addresses or another redirect mechanism. No operator action is required for Unity or Loader to react to the outage.

## Partial Unity Process Failures

The Unity software stack consists of four components: the Endpoint, Watchdog, Sequencer, and Dispatcher. These components include built-in high availability. The Unity Watchdog, Endpoint, and Dispatcher have built-in automatic capabilities to handle failures. The dispatcher always has an associated standby process running on the second Unity server. That server becomes active immediately if the active dispatcher for a Teradata system fails. All of the processes restart automatically if they fail unexpectedly, after a period of 5 minutes (for the Endpoint, Sequencer, or Dispatcher) or 10 minutes (for the Watchdog).

# Security and Authentication

## Providing the Required Passwords to Teradata for Change Controls

Configuring additional Unity servers or Teradata systems in the Unity environment requires usernames and passwords. Unity has a tool to provide these credentials in a convenient and secure way. This is the recommended way to collect the usernames and passwords required to install and configure Unity.

1. On the primary Unity server, use the `unityPasswd` tool.

```
# /opt/teradata/unity/bin/unityPasswd

Welcome to the Teradata Unity password tool.

This interactive tool will allow you to set passwords required for
maintenance on
  your Unity & Teradata environment in a safe an secure way. The password
values set
  will be stored in an encrypted format in TDWallet on this server.

This tool simply collects passwords. It does not perform any changes on the
unity
  servers or Teradata systems. It will validate the passwords you have entered
are
  correct by attempting to connect to the unity server or Teradata system.

This tool will NOT affect or change any passwords or value required to run
Unity.
  These values are only used for maintenance.

At the conclusion of the maintenance, you can remove these passwords
using:
  # unityPasswd --erase
  Removing any current passwords.

-----
-----
  Would you like to provide the credentials to configure a unity server?
```

```

-----
-----
Would you like to provide the credentials to configure a:

1. Unity server?
2. Teradata system
Please enter 1 or 2:

```

The unityPasswd tool attempts to verify the passwords by connecting to the specified systems or Unity servers.

2. Verify the passwords before starting a configuration change:  
# /opt/teradata/unity/bin/unityPasswd --check
3. When the configuration change is complete, remove stored passwords using the --erase option to secure the environment:  
# /opt/teradata/unity/bin/unityPasswd --erase

## Changing the Unity Management User Password

To monitor the Teradata systems managed in a Unity environment, use the unitymgmt user. By default, this user is created automatically on a Teradata system when it is first added. The password for this user is initialized to a random string (up to 32 characters long, consisting of alphanumeric and special characters), and stored in tdwallet for security. It is not necessary for the DBA or operator to know the password.

1. If it is necessary to update this password value, run one of the following commands:

To set the value of the unitymgmt password to a known value	At the unityadmin prompt, run <code>UNITYMGMT PASSWORD SET &lt;new_password&gt; [REASON '&lt;reason&gt;']</code>
To update the current password of the unitymgmt user on a Teradata system to the current secret value in the Tdwallet	From the Unity server, run <code># setUnitymgmtPassword -s [tdpid] ( -d [dbc] ) ( -dP [dbc password] )</code>

## Changing the Unity Admin Password Used by Viewpoint

Follow these steps to change the Unity administrator password used by the Viewpoint portlets to retrieve information from Unity.

1. From unityadmin, update the Unity admin password:  
`ADMIN CHANGE PASSWORD <adminuser> <newpassword>;`
2. On each Viewpoint server, update the Unity admin passwords as a Viewpoint user using the ud-commons.jar file:  
`/opt/teradata/jvm64/jre7/jre/bin/java -cp /opt/teradata/unityui/scripts/  
ud-commons.jar com.teradata.unity.security.PasswordManager  
UNITYADMIN_PASSWORD <newunityadminpassword>`



## Updating Passwords for Unity System Accounts

In every Unity environment, there are multiple servers where passwords must be updated and kept in sync.

### Changing the Default Password

The default Unity Administrator password must be changed as soon as possible.

---

#### Important:

You must retain a record of the new password because it is not possible to restore or recover it.

---

1. Browse to `/opt/teradata/unity/bin/`.
2. Change the default Unity Administrator password:  

```
unity admin -u admin -P admin -e "admin change password admin password"
```

### Changing the unitysync and root OS User Password

On every Unity managed server, as well as the root account, there is a service account named 'unitysync'. This account is used for syncing configuration information between Unity servers. The password for these service accounts can be changed without any special steps.

### Changing the Unity Management Password

The Unity Management user 'unitymgmt' is used by Unity to connect to the managed Teradata systems. To update the password:

1. At the unityadmin prompt on the Active Sequencer, enter the following command:  

```
UNITYMGMT PASSWORD SET <new_password>
```
2. On ALL managed systems or through Unity, enter the following command:  

```
MODIFY USER unitymgmt AS PASSWORD='<new_password>';
```

## Authenticating Unity Admin Users with LDAP

Optionally, you can configure Unity to authenticate unityadmin users using an LDAP server. Users can be configured in an admin or readonly role. Users with the readonly role are not able to change any configuration, system, or object state within Unity.

1. Enable LDAP authentication by setting the configuration setting **LDAPServerURI**.

```
unityadmin> config update LDAPServerURI ldap://server.domain.com:389/ REASON
'ldap config';
```

---

**Note:**

You need to configure this setting only once.

---

2. Add the user using the ADMIN ADD command, use LDAP for the password.

```
unityadmin> admin add username 'First, Last' LDAP readonly;  
Successfully added the administrator 'username' (role readonly)
```

# Unity Maintenance

## Changing the State of a Table for Maintenance

1. In the **Unity** UI, select the **Objects** page.
2. Select one or more tables to change.
3. Select the Change state icon to change the status to **Out of Service** or **Read-only**.
4. Select **Change State**.

## Changing the State of a System for Maintenance

Follow these steps to change the state of a system before and after maintenance.

1. From the **Unity** UI, select **Systems**.
2. Select the Change state icon.
3. Select **Out of Service**.
4. Select **Change State**.

## Failing Over to the Standby Sequencer for Planned Maintenance

Follow these steps for a manual failover during planned maintenance.

1. From unityadmin, log on to the active sequencer.
2. Stop all systems:  
`unityadmin> SYSTEM HALT <system1>, <system2>...;`
3. Log on to the standby sequencer.
4. Fail over to the standby sequencer:  
`unityadmin> SEQUENCER FAILOVER;`

## Blocking Incoming User Sessions for Maintenance

You can use the following options to block user sessions. You can use them separately or in conjunction.

1. From unityadmin, run one of the following commands:
  - [ENDPOINT DISALLOW NEW CONNECTIONS](#)
  - [REGION DISABLE LOGONS](#)

## Shutting Down Unity

In situations where you must stop all processes and shut down Unity, do the following:

1. Shut down the processes on each Unity server: `/usr/bin/unity stop`.  
Shut down the Unity server with the active sequencer last.

## Deadlock Detection

A deadlock happens when two or more transactions are waiting for each other to give up locks. Three different types of deadlocks can occur with Unity.

### Deadlocks within Unity-Managed Sessions

Unity performs deadlock detection for all managed sessions (sessions associated with managed routing rules). Since Unity does its own locking for all requests of managed sessions, Unity detects deadlocks before they have a chance to happen on the underlying managed systems. Unity resolves the deadlock at the time of detection.

### Deadlocks on Underlying Unity-Managed Systems

With passive sessions, deadlocks at the database are possible. Because passive sessions bypass Unity standard locking and deadlock detection and are sent straight to the system (through the dispatcher process), deadlocks can occur at the Teradata system. Once Teradata detects a deadlock, it automatically takes appropriate action to resolve the deadlock.

### Deadlocks Between Teradata and Unity

With Unity passive routing, deadlocks may not occur on the managed server or within Unity independently, but the system as a whole deadlocks. Unity periodically obtains blocking locks information from the underlying systems, using the Teradata PMPC API. This lock information is combined with Unity lock information to detect deadlocks. If Unity finds a deadlock, Unity or the Teradata server takes action to resolve the deadlock.

## Setting the Global Deadlock Detection Interval

A global deadlock refers to a deadlock between Unity and Teradata Database. A global deadlock may occur when Unity uses passive and managed routing at the same time with the option `Passive Routing Allow Managed Writes` selected.

You can change how often Unity attempts to detect a global deadlock between Unity and a Teradata system.

---

### Note:

Because of performance degradation, enable Global Deadlock Detection only if you encounter deadlock issues.

---

1. From `unityadmin`, enter the following command with the new time interval:

```
CONFIG UPDATE globaldeadlockdetectioninterval <numberofseconds> REASON 'Enabling
global deadlock detection';
```

#### Related Information:

[Global Configuration Parameters](#)

## Best Practices for Backup and Restore in a Unity Managed Environment

Unity provides synchronized data across multiple Teradata systems, protecting against the loss of data during system or data center failure. Some situations still require backup and recovery. Database backups are necessary to protect against logical damage or corruption to the database, for example due to human errors.

### Back Up Single or Multiple Managed Systems

For environments that use Unity on two or more Teradata systems with identical content in the same data center, a single backup is enough for both systems. If systems are located in separate data centers, it is recommended to maintain backup copies in both locations. This protects against the loss of a data center and ensures that if a restore is needed, a local backup is available. This protects against transferring large amounts of data across the WAN joining the two data centers.

### Using Standard Offline Backup

Unity allows applications to run on a second Teradata system while the offline backup has tables locked.

Use the unityadmin shell.

1. Start a Read-only outage on the table, database, or system that you want to archive:

Option	Description
table	<code>unityadmin -u admin -P admin -e 'object freeze &lt;db.tablename&gt; on &lt;system_name&gt;'</code>
database	<code>unityadmin -u admin -P admin -e 'database freeze &lt;db_name&gt; on &lt;system_name&gt;'</code>
system	<code>unityadmin -u admin -P admin -e 'system freeze &lt;system_name&gt;'</code>

For example, to begin an outage on a table named 'dbtest.table1' on Teradata 'system1', enter the following: `unityadmin -u admin -P admin -e 'object freeze dbtest.table1 on system1'`

2. Recover the system, database, or table:

Option	Description
table	<code>unityadmin -u admin -P admin -e 'object recover &lt;db.tablename&gt; on &lt;system_name&gt;'</code>

Option	Description
database	<code>unityadmin -u admin -P admin -e 'database recover &lt;db_name&gt; on &lt;system_name&gt;'</code>
system	<code>unityadmin -u admin -P admin -e 'system recover &lt;system_name&gt;'</code>

## Using an Online Backup

Backups can also be taken using an online backup. Online backups take longer than offline backups, but you do not need to make the system, database or table READONLY before running the backup.

## Restoring to a Teradata System Managed by Unity from a Backup

You can do a complete restore of all copies of a system, database, or table from a backup archive. Restoring a table, database, or system requires a complete outage of the object. Follow these steps:

1. Use the unityadmin shell to deactivate the object so it is no longer accessible through Unity:

Option	Description
table	<code>unityadmin -u admin -P admin -e 'object deactivate &lt;dbtest&gt;'</code>
database	<code>unityadmin -u admin -P admin -e 'database deactivate &lt;dbtest&gt;'</code>
system	<code>unityadmin -u admin -P admin -e 'system deactivate &lt;system1&gt;'</code>

2. Restore all copies of the object to the same point.
3. Make sure all copies of the system, database, or table are in-sync.
4. Use the unityadmin shell to activate the object so that, moving forward, all writes are recorded for the object in the Recovery Log:

Option	Description
table	<code>unityadmin -u admin -P admin -e 'object activate &lt;dbtest&gt;'</code>
database	<code>unityadmin -u admin -P admin -e 'database activate &lt;dbtest&gt;'</code>
system	<code>unityadmin -u admin -P admin -e 'system activate &lt;system1&gt;'</code>

# Viewpoint Unity Setup Portlet

The **Unity Setup** portlet lets you customize how Unity works. Use the **Unity Setup** portlet to do the following:

- Edit User Mappings, Routing Rules, Time Windows, and ErrorList profiles.
- Select alerts to use.
- Change threshold settings.
- Modify global settings for the portlet.

## Session Routing Tab

The **Session Routing** tab lets you specify how sessions route requests from clients to Teradata systems. The Session Routing tab lets you view deployed session routing. You can also create or modify user mappings, routing rules, and error profiles.

---

**Note:**

When you edit user mappings, routing rules, or error profiles, Unity locks the view, and no other user is allowed to edit. The Session Routing views show only the data available from the Sequencer to other users. The lock is released when the **Edit Locks** setting expires on the **Global Settings** tab. Locks occur separately on a per page basis for each portlet instance.

---

## User Mappings Tab

The **User Mappings** tab allows you to define session routing based on user logon properties. Unity relies on the user mapping to determine how to route the session.

Unity looks for the first rule in the user mapping list that maps to the logon properties of the requesting user or client application. This is the rule used to define the routing selected for the session. If no rules match, the Default rule is used. Changes to user mappings are recorded in audit logs. When associated with a routing rule, Unity matches the properties of a user logon session request with the logon requirements of a user mapping, and then determines the routing of the new session.

You can create multiple user mappings to match different user and client logons for an ecosystem.

The **User Mappings** tab shows the following information for each user mapping.

Field	Description
Order	Evaluation order of the user mapping.
User, Account, Role, Profile, and Region	User information associated with user mapping.
Routing Rule	Routing rule selected for user mapping.
Time Windows	Defines time triggers for user mapping.

Field	Description
Deploy Status	Status of rule.



## Adding a User Mapping

User mapping rules are made up of an ordered list of filters that map a logon request to a specific routing rule. When you add a user mapping, you define which logon properties are assigned to a routing rule.

### Note:

When you create a user mapping, the user role or profile you enter must already exist on the Teradata system. Roles and profiles are user-level security controls for the Teradata Database.

Use the wildcard (\*) character when entering **User**, **Account**, **Role**, **Profile**, and **Region** string values as needed. Use a single wildcard (\*), a leading or trailing wildcard (\*user or user\*), or a leading and trailing wildcard (\*user\*).

1. In the **Unity Setup** portlet, click the **Session Routing** tab.
2. Click **User Mappings** tab.
3. Click  **Edit Session Routing**.
4. Next to **User Mappings**, click .
5. In **Add User Mapping**, complete the following fields:

Field	Description
Evaluation order	Number indicating the ordered ranking of this user mapping in the <b>User Mappings</b> list. Unity checks user logon properties against the prioritized <b>User Mapping</b> list and uses the first mapping that matches. For example, if there are five user mappings, you can enter the number 6 for the least priority or any number between 1 and 5 to increase priority. If you use a number that already exists, the rule currently at that number and those below increase their evaluation order by 1.
User	Teradata username (userid). The user-string can be a complete string (user) or include one or more wildcard (*) characters.
Account	Account string value. The account string can consist of alpha-numeric and wildcard (*) characters.  <b>Note:</b> Account strings are loaded periodically, every 10 minutes by default, on each system. If there are changes with a user account string, an alert is sent to the Unity administrator. This alert must be manually cleared by the administrator. If this alert persists, it could indicate the user account strings does not match each of the Teradata systems. Administrator should validate the account strings for all users match on all managed systems.
Role	User role for this mapping. The role is an LDAP or Kerberos value and can be a complete string (for example, crmusers) or a wildcard (*) character to include multiple roles.

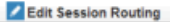


Field	Description
Profile	Name of the profile associated with the user. Profile only applied for external authentication (LDAP/Kerberos).
Region	Region name from the list or <b>Any Region</b> (*). The <b>Any Region</b> option applies this user mapping to all defined regions.
Routing rule	Routing rule name from the list.

- Click **Save**.  
The new mapping appears in the **User Mappings** list.
- Click **Deploy** to deploy the user mapping.  
Click **Revert to Deployed** to discard all changes.



## Editing a User Mapping

Unity locks the **User Mappings** view while you edit.

- In the **Unity Setup** portlet, click the **Session Routing** tab.
- Click the **User Mappings** tab.
- Click .
- Select the name of a user mapping.
- In the **Edit User Mappings** view, update the appropriate fields.
- Click **Save**.
- Click **Deploy** to deploy edits to the user mapping.

## Deleting a User Mapping

Before you delete a user mapping, verify session routing is not using the user mapping.

- In the **Unity Setup** portlet, click the **Session Routing** tab.
- Click the **User Mappings** tab.
- Click .
- Click  next to the user mapping you want to delete, and select **Delete**.
- Click **Delete** to confirm.

## Searching for a User Mapping

You can search for a user mapping. Enter the evaluation order for the mapping or enter a value in the corresponding filter box.

- In the **Unity Setup** portlet, click the **Session Routing** tab.
- Click the **User Mappings** tab.

3. Choose one of the following search methods:

Option	Description
Identification number of the mapping	In the <b>Order</b> filter box, type the number associated with the user mapping for an exact match or use > (greater than) or < (lesser than) to indicate a range; for example, >120.
userid, accountstring, role, profile, region, routing rule	In the corresponding filter box, type the search value for the mapping to return an exact match. You can use wildcards (* and ?) to expand your search.
Time Windows	In the <b>Time Windows</b> filter box, type the search value for the mapping to return an exact match. You can use wildcards (* and ?) to expand your search.

## Routing Rules Tab

The **Routing Rules** tab shows the following information about routing rules.

Field	Description
Rule	Name of the routing rule. DefaultRouting cannot be removed or renamed.
Rule Type	Type of routing rule: Managed or Passive. Managed routing rules define how read and write requests are distributed across Teradata systems. Passive rules define which system a passive session is opened on. All read and write requests go to that system.
Read Routing	Method for read routing: Default or Preferred. Default routes to any system that matches the DefaultRouting rule. Preferred routes to a specified system or systems in order.
Create Routing	Method for how new objects are created: Preferred, Balanced, or All. Preferred specifies the system or systems in order. Balanced specifies the systems that are used to load balance. All specifies the default method.
Balancing Method	Method for balancing sessions: First Available, Least Used, Round Robin, and Weighted Round Robin. First available is the first system that can open a session. Least used specifies the system with the most available space. Round robin is a looping list of systems in which to open one session each. Weighted round robin is a looping list of systems and associated number of sessions in which to open.
Systems	Specified system for routing rule.
Error Profile	Error profile associated with routing rule.
User Mappings	User mappings associated with routing rule.

## Routing Behavior Summary

Unity routes requests by request type (Read, Write, or Read/Write) or by connection type (Balanced, Default, Preferred, or Passive).

## Routing by Request Type

The following table shows how various types of requests are routed in sessions that are managed (associated with a managed Routing Rule). For passive sessions, all queries go directly and exclusively to the single system where the session is open.

Request Type	Routing Rule	Types
Read	Specifies how Read requests are routed across Teradata systems.	Default, Preferred, and Create Routing-Preferred.
Write	Specifies which systems are involved in processing Write requests for a session.	Default, Create Preferred, and Create Balanced.
Read/Write	Specifies how Read and Write requests are routed for a session.	Default and Preferred.

## Routing by Connection Type

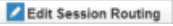

Session routing determines how Read and Write requests are routed from the client application or user to Teradata Database systems. This table shows routing behaviors based on connection type.

Connection Type	Session Source	To Add Routing Rule	Routing Behavior
<b>BALANCED:</b> User connection, as defined in user mapping, connects to multiple systems that are load balanced for optimum performance.	<ul style="list-style-type: none"> <li>Supported for Write session routing only.</li> <li>Unity connects the client to all Teradata systems identified by the user mapping and routing rule.</li> <li>If logon fails, log entry is created.</li> <li>An error returns to the application only when all logons fail.</li> <li>Active sessions are connections to the load-balanced Teradata system.</li> <li>Other connections are passive until failover occurs.</li> </ul>	<ul style="list-style-type: none"> <li>Specify two or more Teradata systems.</li> <li>Specify "Balanced" to set load balancing for Write requests to systems.</li> <li>First system name in list cannot start with a wildcard (*).</li> </ul>	<ul style="list-style-type: none"> <li>Failover is possible.</li> <li>Active sessions are constantly changing based on load balancing requirements.</li> <li>Failover occurs on errors only if the error code profile setting is "Fail over".</li> <li>Exit occurs on errors only if the error code profile setting is "Exit".</li> </ul> <p>If <b>CREATE ROUTING - BALANCED</b> routing is selected:</p> <ul style="list-style-type: none"> <li>Request submitted only to the designated system for the session.</li> <li>If request fails, the request is submitted to the next system on the Write list in a valid state.</li> <li>All systems in the list are tried until the session request is successfully processed.</li> <li>Read list must contain all systems in the Write list.</li> </ul>

Connection Type	Session Source	To Add Routing Rule	Routing Behavior
<p><b>DEFAULT:</b> User connection that matches the defined user mapping, connects to any system defined by the Default routing rule. Used by Unity when a routing rule is not specified for the <code>userid</code> in the user mapping.</p>	<ul style="list-style-type: none"> <li>Supported for both Read and Write requests.</li> <li>For any user, Unity connects to all Teradata systems specified by the Default routing rule.</li> <li>If logon fails, a log entry is created. An error is returned to the application only when all logons fail.</li> <li>Users are logged on to the Teradata system defined in the Default routing rule.</li> </ul>	<ul style="list-style-type: none"> <li>Select the <code>DefaultRouting</code> routing rule when associating user mappings with default routing behavior.</li> <li>Used if there is no user mapping for the <code>userid</code>.</li> </ul>	<ul style="list-style-type: none"> <li>Failover and Exit are possible and occur as defined by the Default routing rule.</li> </ul>
<p><b>PREFERRED:</b> User connection, as defined in user mapping, connects to</p> <ul style="list-style-type: none"> <li>Single named Teradata system, or</li> <li>Multiple systems in a preferred order.</li> </ul> <p>Options are:</p> <ul style="list-style-type: none"> <li>Read Preferred</li> <li>Create Routing Preferred</li> </ul>	<ul style="list-style-type: none"> <li>Supported for both Read and Write requests.</li> <li>Unity connects the client to all Teradata systems identified by the user mapping and routing rule.</li> <li>An error returns to the application only when all logons fail.</li> <li>Active session is the connection to the preferred Teradata system.</li> </ul>	<ul style="list-style-type: none"> <li>For single system for routing both Read and Write requests, specify one Teradata system.</li> <li>For multiple systems in a preferred order, specify two or more Teradata systems.</li> <li>First system name in list cannot start with a wildcard (*).</li> <li>Select <b>Preferred</b> to indicate that the order of Teradata systems in the list is relevant to routing operations.</li> <li>Prioritize order of systems to determine failover order.</li> <li>Associate user mappings with the Preferred routing rule.</li> </ul>	<ul style="list-style-type: none"> <li>No failover occurs for single named Teradata systems, and errors are returned to the application.</li> <li>Failover is possible in a multiple system configuration.</li> <li>Active sessions fail over to the next Teradata system specified in the routing rule system list.</li> <li>Failover occurs on errors only if the error code profile setting is "Fail over".</li> <li>Exit occurs on errors only if the error code profile setting is "Exit".</li> </ul> <p>If <b>CREATE ROUTING - PREFERRED</b> routing is selected:</p> <ul style="list-style-type: none"> <li>Request submitted to the first system on the WRITE list that is in a valid state.</li> <li>If request fails, the request is submitted to the next system on the WRITE list in a valid state.</li> <li>Sessions to the Teradata systems are closed after</li> </ul>

Connection Type	Session Source	To Add Routing Rule	Routing Behavior
			<p>all pending requests are processed by the system.</p> <ul style="list-style-type: none"> <li>Read list must contain all systems in the Write list.</li> </ul> <p><b>Note:</b> If both Read Preferred and Create Routing Preferred options are specified, the system lists must be identical, Moving or failing over data requires that the new preferred system be able to apply the rules of both the Read Preferred and Create Write Preferred routing.</p>
<b>REJECT LOGON:</b> Connections from the specified user are rejected by Unity.	Sessions are not allowed, and the connection is rejected.	Not applicable	Not applicable
<b>PASSIVE:</b> User connection matches a User Mapping, as defined in the associated passive routing rule. This rule is used to open a connection to a single system.	<ul style="list-style-type: none"> <li>Supported for all clients that only want to read/write data to one system</li> <li>Unity connects the client to one system only</li> <li>If a logon fails, other systems in the routing rule (if any) are tried. Logon failures are returned to the user if no systems can establish a session.</li> </ul>	<ul style="list-style-type: none"> <li>Select <b>Passive</b> when adding the Routing Rule on the Session Routing tab, or specify "Passive" as the keyword before the routing rule name with the unityadmin commands.</li> <li>Prioritize the order of systems to determine failover order (* not supported for systems; user must specify all systems by name).</li> </ul>	<ul style="list-style-type: none"> <li>All Reads and Writes go to the system where the session is opened, by-passing the Unity Data Dictionary.</li> <li>No locking/sequencing is performed, and sessions are excluded from Unity's Failover and Recovery.</li> <li>If a request fails and the error code matches a Failover error code, the session is fully closed and moved to the new system automatically. This occurs even when the session was in the middle of an explicit transaction (after Failover, a new explicit transaction is not started and only the query is retried).</li> </ul> <p><b>Note:</b> Unity also allows for manual Failover of a session.</p>

## Adding a Managed Routing Rule

1. In the **Unity Setup** portlet, click the **Session Routing** tab.
2. Click the **Routing Rules** tab.
3. Click .
4. Next to **Routing Rules**, click .
5. In **Rule Name**, type the name for this rule.  
Create a name that identifies the type of routing used for sessions that match this rule. For example, `readonly-rpref` indicates the routing rule is for Read-only Preferred routing.
6. Select **Managed**.
7. For Read-only routing, use the **Read Routing** list to select **Default** or **Preferred** and check two or more Teradata systems under the **Read** column.

Option	Description
Default	Unity selects the system with the shortest work queue and sends the Reads to that system.
Preferred	All Read requests are sent to the first system in the list. If that system is unavailable, the Read session fails over to the next system in the list. There is no load balancing.

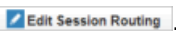

8. To specify where new objects are created, use **Create Routing**.  
All writes are always sent to all systems on which the object exists. If an object exists on a system that is not part of the Write list, an error message appears.

Option	Description
Preferred	New objects are created on the first system in the list. If that system is unavailable, new objects are created on the next system in the list. There is no load balancing.
Balanced	New objects are created on one of the systems specified in the system list. The system is picked at logon and all new objects for a given session are always sent to the same system. Unity balances the systems for each logon session.
All	New objects are created on all systems. Default routing is applied.

9. Select the systems to use to send the reads and writes.  
If a system is selected for **Write**, it must also be selected for **Read**.
10. Using the **Error Profile** list, select the name of the error profile to associate with this routing rule.
11. Click **Save**.  
The new routing rule is added to the summary table.

## Adding a Passive Routing Rule

1. In the **Unity Setup** portlet, click the **Session Routing** tab.

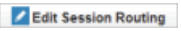
2. Select the **Routing Rules** tab.
3. Click .
4. Next to **Routing Rules**, click .
5. In **Rule Name**, type the name for this rule.  
Create a name that identifies the type of routing used for sessions that match this rule. For example, `passiveSystemX` indicates this rule is for passive routing.
6. In **Routing Type**, select **Passive**.
7. If writes are allowed for objects managed in the Data Dictionary, check the box next to **Allow writes to managed objects**.
8. Select the **Balancing method**.

Balancing Method Name	Description	Example
First Available	From the ordered system list, the first available system is used to establish the session.	Ordered system list: TD1, TD2, TD3 Each session is attempted first on TD1. If the session fails, the session is attempted on TD2 and then TD3, in order.
Least Used	Each session is opened on the system that has the least number of open connections at that time.	Ordered system list: TD2, TD3 Each session is opened on either TD2 or TD3, given whichever has the smaller number of open sessions at the time.
Round Robin	Each session is opened on the next system in the specified order.	Ordered system list: TD1, TD2, TD3 First session is opened on TD1, next on TD2, next on TD3, and the fourth on TD1, and so on.
Weighted Round Robin	Sessions are opened a defined number of times on the first system, then sessions are opened on the next system the defined number of times, and so on.	Ordered system list: TD1x10, TD2x3, TD3x1 The first 10 sessions are opened on TD1, next 3 sessions are opened on TD2, next session on TD3, and then the next 10 sessions are opened on TD1, and so on.

9. Select the systems to include in the routing rule.  
To reorder the systems in the list, click and drag the icon on the left of the system ID.
10. In **Error Profile**, select the associated Error profile.
11. Click **Save**.  
The new routing rule is added to the summary table.

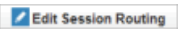

## Editing Routing Rules

1. In the **Unity Setup** portlet, click the **Session Routing** tab.
2. Select the **Routing Rules** tab.

3. Click  .
4. Select the routing rule to edit.  
The Edit Routing Rule view appears.
5. Make changes, and then click **Save**.


## Deleting Routing Rules

You cannot delete the DefaultRouting rule.

1. In the **Unity Setup** portlet, click the **Session Routing** tab.
2. Click the **Routing Rules** tab.
3. Click  .
4. Click  next to the rule you want to delete, and then select **Delete**.
5. Click **Delete** to confirm.

## Assigning Default Routing Rules

The DefaultRouting rule is provided by Unity. You can select it from the **Routing Rule** list when you create a user mapping.

1. In the **Unity Setup** portlet, click the **Session Routing** tab.
2. Click the **User Mappings** tab.
3. Click the user mapping to associate with the DefaultRouting rule.
4. From **Edit User Mapping**, click **Routing rule**  , and select **DefaultRouting**.
5. Click **Save**.

## Using Wildcards in Routing Rules

You can use a wildcard character (\*) to define routing rules when more than one Teradata system is managed by Unity. Use of a wildcard creates a Preferred routing rule because it automatically includes multiple systems and a preferred order for Read/Writes and Failover.

You can use a wildcard character (\*) for the userid and accountstring in user mappings. Unity recognizes four types of userid pattern matching and four types of accountstring pattern matching.

Pattern	Description	Example
Exact match	Exact name match and exact account string match are always first choice in mappings.	patriciadarrell syssd1
Trailing wildcard match	Field must start with the pattern.	pat* syssd*
Leading wildcard match	Field must end with the pattern.	*darrell *sd1



**Note:**

You cannot use a wildcard character (\*) within a string. Use wildcards only at the beginning or end of the string.

When using wildcards, be careful with overlapping wildcards. A `userid` and `accountstring` could match multiple entries in the routing file. Unity uses the user mapping with the lowest ordinal value that matches the `userid` and `accountstring`.

## Example of Routing Rules

All sessions that successfully connect to Unity are mapped to a routing rule by evaluating the user mapping rules. The user mapping rules are specified by `userid`, `region`, `account string`, and `role`. The Routing Rule name refers to a specific rule that defines Read and Write request routing properties such as to which system to send the request if a Preferred routing is specified. Profiles refers to a specific error profile that defines whether to exit or failover when a specific error occurs.

Priority	User ID	Region	Account String	Role	Profile	Routing Rule
1	Susan	RegionA	*	*	*	RoutingA
2	John	*	*	DBA	*	RoutingA
3	*	*	*	DBA	*	RoutingB
4	*	*	CRM	*	*	RoutingC
5	*	*	*	*	*	DefaultRouting

The first rule with Priority 1 says that if Susan logs on from RegionA, Unity performs session routing based on the rules defined in RoutingA. The wildcard in Profile means that all error profiles are used. If John logs on with an assigned DBA role, then Unity performs session routing also using the RoutingA rule. Otherwise, any user who logs on with the DBA role initiates a session that uses RoutingB.

## Examples of Managed Routing Behavior

User mappings and routing rules define how session routing occurs based on its logon properties. This table shows examples of managed routing rules and how different rules determine different session routing strategies.

Routing Rule	Read System	Read	Write System	Create Write	Error Profile
RoutingA	TD1	Preferred			DefaultProfile
RoutingB	TD1, TD2	Preferred	TD1, TD2, TD3	None	DefaultProfile
RoutingC	TD1, TD2	Default	TD1, TD2, TD3	None	DefaultProfile

Routing Rule	Read System	Read	Write System	Create Write	Error Profile
RoutingD	TD3, TD1	Preferred	TD3, TD1	Preferred	DefaultProfile
RoutingE	TD1, TD2, TD3	Default	TD1, TD2, TD3	Balanced	DefaultProfile
DefaultRouting		Default		None	DefaultProfile

RoutingA describes a Read-only rule with Preferred routing to a single system. A session is only created on system TD1. All Read requests are sent to system TD1, and Write requests are rejected. A TD1 system failure will return an error to the client. Any errors classified as RESUBMIT will be returned as errors to the client.

RoutingB describes a Read/Write rule with Read routing to Preferred systems TD1, TD2 using Write systems TD1, TD2, and TD3. Sessions are created on TD1, TD2, and TD3. All Read requests are routed to TD1, and only if needed to TD2. Write requests are sent to TD1, TD2, and TD3 depending on which system the data objects reside.

RoutingC describes a Read/Write rule with Default routing. Routing C is identical to Routing B except that it uses automated routing and distributes Reads to both TD1 and TD2 using the shortest queue algorithm to balance the workload.

RoutingD describes a Read/Write rule with Preferred Read routing to TD3, TD1 using Write systems TD3 and TD1. Sessions are created on TD1 and TD3. All Read requests are routed to TD3 and only if needed to TD1. Write requests are sent to both TD3 and TD1. Write requests that also affect data objects on TD2 are rejected. This rule uses Create Preferred routing which means that CREATE statements are executed preferably on TD3 and use TD1 only if TD3 is unavailable.

RoutingE describes a Read/Write rule with automatic Read routing to TD1, TD2, TD3 using Write systems TD1, TD2, TD3. Sessions are created on TD1, TD2, TD3. Read requests are balanced among the systems that resolve each request. Write requests are sent to all systems that contain the data object requested. This rule uses the CREATE ROUTING - BALANCED routing which means that the CREATE statement is executed on one system for each session created. The system used for the CREATE statement for the session is selected from TD1, TD2, TD3 at the start of the session and will not change unless failover occurs on the preferred system.

DefaultRouting describes a Read/Write rule that includes all systems that are managed by Unity at the time of logon. All systems receive both Read and Write requests. Sessions are created on all systems. Read requests are automatically routed using the shortest queue algorithm to load balance across all systems. Write requests are also sent to all systems. The DefaultRouting routing rule always exists and cannot be deleted.

**Note:**

CREATE statements on sessions that **do not** use the Create Preferred or Create Balanced routing are sent to all systems on the WRITE system list. If both Read Preferred and Create Preferred options are specified, the system lists must be identical because moving or failing over data requires that the new preferred system be able to apply the rules of both the Read Preferred and Create Preferred routing.

## Examples of Passive Routing Behavior

User mappings and routing rules define how session routing occurs based on its logon properties. This table shows examples of passive routing rules and how different balancing methods determine different session routing strategies.

Passive Routing Rule	Access System	Balancing Method	Error Profile
RoutingPassive1	TD1, then if it fails, TD2	First Available	DefaultProfile
RoutingPassive2	TD4, TD3, TD4, TD1 The least used opens first session each time.	Least Used	DefaultProfile
RoutingPassive3	TD1, TD2, TD3 then TD1, TD2, TD3 One session is opened per system in order.	Round Robin	DefaultProfile
RoutingPassive4	TD1x10, TD2x3, TD3x1 then TD1x10, TD2x3, TD3x1 Sessions are opened the defined number of times on the first system, then sessions are opened on the next system the defined number of times, and so on.	Weighted Round Robin	DefaultProfile

## Error Profiles Tab




An error profile is a set of database error codes, which can cause a request to either resubmit or failover to another system, or to terminate the session. When a request returns an error code, Unity checks it against the list of errors in the profile.

The **Error Profile** tab shows the following information for error profiles.

Field	Description
Error Profile	Name create for error profile. DefaultErrorList cannot be renamed.
Error Codes Configured	Number of error codes configured.
Routing Rule Mappings	Number of routing rule mappings.

## Adding an Error Profile

The following steps describe how to create an error profile that you can associate with a routing rule.

1. In the **Unity Setup** portlet, click the **Session Routing** tab.
2. Click the **Error Profiles** tab.
3. Click .
4. Next to **Error Profiles**, click .
5. In **Error Profile**, enter a new name or leave the default name in the box.
6. Next to **Error Codes**, click  add error codes.
7. In **Add Error Code**, complete the following:

Option	Description
Error Code	Error identification code number.
Description	Description displays if the error code matches a Teradata error code. If there is no match, this box is blank.
User Comments	[Optional] Additional information about the error code.

8. Select an the action for Unity to perform if the specified errors occur:



Action	Description
Retry (same system)	For both managed and passive session routing, the Retry Error Action retries the request on the same system where it was originally executed. The query is retried up to three times by default. For managed sessions (sessions associated with a managed routing rule), only Read requests are retried, and only if it is an implicit transaction, or if the Read request is the very first request in an explicit transaction. For passive sessions, all Read/Write requests are retried (if the error code or action matches), and the type of transaction (implicit or explicit) does not affect the decision.
Resubmit (other system)	Resubmit means trying the query on another system. For managed sessions: if an error code indicates resubmit, the query is submitted to another system per the Read routing rule associated with the routing rule. For passive sessions, a resubmit closes the session on the current system, opens the session on the new system (chosen per routing rule), and then the query is attempted on the new system.
Retry then Resubmit	Combination of Retry Error Action and Resubmit.
Exit	Terminates the session.

9. Click **Save**.
10. Verify the error code is added to the **Error Profile** list.  
You can now associate this error profile with a routing rule.

11. Click **Close**.

## Adding Error Codes to Existing Error Profiles

Error profiles consist of a list of error code numbers that you add to the profile. Each error code includes an action (retry, resubmit, retry then resubmit, or exit session) to occur when the request matches the error profile.

1. In the **Unity Setup** portlet, click the **Session Routing** tab.
2. Click the **Error Profiles** tab.
3. Click .
4. Select an error profile.
5. Next to **Error codes**, click .
6. In **Error Code**, type the identification number of the error code or range or error codes you want to add to the profile list.  
If adding a range, separate the start and end error code with a dash. The error codes specified in the range are inclusive. For example, if you enter 3807 - 3200, all error codes from 3807 to 3200, inclusively are added.
7. In **Description**, type a brief statement that describes the error code and selected action.
8. In **User Comments**, type any comment about this error code.
9. Select an action to occur if an error occurs.
10. Click **Save**.
11. Verify the error code is added to the list.

## Error Codes

Number	Message	Summary	Recommendation
1068	Teradata socket read disconnect.	An error occurred while receiving data. Connection to the underlying Teradata instance was lost before all expected data was received.	Resubmit the transaction.
2631	Transaction aborted due to Deadlock.		Resubmit the transaction.
3029	Associated LSN was not found.	Invalid Logical Session Number. An LSN associate connect request was received for a host/LSN which cannot be found in the system.	Check the LSN sent to the DBC to see if it already exists.

Number	Message	Summary	Recommendation
3615	Statement not permitted during Fast Load.	Only BT, CHECKPOINT LOADING, END LOADING, ET, and INSERT are permitted during Fast Load.	Correct the statement and resubmit the request.
3807	Object does not exist.	An object associated with the request was not found.	If the object exists in the database, check to see if the object was deployed in the Unity Dictionary.
3807	Object '%s' does not exist.	Object name provided (%s) but no period at end of message.	If the object exists in the database, check to see if the object was deployed in the Unity Dictionary.
4500	Insufficient memory to complete the request.		Wait and resubmit request.
4501	Invalid request parcel received.		Check all TTU client tools are the correct version.
4502	Internal error: System error occurred. Contact your administrator.		Check Unity error logs for more details.
4503	Internal error: Unexpected response or invalid state encountered. Contact your administrator.	An unsupported message was received from the database.	Check Unity error logs for more details.
4504	Recovery table is not accessible.	During recovery a table or system became inaccessible.	
4505	Connection does not exist.		
4506	Connection name in use or invalid.		
4507	Database system is unavailable.		
4508	The request was canceled by the dispatcher due to a dispatcher reset command.		
4509	Failed to define execution context.		
4510	No systems available.		
4511	Systems to be queried incompatible with current transaction.		
4512	Query not allowed from read-only session.		

Number	Message	Summary	Recommendation
4513	An error has occurred while receiving data. Connection to the underlying Teradata instance was lost before all expected data was received.		
4514	Query is disallowed because it writes to a table containing an identity column.		
4514	Write disallowed because it contains a TOP clause without an ORDER BY clause and is therefore inherently non-deterministic.		
4514	Write disallowed because it contains a SAMPLE clause and is therefore inherently non-deterministic.		
4515	Client-side file access and deferred LOB access is not currently supported by Unity.		
4516	The request could not be processed because the session is interrupted.		
4517	Logon rejected. The default database for this user is not the same across all systems managed by Unity.		
4518	Load feature not enabled.		
4519	An unspecified error occurred while Unity was receiving elicit packets.		
4520	Loading not allowed through read only session.		
4521	Unity table restart state not consistent with DB.		
4522	Load cannot continue, no more space on Unity server.		
4523	Synchronous write returned inconsistent results.		
4524	Specified system is not accessible for the session.		

Number	Message	Summary	Recommendation
4525	Queryband only applicable to create routed session.		
4526	Export feature not enabled.		
4527	Unity heartbeat failed for all managed databases. Check active sequencer.		
4528	Unity failed to communicate with an endpoint to begin load operation.		
4529	Object(s) required for this query are not available.		
4530	Query rejected because associated routing rule prohibits writing to managed objects.		
4531	The request limit for a single transaction has been reached.		
4532	User aborted elicited load.		
4533	Timeout waiting for preceding message. Please resubmit the request.		
4534	Passive Routing feature not enabled.		Make sure Unity is fully upgraded across all servers in the cluster, then enable Passive Routing with the unityadmin command.
4535	The transaction log limit for this session has been reached.		
4536	Number of checkpoints exceeded checkpoints limit setting.		
4537	SQL request is not supported by Unity.		
4538	Invalid request causing lock queue out of sequence.	This is a self-correcting error. The affected object should be interrupted and get autorecovered.	
4539	Failed to modify object(s) in dictionary.		
4540	Query disallowed because it writes to a mix of objects that		




Number	Message	Summary	Recommendation
	are managed and unmanaged by Unity.		
4541	Write to Unity managed objects disallowed from unmanaged stored procedure. Please issue an object scan to update the Unity dictionary.		
7423	Object already locked and NOWAIT. Transaction Aborted.	Request object already locked and NOWAIT option specified. Transaction are rolled back.	Resubmit the transaction.
8017	The UserId, Password, or Account is invalid.		
8018	The session id is illegal.		
8019	Server instance shutdown or failure.	The parcel stream following the message header on the assign, reassign, authorization, single sign-on, connect, or reconnect request is not valid either because the parcel(s) sizes or parcel(s) types are wrong.	Repeat the assign sequence with a good parcel stream.
8019	Internal server error (invalid sequencer response).	The parcel stream following the message header on the assign, reassign, authorization, single sign-on, connect, or reconnect request is not valid either because the parcel(s) sizes or the parcel(s) types are wrong.	Repeat the assign sequence with a good parcel stream.
8019	Invalid query text.	A PCLREQ packet was received with a negative query text length.	Repeat the assign sequence with a good parcel stream.
8019	Failed to prepare dictionary for dynamic DDL.	The parcel stream following the message header on the assign, reassign, authorization, single sign-on, connect, or reconnect request is not valid either because the parcel(s) sizes or the parcel(s) types are wrong.	Repeat the assign sequence with a good parcel stream.

Number	Message	Summary	Recommendation
8019	Disallowed connection type.	The connection was rejected because it came from an unsupported client tool (for example, FastLoad Stand Alone).	Repeat the assign sequence with a good parcel stream.
8026	The LAN message RequestNo field is invalid.	This error is generated when a start request contains a request number lower than that of any previous start request or when a continue request contains a request number higher than any previous start request.	Correct the request number and retry the continue or start request.
8051	Failed to run statement due to dictionary modification failure.		
8051	Failed to drop object(s) due to dictionary modification failure.		
8051	Failed to modify object(s) in dictionary.		
8051	Internal memory failure.		
8917	Session targeted Teradata system(s) are unavailable.	Same error as 4513, but used for read-override (Query Director Like) mode.	
9999	Write to objects on non-readable systems.		
9999	Write to objects on systems unavailable in session.		



## Editing an Error Profile

Unity locks the **Error Profile** view while you edit. You can add and remove error codes and change the actions associated with those errors.

1. In the **Unity Setup** portlet, click the **Session Routing** tab.
2. Click the **Error Profiles** tab.
3. Click  .
4. Select a profile name to edit.
5. From **Edit Error Profile**, make edits.
6. Click **Close**.



## Deleting an Error Profile

Review error profiles to ensure they are up to date and delete those that are no longer relevant.

1. In the **Unity Setup** portlet, click the **Session Routing** tab.
2. Click the **Error Profiles** tab.
3. Click .
4. Next to the profile name you want to delete, click  and select **Delete**.
5. Click **Delete** to confirm.  
The deleted profile no longer appears in the error profile list.

## Assigning the Default Error Profile

The default error profile is named DefaultErrorList. You can assign the default profile to a routing rule when you create the rule.

1. In the **Unity Setup** portlet, click the **Session Routing** tab.
2. Click the **Routing Rules** tab.
3. Click .
4. Select a routing rule.
5. From **Edit Routing Rule**, click the **Error profile** , and select **DefaultErrorList**.
6. Click **Save**.

## Time Windows Tab

Time Windows allows users to specify automated start and stop times for the following:

- Regular updates
- Commonly performed, regular tasks
- Planned outages
- User Mapping
- Routing Rules


---

### Note:

Time Windows can start a script when opening (active) or closing (inactive).

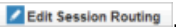
---

## Setting Up Time Windows

1. Open Unity Setup, select **Session Routing**, and select **Time Windows**.
2. Select .
3. Fill out the **Add Time Windows** card and select **Save**.

## Deploying Session Routing

Deploying session routing makes the edited rules active for session management.

1. In the **Unity Setup** portlet, click the **Session Routing** tab.
2. Click on a tab (User Mappings, Routing Rules, Error Profile).
3. Click .
4. Click **Deploy**, and then click **Deploy** to confirm.  
This deploys all the User Mappings, Routing Rules, and Error Profiles at the same time.
5. Click **OK**.

## Importing a Session Routing File

1. In the **Unity Setup** portlet, click the **Session Routing** tab.
2. Click **Import**.
3. Click **Browse** to specify the file to import and then click **Open**.
4. Click **Import**.

## Exporting a Session Routing File

You can export the routing data to a text (.txt) file. You can edit the file and import it back to Unity to update session routing.

1. In the **Unity Setup** portlet, click the **Session Routing** tab.

2. Click **Export**.

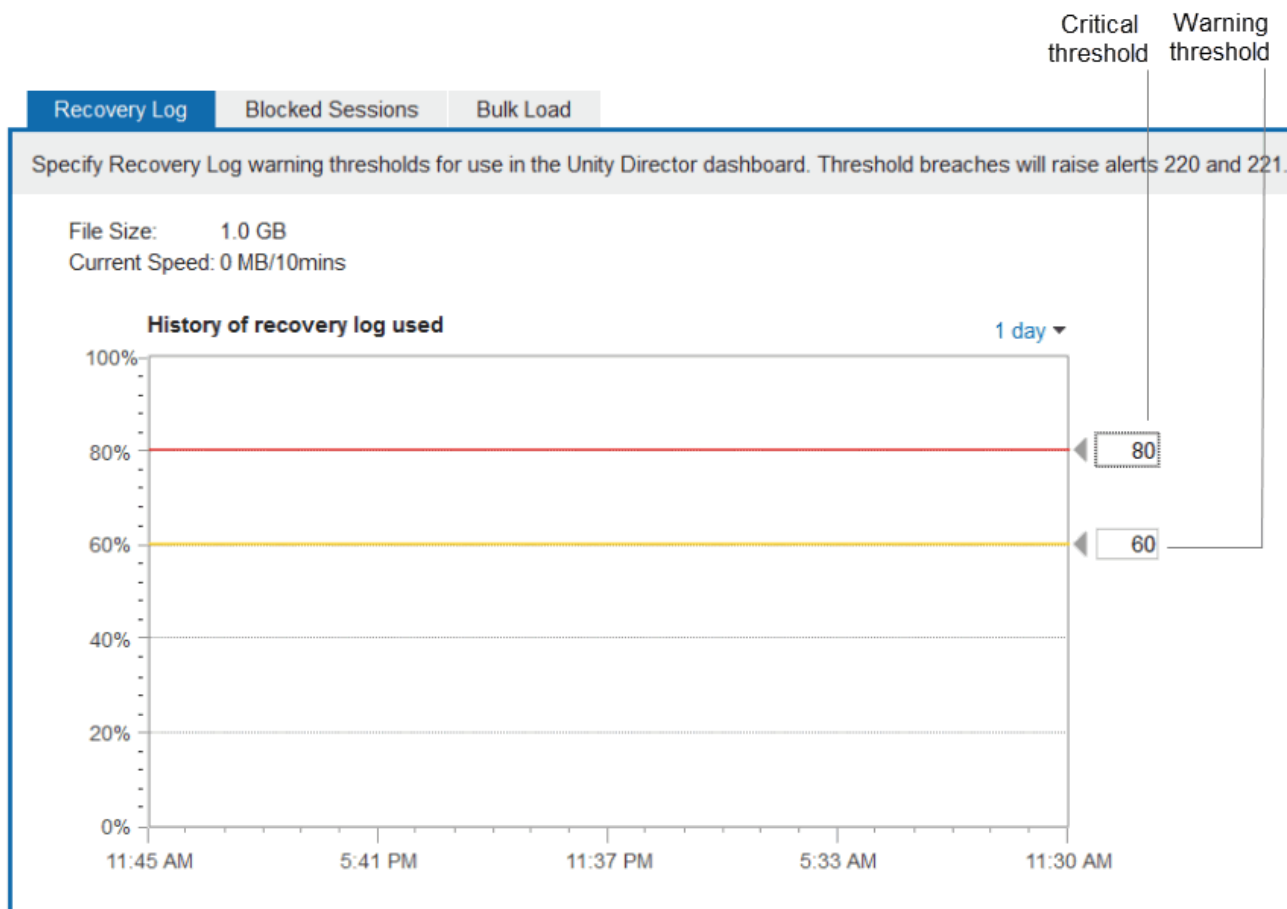
The export file is named sessionRouting.txt.

## Thresholds Tab

The **Threshold** tab contains views for managing Recovery Log, blocked session, and bulk load thresholds. You can set maximum thresholds for recovery log space usage, number of blocked sessions, and disk usage during bulk load operations. If a value exceeds a threshold, an alert is generated.

## Recovery Log Tab

Unity monitors how much disk space the Recovery Log file uses. The **Recovery Log** tab under the **Thresholds** tab lets you define the warning and critical thresholds for the Recovery Log. The thresholds set the maximum percentage of allotted space allowed before a warning or critical alert is raised.



## Modifying Recovery Log Thresholds

1. In the **Unity Setup** portlet, click the **Thresholds** tab.

- Click the **Recovery Log** tab.
- Modify the threshold settings in the graph.

Option	Description
Set the critical threshold	Locate the ▲ icon next to the critical line. Click and hold the ▲ icon and slide it up (increases threshold) or slide it down (decreases threshold). You can also enter the percentage.
Set the warning threshold	Locate the ▲ icon next to the warning line. Click and hold the ▲ icon and slide it up (increases threshold) or slide it down (decreases threshold). You can also enter the percentage.

- Click **Apply**.

## Blocked Sessions Tab



When an active session locks data or hangs, it can block other sessions from accessing data and completing transactions. Session deadlock can occur if the number of blocked sessions exceeds normal performance thresholds.

The **Blocked Sessions** tab lets you define the warning and critical thresholds for blocked sessions. A blocked session threshold sets the maximum number of blocked sessions allowed before a warning or critical alert is raised.

## Modifying Blocked Session Thresholds

To help monitor blocked sessions, set warning and critical thresholds to indicate how many blocked sessions are in the Normal or Warning range.

- In the **Unity Setup** portlet, click the **Thresholds** tab.
- Click the **Blocked Sessions** tab.
- Modify the thresholds for one or more blocked sessions in the graphs.

Option	Description
Set critical threshold	Do one of the following: <ul style="list-style-type: none"> <li>Locate the ▲ icon that marks the beginning of the critical  section of the column. Click and hold the ▲ icon to move it up (increases threshold) or move it down (decreases threshold).</li> <li>Enter a value into the number indicator next to the slider.</li> </ul>
Set warning threshold	Do one of the following: <ul style="list-style-type: none"> <li>Locate the ▲ icon that marks the beginning of the warning  section of the column. Click and hold the ▲ icon to move it up (increases threshold) or move it down (decreases threshold).</li> <li>Enter a value into the number indicator next to the slider.</li> </ul>

- Click **Apply**.

## Bulk Load Tab

Bulk Load is available through Unity. The **Bulk Load** tab lets you define the percentage of space used before a warning or critical alert is raised.

### Modifying Bulk Load Thresholds

1. In the **Unity Setup** portlet, click the **Thresholds** tab.
2. Click the **Bulk Load** tab.
3. Modify the threshold settings in the graph.

Option	Description
Set the critical alert threshold	Locate the ▲ icon next to the critical line. Click and hold the ▲ icon and move it up (increases threshold) or move it down (decreases threshold). You can also enter the percentage.
Set the warning alert threshold	Locate the ▲ icon next to the warning line. Click and hold the ▲ icon and move it up (increases threshold) or move it down (decreases threshold). You can also enter the percentage.

4. Click **Apply** for the changes to take effect.

## Alerts Tab

You can select which alerts to enable for use in Unity.

### Editing Alerts

1. In the **Unity Setup** portlet, click the **Alerts** tab.
2. Click the alert.
3. Select the **Enable alert** check box to allow Unity to monitor this alert.
4. Select a severity from the **Severity** list.
5. Click the default recommended action or add a custom action.
6. Click **Save**.

## Components Tab

The **Components** tab lets you edit the layout of the Dashboard view. The **Configuration Information** shows details for Dispatchers, Sequencers, and Endpoints.

### Setting Up the Dashboard Layout

1. In the **Unity Setup** portlet, click the **Components** tab.

2. Drag and drop the regions or Teradata systems to reorder.
3. Click **Apply**.

## Viewing Component Configuration Settings

1. In the **Unity Setup** portlet, click the **Components** tab.
2. Click **Dispatchers**, **Sequencers**, or **Endpoints**.  
You can view different regions. The configuration parameter values for the Sequencers and Dispatchers are listed. To change a parameter value, see [CONFIG UPDATE](#).



# Viewpoint Unity Portlet

The Unity portlet lets you monitor components, table health, alerts, sessions, operation details, interrupted sessions, and bulk load jobs.

---

**Note:**

Not all features are available with the Unity Passive Routing license.

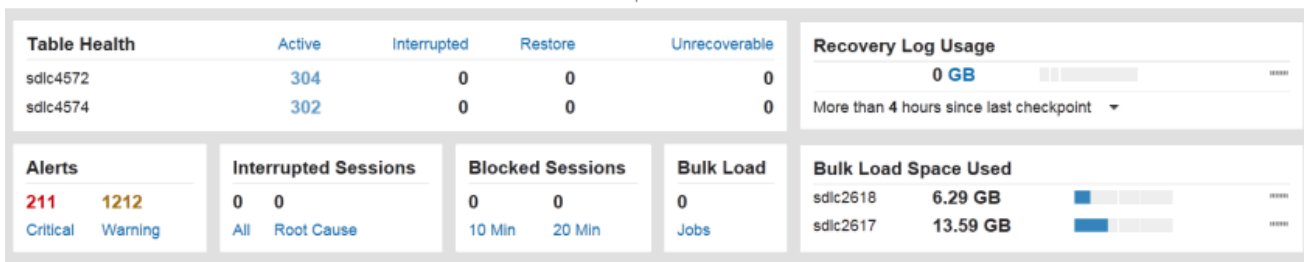
---

## Dashboard Tab

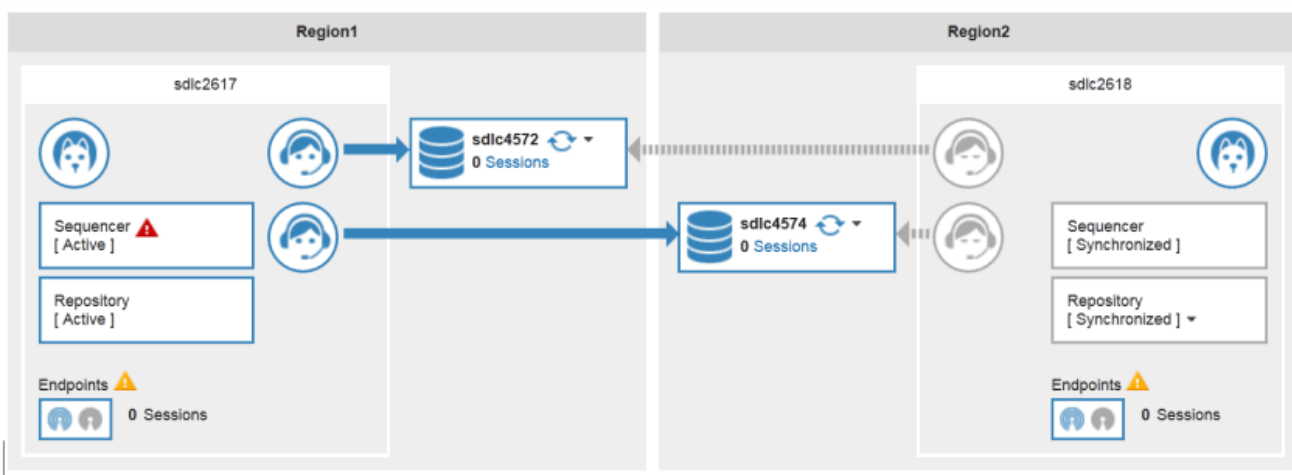
The **Dashboard** tab provides high-level information on the following Unity features:

- **Table Health** shows the status of synchronized data across Teradata Database systems.
- **Alerts** shows the number of critical and warning alerts.
- **Interrupted Sessions** shows the number of interrupted sessions and number of sessions that are the root cause.
- **Blocked Sessions** shows the number of blocked sessions based on durations and alert threshold states.
- **Bulk Load** shows the number of bulk load jobs running.
- **Bulk Load Space Used** shows the amount of space currently used for bulk load jobs.
- **Recovery Log Usage** shows the amount of recovery log space used and the time since the last checkpoint.
- **Components** shows the processes configured for Unity.

## Dashboard




## Components



## Components pane

## Viewing Dashboard Properties

1. In the **Unity** portlet, click the **Dashboard** tab.
2. In **Table Health**, click a status to view tables with that status.
3. In **Recovery Log Usage**, click the size of the Recovery Log to view the Recovery Log Occupancy Per Table.
4. In **Alerts**, click the warning level to view alerts with that level.
5. In **Interrupted Sessions**, click **All** or **Root Cause** to view the interrupted sessions.
6. In **Blocked Sessions**, click a time limit to view blocked sessions for that time.
7. In **Bulk Load**, click **Jobs** to view current bulk load jobs.
8. To view an alert for a component, click .

## Recovery Log Tasks

From the **Unity Setup** portlet, you can view and control the Recovery Log's checkpoints. Using the unityadmin commands, you can also change the Recovery Log size and the Automatic Recovery interval.

## Viewing Recovery Log Space Usage

1. In the **Unity** portlet, click the **Dashboard** tab.
2. Review the **Recovery Log Usage** graph.  
The total hours since the last checkpoint appear.
3. [Optional] To access the **Recovery log occupancy per table** view, click the amount listed next to the graph.

## Changing the Recovery Log Size

When you change the recovery log size, you must delete the recovery log file. Changing the recovery log size removes the Unity object state information. After the sequencers are started, all systems exist in the unrecoverable state. To promote systems to an active state, see [Activating a Managed System](#).

---

### Important:

All objects managed by Unity remain in an unsynced state after this process is completed.

---

1. From unityadmin, change the **RecoveryLogSize** parameter value to the new size in gigabytes:  
`CONFIG UPDATE SEQUENCER RecoveryLogSize <newSize> REASON 'Increasing recovery log size';`
  2. Retrieve the recovery log file name and path.  
The default location is `/data/recovery.log`. The recovery log path is stored in the configuration variable, *RecoveryLogPath*, and the filename is stored in the configuration key *RecoveryLogName*.
- 

### Note:

Run the unityadmin commands `config list key RecoveryLogPath` and `config list key RecoveryLogName` to query the current setting for these parameters.

---

3. Run the following commands to resize the recovery log.
- 

### Note:

Resizing the recovery log involves the deletion of the current recovery log file. Make sure all systems and objects are fully recovered and in the active state before running the following commands.


---

- a. Stop both sequencer processes:  
`/etc/init.d/unity stop`
- b. Remove the old recovery log file on both Unity servers:  
`rm /data/recovery.log`
- c. Restart the sequencer processes on both Unity servers:

```
/etc/init.d/unity start
```

4. Reactivate the cluster.

## Forcing a Recovery Log Checkpoint

1. On the **Unity** portlet, click the **Dashboard** tab.
2. Next to **More than {0} hours since last checkpoint**, click  and then select **Force Checkpoint**.

## Setting the Automatic Recovery Interval

1. From unityadmin, change the time between automatic recovery attempts:  

```
config update sequencer recoveryinterval <recoverytime> REASON 'Increasing recovery interval';
```

 To disable automatic recovery, set the interval to 0. The default time is 300 seconds (5 minutes).

## Recovery Log Settings

### RecoveryLogSyncMode

The **RecoveryLogSyncMode** parameter controls how the Recovery Log is synchronized across Unity servers. The parameter default value is `async`. You can change the parameter to: `near-sync` or `sync`.

For `near-sync`, the Active Sequencer waits for all entries to be written to the Recovery Log of the Standby Sequencer before it sends the writes to the Teradata Database system. Performance impact for this operation depends on the network latency between the two Unity servers. In this mode, the operating system periodically writes to the Recovery Log cache on the Standby Sequencer, and writes to the disk. If both Unity servers fail at the same time, the Recovery Log on the Standby Sequencer might be missing entries, because the writes were not flushed to the disk.

For `sync`, the Standby Sequencer pushes all writes to the disk before responding to the Active Sequencer. This mode has the most data protection and the largest performance impact.

For `near-sync` or `sync`, the **RecoveryLogSyncTimeout** parameter specifies the maximum amount of time the Active Sequencer waits for a response from the Standby Sequencer before continuing. If the Active Sequencer does not receive a response from the Standby Sequencer, it continues sending requests to the Teradata Database systems. If the **RecoveryLogSyncTimeout** is set to 0, then the Active Sequencer waits indefinitely for a response. Setting this parameter to 0 is not recommended.

### RecoveryLogFullyCreate

Unity creates the Recovery Log when the Sequencer process starts. The **RecoveryLogFullyCreate** parameter determines the space allocation for the Recovery Log file on startup. The parameter can be set to one of the following values: `true` or `false`.

If `true`, this parameter does not result in a sparse file; space is allocated on initial startup. Startup time for the Sequencer increases the first time the Sequencer is started; startup time is not impacted for the subsequent

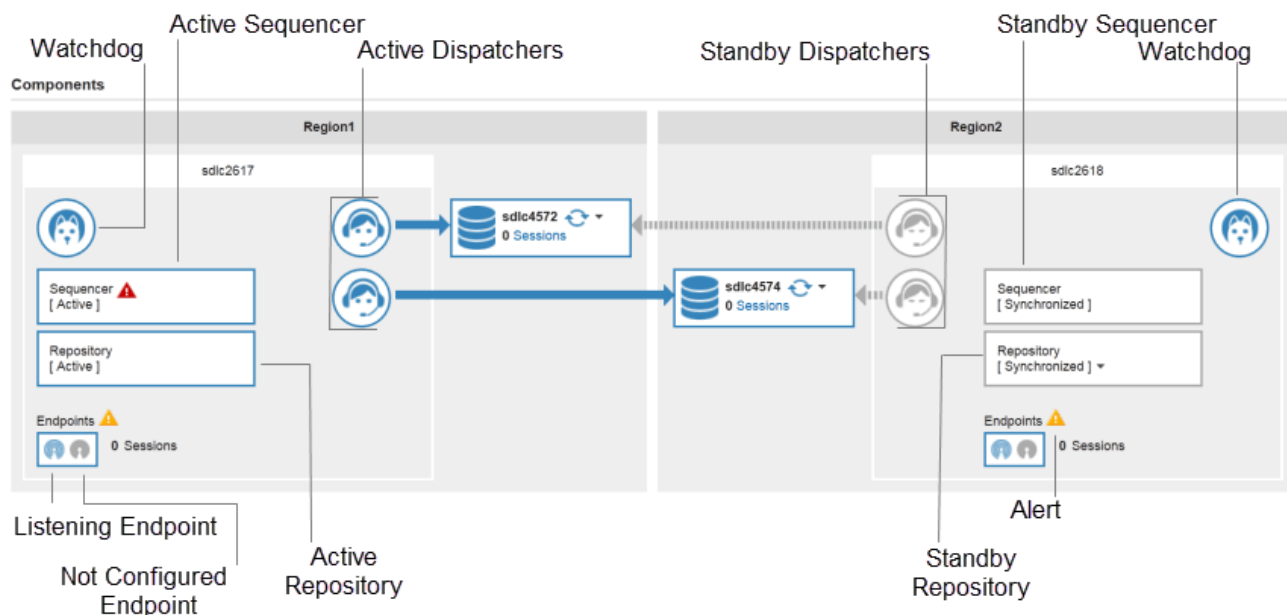
restarts of the Sequencer. This also guarantees that there is sufficient space available for the Recovery Log and Writes will not fail due to out-of-disk errors. Use to ensure sufficient space for the Recovery Log.

If false, this parameter results in a sparse file; space is allocated for the Recovery Log as it receives data Writes. Use when there is sufficient disk space for the Recovery Log and another process is not using the disk space to enable faster process startup. This does not impact the Sequencer process startup time. Unity only detects insufficient disk space for the entire Recovery Log during operation, resulting in a fatal error.

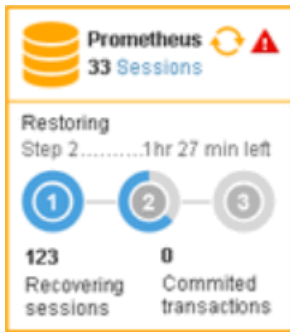
## Components Pane

The **Components** pane shows the configuration of a Teradata system managed by Unity. The layout of the components shows whether there is a single or dual-system configuration. A dual-system configuration is a high availability (HA) system with fail-over capability to a Standby system. You can change the state of components in your ecosystem.

The Components pane shows two regions for HA systems. Each region shows the number of sessions, type of processes installed, name of the processes, state of processes, and any alerts. For example, Region 1 shows an Active Sequencer with an alert.



If a Recovery is in process, the status appears in the Components pane. The Recovery process includes three steps. This example shows the second step.



## System Component Properties

The following table describes the potential states for each component process in Unity.

Component	States
Endpoints	<ul style="list-style-type: none"> <li>• Not configured</li> <li>• Configured, but not listening</li> <li>• Listening to one host group</li> <li>• Listening to all</li> <li>• Session connected to the Endpoint</li> </ul>
Dispatcher	<ul style="list-style-type: none"> <li>• Active</li> <li>• Standby</li> </ul>
Sequencer	<ul style="list-style-type: none"> <li>• Active</li> <li>• [Synchronized] Standby - Recovery log is in sync with the Active Sequencer</li> <li>• [Synchronizing] Standby - Sequencer is receiving Recovery Log information from the Active Sequencer</li> <li>• [Unsynchronized] Standby - Recovery Log is out of sync and cannot contact the Active Sequencer</li> <li>• [Waiting] Waiting to communicate with other Sequencer to determine which is Active</li> <li>• [Disabled] - Cannot determine which Sequencer should be Active.</li> <li>• Down</li> </ul>
Watchdog	<ul style="list-style-type: none"> <li>• Up</li> <li>• Down</li> </ul>

## Paired High Availability Status

The state of individual Sequencer processes on each paired Unity server determine HA status.

This table describes the states that indicate status of paired HA servers.

Status	Description
Not Enabled	Indicates the Unity servers are not paired for high availability.
In Peer	Indicates one of the Sequencer processes is in the <i>Active</i> state, and the other is in the <i>Standby Synchronized</i> state.
Not in Peer	Indicates the Unity servers are paired, but the Sequencer process cannot perform an automatic failover.
Establishing Peer	Indicates the Unity servers are paired, but the Sequencer process cannot perform an automatic failover.
Split Brain	Indicates the Unity servers are paired, but the Sequencer process cannot perform an automatic failover.

You must intervene if the Sequencer process cannot perform an automatic failover.

This table describes the relationship between the states of both Sequencer processes and the resulting paired HA status.

Sequencer State	Combination	Paired HA Status
Down	Waiting	Not in Peer: Waiting for Sequencer startup
Waiting	Waiting	Establishing Peer: Waiting for Sequencers
Active	Standby-Synchronizing	Establishing Peer: Synchronizing
Active	Standby	In Peer
Active	Standby-Unsynchronized	Not in Peer: Cannot fail over
Active	Down	Not in Peer: Cannot fail over
Down	Standby-Unsynchronized	Not in Peer: Cannot fail over
Active	Active	Split Brain
Disabled	Disabled	Split Brain

## Activating a Managed System

After a Teradata system is installed, all systems managed by Unity are placed in the Unrecoverable state. To synchronize data and bring all components in a managed system into the Active state, an administrator must activate the Teradata system. This operation is required for both high-availability (HA) and single system (non-HA) configurations.

Perform the following steps for each managed system.

1. In the **Unity** portlet, click the **Dashboard** tab.
2. Verify that components for the intended Active system are installed and configured properly.

The **Components** pane shows all installed components and their current state.

- Click ☐ next to the system and select **Active**.  
All components in the managed system promote to the Active state.

## Changing a System State

- In the **Unity** portlet, click the **Dashboard** tab.
- In the **Components** pane, click ☐ located beside the state icon on the component.
- Select a target state from the list.
- Verify the target state, **Component ID**, and **Component Type** fields.
- [Optional] Enter a reason for the state change.
- Click **Change State**.

## Responding to an Interrupted Table

If a table or Teradata system becomes *Interrupted*, an alert is generated. The alert contains the reason for the interruption, the response from the system where the request was a success, and the response from the system where the failure occurred.

Check the database request in the alert and the Teradata system to find the cause of the failure. Once you fix the root cause, Unity begins recovery of the Interrupted Table. If you need to trigger a recovery, follow these steps:

- In the **Unity** portlet, click the **Dashboard** tab.
- In the Components pane, click ☐ next to the interrupted system, and select **Recover**.

## Table Health Tab

The **Table Health** tab lists all database tables managed by Unity and shows their current state.









### Note:

This feature is not available with the Unity Passive Routing license.

Field	Description
Database Name	Name of associated database.
Table	Name of table.
Table ID	Generated identification.
Systems	Name of systems where table resides.



## State Icons

Icon	State	Description
	Active	A table or system is available and enabled for Read/Write operations. Data synchronization is enabled on Active systems.
	Standby	A database system or table is up-to-date and available for a failover.
	Interrupted	An automatically recoverable transient error has occurred. If a Teradata system is interrupted, Read/Write requests are routed to a Standby Teradata system until the original system recovers from the error. All tables on an interrupted Teradata system are also placed in the Interrupted state. If a table is in an Interrupted state, the table cannot be accessed.
	Read-only	An application has only Read access to a database system or table.
	Restore	A database system or table is being recovered. Unity applies any missed Write operations from the Recovery Log.
	OOS	A system is out of service and unavailable. There is no monitoring of the database system or tables, and no alerts. All Unity sessions connected to the Out of Service (OOS) system are closed.
	Unrecoverable	A database system or table is unavailable, and Unity does not recognize the state of the data. If the Teradata Database system is in an Unrecoverable state, access to its tables is not allowed.
	Pending Writes	The number of writes waiting.

## State Filter Bar

The **Table Health** state filter bar displays the number of tables currently in each table state.



This example shows 406 Active tables. To filter tables by state, select one of the states.

## Changing Table Placement and Removing Tables

1. In the **Unity Director** portlet, to the left of the target table, click the down arrow.

To remove a table	<ol style="list-style-type: none"> <li>a. Select <b>Remove</b>.</li> <li>b. In the <b>Remove Table</b> confirmation window, select <b>Remove</b>.</li> </ol>
To change which systems a table is managed on	<ol style="list-style-type: none"> <li>a. Select <b>Manage</b>.</li> <li>b. In the <b>Manage Table</b> window, select or deselect the systems the table is managed on.</li> </ol>

c. Click <b>Manage</b> .
--------------------------

## Changing States in Table Health Tab




You cannot change all table states manually. Unity displays only supported state transitions in the **Change State to** list. For supported transitions, see [Supported State Transitions](#).













1. In the **Unity** portlet, click the **Table Health** tab.
2. Select a method and follow the steps to change the table state.  
All tables that you select at the same time must be able to transition to the same state. If you select multiple tables that cannot transition to the same state, the method is not available.

Method	Steps
Select tables by state	<ol style="list-style-type: none"> <li>a. Use the state filter bar to select one or more states.</li> <li>b. In the <b>Change state of filtered to</b> list, select a new state.</li> <li>c. Select the systems for the change.</li> <li>d. [Optional] Enter the reason for the change.</li> <li>e. Click <b>Change State</b>.</li> </ol>
Select tables by managed Teradata systems	<ol style="list-style-type: none"> <li>a. Select the system.</li> <li>b. [Optional] Deselect any tables that you do not want to change.</li> <li>c. In the <b>Change state of selected to</b> list, select the new state.</li> <li>d. [Optional] Enter the reason for the change.</li> <li>e. Click <b>Change State</b>.</li> </ol>
Select individual tables	<ol style="list-style-type: none"> <li>a. Select the box next to the table information.</li> <li>b. [Optional] Deselect any systems that you do not want to change.</li> <li>c. In the <b>Change state of selected to</b> list, select the new state.</li> <li>d. [Optional] Enter the reason for the change.</li> <li>e. Click <b>Change State</b>.</li> </ol>

## Supported State Transitions

This table describes Unity system and table states and shows supported transition states.

State	Current State Description	Supported State Transitions
 Active	Table is available for Read /Write operations. Data synchronization is enabled on Active systems.	<ul style="list-style-type: none"> <li>•  OOS (Out of Service): This allows you to halt table synchronization.</li> <li>•  Read-only: This allows you to freeze the table. A table may also automatically be placed in Read-only if a session encounters an error and the <b>SetReadOnlyOnWriteFailure</b> parameter is set to true.</li> </ul>

State	Current State Description	Supported State Transitions
 Standby	Database table is up-to-date and available for failover Read/Write operations.	<ul style="list-style-type: none"> <li>•  Active</li> </ul>
 Interrupted	An error occurred on a table for one system. This error was not duplicated for the same table on other systems. This error is potentially recoverable. Until the system recovers from the interruption, tables in this state are unavailable. For example, when a session cannot be opened on a Teradata Database system due to insufficient Teradata Database system resources, the state changes to <i>Interrupted</i> .	<ul style="list-style-type: none"> <li>•  Restore: Table normally recovers automatically. Investigate the problem that caused the interruption before attempting to restore to an Active state. If the problem is not resolved, the interruption may reoccur. After the problem is resolved, to force recovery, place the table in the Restore state. This synchronizes data and sets the table state to the Active state.</li> </ul>
 Read-only	In a Read-Only state, only Read access is allowed to the Teradata Database system or table.	<ul style="list-style-type: none"> <li>•  Restore: To change the table to Read/Write, use the Restore state. This synchronizes data and sets the table state to the Active state.</li> </ul>
 Restore	Table is in active recovery operation. Unity applies any missed Write operations from the Recovery Log.	<ul style="list-style-type: none"> <li>•  Active: After restore completes, verify the table was successfully recovered and placed in the Active state.</li> </ul>
 OOS	No Read or Write requests are being sent to this table in the Out of Service state.	<ul style="list-style-type: none"> <li>•  Restore: To recover from this state, place the table in the Restore state. This synchronizes data and sets the table state to the Active state.</li> </ul>
 Unrecoverable	System or table is unavailable. Because Unity does not recognize the state of the data, it does not automatically recover a table in the Unrecoverable state.	<ul style="list-style-type: none"> <li>•  Active: For database tables: Tables in the Unrecoverable state must be synchronized manually. You cannot restore a table if it is unrecoverable on one system and active on another. You must mark the active table as unrecoverable and activate the table across all systems. For Teradata systems: During configuration, a Teradata system are first placed in an Unrecoverable state. If all system are Unrecoverable, you must manually synchronize the data across all your Teradata systems. You can activate all systems in the <b>Components</b> pane.</li> </ul>

## System and Table States for Managed Sessions

Unity tracks the state of tables and systems for managed sessions.

## Active State

When a Teradata Database or table is in an Active state, it allows read and write requests. Tables must be in an Active state on at least one system to enable clients to issue write requests. Even if a table is Active, it can have a pending Write request queue.

### Related Information:

[Unityadmin Command Line Guidelines](#)

## Standby State

When a Teradata system or table is in a Standby state, it only receives writes to keep current as long as there is another active copy. No reads are sent to a Teradata Database system or table in Standby unless no active Teradata system or table is available.

When demoting a system or table to a Standby state, all in-flight reads are complete and any new reads are rerouted to another Teradata system.

When a Teradata Database system is in Standby state, all tables on that system become Standby.

### Related Information:

[Unityadmin Command Line Guidelines](#)

## Read-Only State

This state enables read access for an application when data is externally copied from the table and must remain unchanged.

For managed sessions, if a table is in a read-only state across all Teradata Database systems, Unity rejects the write with an error. If only a single Teradata Database system or a table on a specific Teradata Database system is in read-only state, then any write requests remain queued in the Recovery Log to be replayed when the Teradata Database system or table is placed into a restore state.

If a Teradata Database system enters a read-only state, all tables automatically enter a read-only state. If a table is read-only on one or more systems and a write occurs on this table (because the table is in read-write mode on at least one other active system), then the read-only state is changed to an interrupted state.

A table may also automatically be placed in read-only if a session encounters an error and the "Read-only on write failure" feature is turned on.

To change a table or system to a read-only state, run the `SYSTEM FREEZE` command. Unity holds any new transactions and waits for any in-flight transactions to close. The Dispatcher holds transactions for the time specified in the **HaltTimeout** parameter. If successful, the Teradata Database system or table becomes read-only. If an error occurs or if in-flight transactions do not finish successfully during the timeout, the operation aborts and the Teradata Database system or table remains in the active state.

From the read-only state, you can change a Teradata Database system or table into a restore state with the `SYSTEM RECOVER` command.

**Related Information:**

[unityadmin Command Line Guidelines](#)

[SYSTEM FREEZE](#)

[SYSTEM RECOVER](#)

## Interrupted State

If a Teradata Database system is Interrupted, it does not allow sessions to start. All tables on that system are automatically placed in Interrupted state when a Teradata system is placed in Interrupted state.

From Interrupted state, a Teradata system or table can automatically enter a Restore state. Unity periodically attempts to recover a Teradata system or table in the Interrupted state. It can also enter a Restore state if the dispatchers for that system are restarted. To change an Interrupted state to the Restore state, use `SYSTEM RECOVER` command.

To change to an Unrecoverable state, use `SYSTEM DEACTIVATE` command.

**Related Information:**

[Unityadmin Command Line Guidelines](#)

[SYSTEM RECOVER](#)

[SYSTEM DEACTIVATE](#)

## Restore State

When in Restore state, the Teradata Database system or table does not receive new reads or writes. During the recovery operation, Unity queued writes are processed for this Teradata Database system and new writes are added at the end of the recovery queue.

A Teradata Database system or table can enter a Restore state in the following situations:

- During automatic recovery.
- After a Dispatcher startup for the associated Teradata system.
- From an Active state, if the number of writes pending for the Teradata Database system is greater than the **RecoveryQueueLimit** parameter.

From Restore state, a Teradata Database system or table can enter the following states automatically:

- Active state if the recovery process is successful.
- Interrupted or Unrecoverable state if an error occurs during recovery.

To change the state from Out of Service or Interrupted state, use the `SYSTEM RECOVER` command.

**Related Information:**

[Unityadmin Command Line Guidelines](#)  
[SYSTEM RECOVER](#)

## Unrecoverable State

Unity does not allow sessions on an Unrecoverable Teradata Database system. All tables automatically become Unrecoverable when a Teradata Database system is placed in the Unrecoverable state.

A Teradata Database system or table automatically becomes Unrecoverable when the following occurs:

- Adding a new Teradata Database system to Unity.
- A Teradata Database system in Active, Standby, or Restore state encounters a data inconsistency during a write request.
- Tables added to Unity through a dictionary deploy also come up as Unrecoverable. The only exception is when the table exists only on one system and that system is in the Active state. In this case, the table is auto-activated once the dictionary deploy finishes.

**Note:**

Unity automatically changes the state of tables based on a recovery mechanism if tables are in different states.

For example, you have Unity with two systems, both in an Active state. When a transaction runs, it completes successfully on a system where both tables X and Y are in an Active state. However, on a system where table Y is in an Unrecoverable state, table X automatically changes to an Unrecoverable state.

To change a table or Teradata system to Active from Unrecoverable, use `SYSTEM ACTIVATE` command.

**Related Information:**

[Unityadmin Command Line Guidelines](#)  
[SYSTEM ACTIVATE](#)

## Read and Write Requests for Managed Sessions

The following table shows when SQL statements are allowed depending on the Teradata Database or table state.

Teradata Database or Table State	Read Requests	Write Requests
Active	Yes	Yes
Read-Only	Yes	No

Teradata Database or Table State	Read Requests	Write Requests
Standby	No (as long as there is another Active copy of the table or system)	Yes
Out-of-Service	No	No
Disconnected	No	No
Interrupted	No	No
Restore	No	No
Unrecoverable	No	No

**Note:**

Write requests are rejected if there are no systems where the table is in an Active or Standby state.

## Error Handling for Reads for Managed Sessions

If Unity encounters an error during a Read request, it checks for the error code and responds as follows.

Error is	Response
Listed in the Error Profile and set to perform a Failover	Resubmits the query to another Teradata Database system. Unity selects the Teradata Database System with the shortest working queue at that time.
Listed in the Error Profile and set to Retry	Resubmits the query to the same Teradata Database system.
Listed in the Error Profile and set to Exit	Session closes on all Teradata Database systems.
Not listed in the Error Profile	Error is returned to the client.

## Error Handling for Writes for Managed Sessions

For Write operations, Unity does not wait for all responses from each Teradata Database system. Unity checks the first response for successful operation or error code. Errors include communication failures, memory errors, disk space, and database errors.

System Response	Type	Description
First Response	Successful operation	Unity notifies the client application.
	Error code	All Active Teradata Database systems must return a response before the client application receives one. The Active Sequencer

System Response	Type	Description
		uses a voting mechanism to determine which response is the most common successful response, and sends that response back to the client application. If no successful response is available, the most common error response returns to Unity. If a responses are tied, the first response that Unity receives returns to the client application.
Return to Client	Error code listed in the Error Profile for exit	Unity returns the error and exits the session on all Teradata Database systems.
	Successful operation	Unity compares results of the remaining Teradata Database systems to determine consistency of data across Teradata Database systems. The response sent back to the client is final, so all responses from the remaining Teradata Database systems must match.

Unity uses the activity count (number of rows) to indicate data consistency for all INSERT, DELETE, UPDATE, and MERGE statements. The activity count from all Teradata Database systems must match with the activity count of the response sent back to the client. The activity count is not checked for any other types of Write operations. For these type of statements, Unity only checks for statement success or failure. If the activity count or success/failure status of the Write operation does not match, Unity automatically changes the table state to *Interrupted* or *Unrecoverable* depending on the type of statement.

## Read-Only on Write Failure Feature

From an Active state, a table may be placed in Read-only if a session encounters an error and the "Read-only on write failure" feature is turned on.

With the **Read-Only on Write Failure** parameter enabled, when a "INSERT to targetTable SELECT FROM sourceTable" fails on one system, the targetTable is Interrupted, a sourceTable becomes Read-Only. As a result, no further writes to sourceTable are allowed on system 1 until the INSERT-SELECT successfully completes recovery. If a write occurs on sourceTable, sourceTable is interrupted on system 1 (will need a recovery).

The feature is used in ETL environments where data is loaded to a stage table, then moved to a base table, and finally moved to a target table. This feature is not needed for a reporting environment. For mixed environment, it is recommended to keep the feature turned on.

### Related Information:

[CONFIG UPDATE](#)

## Sessions Tab



The **Sessions** tab displays the sessions on clients connected to a Teradata Database system that Unity manages. It reports the status for the sessions. The Sessions tab provides the following information.


Field	Description
Unity ID	Generated Unity identification.
TD-Session ID	Generated Teradata Database Session identification.
Username	User associated with session.
Client IP Address	Location of session.
Logon Source	User logon location.
Endpoint Name	Process name.
State	State of session.
Database	Database associated with session.
Connect Time	Timestamp when session connected.
Idle Time	Timestamp when session started waiting.
Routing Rule	Routing rule defined for session.
Routing Type	Managed or passive routing.
Error List Name	Defined error list for session.

## Session States


Session states indicate the current process state of the session. Sessions can be one of the following states.

State	Description
Connecting	Session is connecting to Unity and the Teradata Database system.
Reconnectable	Session connection was lost without the client properly logging off. The connection loss can be caused by network issues. If the client reconnects within 20 minutes, the session is re-established; otherwise, Unity automatically closes the session.
Idle	No transaction is running, and the session is idle. Also known as 'Active: No Transaction' state.
Blocked	Session is waiting for Teradata Database to respond. Also known as 'Active - Waiting' state.
In Transaction	Transaction is in progress. Also known as "Active: In Transaction" state.

## Viewing Blocker Sessions

1. In the **Unity** portlet, click the **Sessions** tab.
2. Click  and select **Blocker Sessions**.

## Viewing Sessions By Locks

1. In the **Unity** portlet, click the **Sessions** tab.
2. Click  and select **By Locks**.

## Session Details View

The **Session Details** view contains the following overview for each session.

Field	Description
State	State of session.
Idle Time	Timestamp when session started waiting.
Endpoint name	Timestamp when session connected.
Routing Name	Name of routing rule.
Error List Name	Name of Error List.
Root Blocker	Cause of session blocking.
Blocking	Any sessions that are blocked by this session.

The Session info for each session includes the connection information.

Field	Description
User name	User associated with session.
Database	Database associated with session.
Client IP Address	Location of session.
Client Host Name	Location of client host.
Connect Time	Timestamp when session connected.
Logon Source	User logon location.

The **Session Details** view also contains Teradata Sessions.

Field	Description
Session ID	Generated Session identification where the query is running.
System Name	System name where the query is running.
State Icon	Icon showing state of session.

Field	Description
In State	Time query has been in current state.
REQ CPU	Number of seconds CPU has been used for query.
Workload	Workload where the query is running.
CPU	Number of CPU seconds query has used in the last minute.
I/O	Number of disk input/outputs the query has performed in the last minute.

The **Session Details** view contains Locks information.

Field	Description
Lock Name	Lock name.
Status	Status of lock.
Lock Type	Type of lock.
Row Hash	Row hash associated with session lock.
Lock Mode	Lock Modes include the following: Exclusive, Access, Write, and Read.
Lock ID	Generated Lock identification.

## Viewing Session Details

1. In the **Unity** portlet, click the **Session** tab.
2. Select a session.  
The session overview appears.

## Lock Modes

The following table lists the types of lock modes. It also defines which modes can block or be blocked by another lock mode. These locks are internal to Unity and are not Teradata Database locks.

Lock Mode	Blocks/Blocked By Modes
Exclusive	Access, Write, Read
Access	Exclusive
Write	Read, Exclusive, other Writes
Read	Write, Exclusive

Unity holds locks for the duration of a transaction. Session mode and TCL statements determine transaction duration. Only one transaction can hold a Write or Exclusive lock on an object at one time, but multiple transactions can be granted Read locks and Access locks simultaneously.

For Read queries, the session isolation level determines whether an Access or Read lock is used. It also determines if any locking modifiers exist in the query and relevant database configuration parameter values. For Write queries, any Read of any object always results in a Read lock. This ensures consistent results across all Teradata systems.

## Aborting Sessions

1. In the **Unity** portlet, click the **Sessions** tab.
2. Select the session to abort.  
The **Session Overview** appears.
3. Click **Abort**.  
Aborting sessions is recorded in the audit log.

## Killing Sessions

Follow these steps to kill a session that is blocking other sessions:

1. In the **Unity** portlet, click **Sessions**.
2. Select the blocking session.  
The **Session Overview** appears.
3. Click **Kill**.  
Killing sessions is recorded in the audit log.

---

**Note:**

A session is killed after the current request finishes processing and a response for it is sent to the client.

---

## Moving Sessions

1. In the **Unity** portlet, click the **Sessions** tab.
2. Select the session that you want to move.  
The **Session Overview** appears.
3. Click **Move**.  
Moving sessions is recorded in the audit log.

## Setting Sessions to Failover

1. In the **Unity** portlet, click the **Sessions** tab.
2. Select the session that you want to set to failover.

The **Session Overview** appears.

3. Click **Failover**.

Sessions that failover are recorded in the audit log.

## Alerts Tab

The **Alerts** tab lists the alerts raised on the Teradata systems configured for Unity management. This collapsed list is associated with the alert and includes all alerts for the selected time period. You can expand the list by clicking on a code. The **Alerts** tab updates by default each minute.

### Note:

Critical alerts displayed in the **Alerts** tab are sent to Teradata Global Support.

### Severity Filter Bar

Sort the alerts by severity.

### Time Period

Includes all the alerts in the select time period.

CODE	TYPE	QUANTITY	SEVERITY	CATEGORY	DESCRIPTION	MOST_RECENT	CLEARED BY	RE
40015	15	3	Warning	MGMT	A failed login attempt	8/17/17 8:03:18 PM	Manual	Syster

## Alerts Tab Metrics

Column	Description
Type	Alert identifier.
Quantity	Number of identical alerts.
Severity	Indicator of the severity level of the alert. The <b>Dashboard</b> uses the following alerts: <ul style="list-style-type: none"> <li>⚠ indicates the severity level is critical and requires the attention of an administrator. The critical threshold is typically pre-configured by the administrator.</li> </ul>

Column	Description
	<ul style="list-style-type: none"> <li>⚠ indicates a warning. A warning alert is a notification that the severity level exceeded the configured thresholds and is approaching critical.</li> <li>🔵 indicates a normal condition. Normal alerts are provided for information purposes only and do not require administrator intervention.</li> </ul>
Category	<p>Category assigned to the alert. Each alert is classified into one of these categories:</p> <ul style="list-style-type: none"> <li>CFG: Configuration alerts related to the configuration of Unity; for example, errors that occur in the Unity configuration file.</li> <li>CONS: Consistency alerts related to data consistency across Unity-managed Teradata Database systems; for example, a database placed in the unrecoverable state due to data mismatch.</li> <li>DBO: Database operation alerts related to errors encountered by the Teradata Database system and sent to Unity; for example, a failed logon attempt.</li> <li>HA: High availability alerts related to the high-availability configuration of Unity; for example, if the sequencers in a dual-system configuration are not in peer states, an alert is raised.</li> <li>MGMT: Management alerts related to errors that occur in Unity management operations; for example, if Unity is unable to suspend the Teradata Database server.</li> <li>SYS: System alerts related to management operation failures and general failures within the Unity environment (not including SQL errors); for example, errors due to loss of a connection to the Teradata Database server.</li> </ul>
Description	Brief statement about the alert condition
Most Recent	Timestamp the alert was generated.
Cleared by	User who cleared the alert.
Resource Type	Process where alert occurred. Processes are sequencer, dispatcher, watchdog, system, and endpoint.
Process	Specific name of process. For example, active_seq.
Region	Name of region where the alert occurred.

## Alerts Tab View

After a code is selected, the alert list expands to include all alerts and is sorted by the alert identifiers. You can select the alert to view all the details for that alert.

## Alerts Tab View Metrics

Column	Description
ID	Alert identifier.
Code	The type of alert.
Type	Alert type.

Column	Description
Severity	Indicator of the severity level of the alert. The <b>Dashboard</b> uses the following alerts: <ul style="list-style-type: none"> <li>🔴 indicates the severity level is critical and requires the attention of an administrator. The critical threshold is typically pre-configured by the administrator.</li> <li>🟡 indicates a warning. A warning alert is a notification that the severity level exceeded the configured thresholds and is approaching critical.</li> <li>🟢 indicates a normal condition. Normal alerts are provided for information purposes only and do not require administrator intervention.</li> </ul>
Category	Category assigned to the alert. Each alert is classified into one of these categories: <ul style="list-style-type: none"> <li>CFG: Configuration alerts related to the configuration of Unity; for example, errors that occur in the Unity configuration file.</li> <li>CONS: Consistency alerts related to data consistency across Unity-managed Teradata Database systems; for example, a database placed in the unrecoverable state due to data mismatch.</li> <li>DBO: Database operation alerts related to errors encountered by the Teradata Database system and sent to Unity; for example, a failed logon attempt.</li> <li>HA: High availability alerts related to the high-availability configuration of Unity; for example, if the sequencers in a dual-system configuration are not in peer states, an alert is raised.</li> <li>MGMT: Management alerts related to errors that occur in Unity management operations; for example, if Unity is unable to suspend the Teradata Database server.</li> <li>SYS: System alerts related to management operation failures and general failures within the Unity environment (not including SQL errors); for example, errors due to loss of a connection to the Teradata Database server.</li> </ul>
Description	Brief statement about the alert condition
Raised at	Timestamp the alert was generated.
Cleared by	User who cleared the alert.
Resource Type	Process where alert occurred. Processes are sequencer, dispatcher, watchdog, system, and endpoint.
Process	Specific name of process. For example, active_seq.
Region	Name of region where the alert occurred.
Operation ID	Every operation in Unity is assigned a unique identifier. Identifiers are tied to the sequencer. Should a sequencer failover, the newly active sequencer will assign its own identifiers.
Repeat Count	Some alerts are encountered multiple times (for example, users failing to login). Instead of raising unique alerts every time, Unity increases the repeat count of the alert.

## Viewing Alert Details

The **Alert Details** view allows you to view additional information about an alert, write recommended actions for the alert, and clear the alert.

1. In the **Unity** portlet, click the **Alerts** tab.
2. Click the row for the alerts you want to expand.
3. Click in the row of the alert to see all the details.

The **Alert Detail** dialog opens and shows detailed information about the selected alert, including a description of the alert.

## Alert Details View Metrics

### General

Column	Description
Alert type	Unique value that identifies an alert.
Alert code	Numeric code for alert.
Severity	Severity level of alert: Normal, Warning, or Critical
Category	Category assigned to the alert: CFG, CONS, DBO, HA, MGMT, SYS.
Cleared by	Name of user who cleared alert.
Description	Brief statement describing the alert condition.
Recommendation	Description of recommended response when this alert is raised.

### Alert Instance

Column	Description
Raised at	Timestamp the alert was generated.
Repeated	Number of times alert has been raised.
Resource type	Name of resource affected, such as Data_Dictionary.
Process ID	Location and process where alert occurs.
Region ID	Location where alert occurs.
Additional info	Additional context information for alert.

### Operations Steps

Field	Description
Operation ID	Operation Identification generated for operation.
Operation Type	Type of operation, such as Recovery Table.
Status	Status of current operation.





Field	Description
Time	Timestamp for operation.
Message	Information about the operation.

## Clearing Alerts

Clearing an alert removes it from the alert list. You can clear all alerts on the **Alerts** tab and you can clear individual alerts in the **Alerts** tab view.

1. In the **Unity** portlet, click the **Alerts** tab.
2. Do one of the following:

Option	Description
Clear Alerts	a. Click  on the row you want to clear, and select <b>Clear Alerts</b> .
Clear Alert	a. Click an alert code to view the individual alerts in the <b>Alerts</b> tab view. b. Click  on the alert you want to clear, and select <b>Clear Alert</b> .

3. [Optional] Enter the reason for clearing the alert.
4. Click **Clear**.

## Alert Scripts

Unity runs the following alert scripts from the Unity server that runs the active sequencer.

Script	Description
alertopen	Script runs each time a new alert is raised.
alertclear	Script runs when an alert clears either automatically or manually.

These scripts must be created in the bin directory. For example, you create the alertopen script in the /opt/teradata/unity/bin/alertopen directory. Sample scripts for alertopen and alertclear are provided in the /opt/teradata/unity/samples directory.

## Operation Details Tab

The **Operation Details** tab displays information about Unity operations running on managed Teradata systems.

Column	Description
Operation Type	Brief description of operation performed, such as Halting Table or Copying Table Queue.

Column	Description
Status	Status values are <i>Finished</i> , <i>In Progress</i> , <i>Canceled</i> , or <i>Failed</i> .
ID	Sequential identification number are assigned to the operation.
Systems	Name of system on which the operation was performed.
Start Time	Timestamp when operation started.
Finish Time	Timestamp when operation completed.
Elapsed Time	Total time to complete operation.
Started By	Administrator who performed the operation.
[Optional] Reason	Reason for action.

## Operation Summary View

To view additional operation details, click the **ID** of an operation in the **Operation Details** tab.

Column	Description
Operation	Description of the operation.
Status	Current status of the operation. Values are <i>Finished</i> , <i>In Progress</i> , <i>Canceled</i> , or <i>Failed</i> .
ID	Unique identification number assigned to the Unity operation.
Systems	Name of one or more systems on which the operation occurred.
Elapsed Time	Date and time of event messages.
Reason	Brief reason for operation
STATUS (icon)	Status if a Warning alert is raised during the operation. The alert icon appears beside the event message to which it relates.
Time	Timestamp for event.
Message	Text of event message logged during the operation.

## Clearing Operations

1. In the **Unity** portlet, select the **Operations Details** tab.
2. Select an operation to remove.
3. Click ☐ and then select **Clear Operation** to remove the selected row.

## Interrupted Sessions Tab

The **Interrupted Sessions** tab on the **Unity** portlet shows the following information for interrupted sessions.

Field	Description
Session ID	Generated session identification.
Transaction ID	Generated transaction identification.
Username	Associated user with session.
System	System location for session.
Initial Time	Time session became interrupted.
Type	Root cause or secondary. Root cause indicates only the interrupted session is the cause. Secondary indicates the session was interrupted because of a dependency on a root cause session.
Error Code	Error code associated with session.
Error Message	Error message associated with session.

## Viewing Interrupted Sessions

1. In the **Unity** portlet, click the **Interrupted Sessions** tab.
2. Select an interrupted session.  
The SQL statement appears.

## About Skipping Transactions

Skipping a transaction may cause other tables to become out of sync. This could result in other tables becoming unrecoverable in subsequent transactions.

## Skipping a Transaction

1. In the **Unity** portlet, click the **Interrupted Sessions** tab.
2. Select the transaction to skip.  
The session detail view appears.
3. Click **Skip Transaction**.  
The **Skip Transaction** view shows which tables are affected.
4. [Optional] Select **Mark these tables unrecoverable**.  
Select this option to change the state of the tables affected by the transaction to Unrecoverable.
5. Click **Skip Transaction** to confirm.  
The skipped transaction is no longer in the summary table.

## Bulk Load Jobs Tab

The **Bulk Load Jobs** tab shows the information for bulk jobs.

Field	Description
Transaction ID	Generated identification number.
Tables	Name of bulk load.
Unity Session	Identification number for Unity session.
Row Count	Table row count verification.
Checkpoints	Number of checkpoints.
Systems Completed	Number of systems that have completed bulk load.

## Bulk Load Details

When you select a bulk load job, you can view the details.

Field	Description
System	System for transaction.
State	State of a system.
Checkpoints	Number of checkpoints per system.

## Viewing Bulk Load Jobs

1. In the **Unity** portlet, click the **Bulk Load Jobs** tab.
2. Click a Bulk Load job to view details in the lower pane.

# Log and Diagnostic Files

## Log File Types

Unity contains the following log file types:

- Process Error and Debug
- Unsync and Interrupt Diagnostic
- Audit
- Dictionary Scanner

## Process Error and Debug Log Files

Each Unity process maintains two log files: an Error and a Debug log file.

Log File Type	Description
Error	Contains all error, warning and info level log messages for the process. The Active Sequencer Error log file contains error and warning level messages from all processes.
Debug	Contains the same information as the Error log file plus trace level log messages.

Unity also provides the following diagnostic files.

Diagnostic File Type	Description
Response mismatch diagnostics	Each time there is a response mismatch between Teradata Database systems, Unity generates a diagnostic file with details. This includes the SQL that caused the mismatch and the different responses returned.
Performance diagnostics	If performance tracing is enabled, Unity generates a diagnostic file with performance metrics. The Unity Dictionary Scanner generates its own log file as part of the schema scanning process.
Unity detected failures	Some TVI alerts are sent from the Unity server directly to Global Support services.

## Unsync Interrupt Diagnostic Log Files

Any Teradata Database system or table that is placed in Unrecoverable or Interrupted state generates a diagnostics file. The file is called `interrupted_diag_system_{system id}_object_{object id}.txt` or `unrecoverable_diag_system_{system id}_object_{object id}.txt`. The system ID is the Unity assigned system ID of the system with the table that is unrecoverable or interrupted. The object ID is the Unity assigned ID of the object involved.

## Audit Log File

The audit log file tracks Unity operations that include the following:

- Changing configuration parameters
- Adding, moving, or removing processes
- Changing names or passwords
- Adding or removing administrators
- Adding or removing error lists
- Importing or updating codes
- Updating or removing user maps
- Marking or canceling sessions for failover
- Terminating sessions
- Adding, reordering, and removing systems

The Audit log is formatted as follows:

*timestamp* : [*user* ] *message*

For example, the following audit log shows when a region was added: 02/03/2016 12:12:03.981:  
[admin] Added region 'NYC'

The Audit log fields are as follows.

Field	Description
timestamp	The timestamp when the entry was logged. In the example, the time is 02/03/2016 12:12:03.981.
user	The name of the Unity administrative user that made the change.
message	A summary of changes and the new value. If available, the message provides the previous value.

## Dictionary Scanner Log File

The scanner.log shows details for scanning jobs. The scannerPerf.log shows start and end times for scan and deploy operations.

The scanner log files are located in the following location:

/opt/teradata/unity/logs/scanner.log

/opt/teradata/unity/logs/scannerPerf.log

The following example shows the progress of the scanner:

```
<logger name="com.teradata.da.unity"> <level value="info"/> <appender-ref
ref="com.teradata.unity.scanner.coreAppender"/> </logger>
```

The following scanner log values are available.

ALL	Includes all standard and custom levels.
DEBUG	Shows events to use to debug.
ERROR	Shows error events.
FATAL	Shows severe error events.
INFO	Shows progress of scanner.
OFF	Turns off logging.
TRACE	Shows events to help debug issues.
WARN	Shows potentially harmful situations.

The following example shows a scanner.log.

```
[2016-11-08 14:43:37,029] (INFO) [main] LoggerConfig.initialize,47: ++++++
+++++
[2020-11-08 14:43:37,032] (INFO) [main] LoggerConfig.initialize,48: ++++++
+++TRACE START+++++
[2020-11-08 14:43:37,033] (INFO) [main] LoggerConfig.initialize,49: ++++++
+++++
[2020-11-08 14:43:37,042] (INFO) [main] ServiceMain.init,102: Load
configuration properties
[2020-11-08 14:43:37,043] (INFO) [main] ServiceMain.init,111: Listen port
is: 61628
[2020-11-08 14:43:37,048] (INFO) [main] ServiceMain.start,59: Scanner
service started
[2020-11-08 14:43:37,057] (INFO) [listenThread] ServiceMain$2.run,72: Receiving
service connections on port 4324
[2020-11-08 14:48:38,418] (WARN) [Thread-1] SocketHandler.run,456: request
header: 00 00 01 20 ff 00 00 00 00 00 01 00 00
[2020-11-08 14:48:38,556] (WARN) [Thread-1] CommandStream.readCommand,123:
found element: ScanRequest
```

## Unity Log File Parameters

The parameters control the path location, size, and attributes for the Unity log files. These parameters are listed in the `unity.properties` file, located in the `/etc/opt/teradata/config` directory. The *type* value can be error, debug, or audit.

Parameter	Description
<code>log.type.size</code>	Size of all Unity log files in megabytes. The default file size is 100 MB.

Parameter	Description
log. type.archivenumber	By default, Unity uses five rotating logs of fixed length. When the current log file is full, the current file moves to a .0 extension, and any existing archive log files increment by one. This value specifies the number of archived log files. For example, if this value is set to 10, Unity archives 10 older log files of the type specified.
log. type.worldreadable	If set to true, log files are readable by all Linux users. If set to false (default), log files are only readable by the Linux user who created the files.

The server keyword marks the entry as affecting all local processes. Prefix each entry with the name of the process it affects, or keyword server, then a period.

Changing the properties file only affects processes on the same physical server as the properties file. The changes are not dynamic. You must restart local processes for changes to take effect.

#### Note:

If making cluster-wide changes for a High Availability installation, make sure you also edit the unity.properties file on all servers where Unity processes reside.

### Archiving an Error File

The following example demonstrates how to set the archive number of the error file to 10 for all processes:

```
server.error.archivenumber=10
```

### Setting Debug Log File to 300

This example sets the size of the region1\_ept debug log file to 300.

```
region1_ept.log.debug.size=300
```

## Log Format

Each entry in the log file has the following format:

*module severity level timestamp:session ID:transaction ID:thread:file and line number:operation: message*

### Log Output Example

Refer to the table below for a description of the values for each log file entry.

```
EPT T 03/17 17:18:21.513:M1001:93:tdpagt-t2:eptstart.c 3538: SQL text: 'BT'
```

Value	Description
Module	Module identifier with one of the following values:
SEQ	Entry refers to a log message from the Sequencer process but not in the context of processing a response from any Dispatcher.



Value	Description	
	EPT	Message from Endpoint process.
	DPT	Message from Dispatcher process.
	WDG	Message from Watchdog process.
	C [nn]	Sequencer logs this message when processing a request from a Dispatcher numbered <i>nn</i> . For example, C00 corresponds to a message the Sequencer logs when processing a response from Dispatcher 0.
	RD [nn]	Dispatcher logs this message and numbers it <i>nn</i> . For example, RDS00 corresponds to a message Dispatcher 0 logs. These messages are added to the Dispatcher log.
	<b>Note:</b> This value can also be RE [nn] for messages relayed from an Endpoint, or RW [nn] for messages relayed from a Watchdog.	
severity log level	Indicates the severity of the message with one of the following values:	
	E	error
	W	warning
	I	informational
	T	trace
timestamp	Timestamp when the entry was logged. For example, 09/09 09:08:26.232 in the Log output. <b>Note:</b> Timestamps in the log file are always written in the local time zone, as defined on the physical box where the process is running.	
session ID	Identifier "m-" for managed sessions with the transaction ID following or "t-" for passive sessions with the request ID following.	
transaction ID	Number of transaction.	
thread	Name of the internal thread that logged the entry. If there are multiple threads of the same name, then the thread number is also included.	
file and line number	Name of the file and line number which logged the message.	
operation	Internal operation being performed when the message was logged.	
message	Message that was logged.	

# Unityadmin Command Reference

## unityadmin Command Line Guidelines

You can run unityadmin commands directly from `/opt/teradata/unity/bin/unityadmin` or at the unityadmin command prompt.

When issuing unityadmin commands, follow these guidelines:

- Use upper or lower case. The commands are not case sensitive.
- Enclose arguments containing spaces in single quotes. For example: `PROCESS STOP rb_dsp_sys3 'stopping to remove';`
- Run only one command at a time.
- Add a semicolon to commands you run from the interactive shell. When you use the `-e` option to run commands directly from the shell prompt, do not use a semicolon.

## Running Commands from the unityadmin Shell

To log on to unityadmin from `/opt/teradata/unity/bin`:

```
/opt/teradata/unity/bin # ./unityadmin -u <username>-P <password> [-e <command>]
```

### user

The Unity Management System user login name.

This must be a valid administrator in the Unity Management System. The default value is the user name in the originating shell.

### password

Unity user password.

### Related Information:

[Running Commands at the unityadmin Prompt](#)

[Syntax Conventions](#)

[Changing States with unityadmin](#)

## unityadmin Shell Parameters

The following table lists optional parameters you can include when running the unityadmin shell.

Command Option	Description
-w <password key>	Uses password stored with TD wallet key <password key> for authentication.
-u <username>	Specifies the Unity username.
-P <password>	Specifies the Unity user password.
-h <host>	Sets the name or IP address of the Unity server to which to connect. The default is localhost.
-p <port>	Sets the listening port of the Sequencer server. The default value is 5344.
-e "<command>"	Enables you to run a single unityadmin command from the command line. The tool exits once the command returns. The user and password must be specified as part of the command-line arguments. The command must be enclosed in quotes. For example, -e "SESSION LIST". To use unityadmin in interactive mode, do not specify -e.
-l <filename>	Sends the session output to the specified log file. There is no default log file.
-nosync or -n	* Sets the unityadmin shell default to asynchronous mode. By default, the unityadmin shell runs all operations in the synchronous mode. You must specify this command option to switch to asynchronous mode. When the command succeeds, the output prints "All operations will be run in asynchronous mode." You can use the abbreviation -n or -nosync for this command option. For example, unityadmin -nosync or unityadmin -n.
-sync or -s	* Sets the unityadmin shell to synchronous mode which blocks each command that runs in the shell until it completes. When a synchronous command succeeds, the output prints "Operations' output will be synchronous." You can use the abbreviation -s or -sync for this command option; for example: unityadmin -s or unityadmin -Sync. This option is enabled by default.
--prompt=<string>	Specifies the command prompt string. The default value is unityadmin.
-?	Displays help information.

\* If both the -n and -s options are entered in the same command, unityadmin uses the last option in the command as the selected operation. For example, in the command `unityadmin -s -n`, unityadmin selects to run all operations in the non-synchronous mode.

## Examples

These examples show a Unity user working in the `/opt/teradata/unity` directory.

The following example shows a user connecting to the Sequencer on the host `ftcp1` and using the default port and username `admin`. The user is prompted for a password.

```
./bin/unityadmin -h ftcp1 -u admin
```

The following example shows a user connecting to the Sequencer on the host `ftcp1` and with the username `admin` and password `adminPass`. The user includes `-e` to run the `SYSTEM ACTIVATE ALL`

command. This format is a useful for running scripting tasks at unityadmin. The unityadmin shell runs by default in synchronous mode, and therefore the -s option used previously to force unityadmin to wait for command completion before continuing is no longer needed.

```
./bin/unityadmin -h ftcp1 -u admin -P adminPass -e "SYSTEM ACTIVATE ALL";
```

The following example shows the Welcome message that displays when you log on. The last line of the message tells you the mode selected for running all operations.

```
unityadmin -u admin -P admin
Login successful, authenticated connection established.

Welcome to the Unity interactive administration shell (version 15.00.00.00)

Commands end with ;
Type 'help' for additional shell commands.

All operations will be run in synchronous mode.
```

## Verifying Unity Status with unityadmin

The following example demonstrates using unityadmin to verify a successful Unity installation.

### Viewing Status Information

Run STATUS to view system details.

```
unityadmin> status;
Sequencer: region1_seq-fastcop1(active)
Sequencer: region2_seq-bltorcop1(standby-synchronized)
Endpoint region1_ept: 10.25.1.4(not listening):1025,
153.65.194.99(listening):1025, 10.25.1.9(not listening):1025,
153.65.194.55(not listening):1025
Endpoint region2_ept: any(listening):1025
System prod1(unrecoverable) Tables: OOS 0, standby 0, unrecoverable 0,
interrupted 0, restore 0, read-only 0, active 0
Dispatcher status: region2_dsp_prod1(standby) - down, region1_dsp_prod1(active)
- up
Gateway status: flash - prod1cop1(up)
System blaze(active) Tables: OOS 0, standby 0, unrecoverable 0, interrupted 0,
restore 0, read-only 0, active 0
Dispatcher status: region2_dsp_prod2(active) - up, region1_dsp_prod2(standby)
- up
```

```
Gateway status: prod2 - prod2cop1(up)
Alerts: 0 critical, 1 warning, 1 normal
```

Output	Description	Status
Sequencers	<ul style="list-style-type: none"> <li>Name of Sequencer</li> <li>Current state (in parentheses)</li> </ul>	When Unity is functioning properly, one Sequencer (region1_seq-fastcop1) should be in the Active state. The second Sequencer (region2_seq-bltorcop1) should be in the standby state (the sub-state can be synchronizing, syncing, synced or synchronized). For more information, see <a href="#">SEQUENCER STATUS</a> .
Endpoints	<ul style="list-style-type: none"> <li>Name of Endpoint</li> <li>List of all IP addresses (or the special keyword "any") where the Endpoint is configured for listening</li> <li>Current state (listening, not listening, etc.) for each IP address</li> <li>Associated port for each IP address</li> </ul>	All Endpoints (region1_ept, region2_ept) should be connected. Each IP address should be in the proper listening/not listening state (10.25.1.4(not listening)). Endpoints listen on all default IP addresses, and not listen on standby IP addresses (unless requested to do so specifically by the Sequencer). If an Endpoint is not connected, "not connected" appears instead of a list of associated IP addresses.
Systems	<ul style="list-style-type: none"> <li>Name of the system</li> <li>State (in parentheses)</li> <li>List of all Unity-managed table counts grouped by table state (with Out of Service state shortened to OOS)</li> </ul>	The first list shows the flash system, which is in the unrecoverable state. There are currently no objects in the Unity dictionary, so that count of tables in each state is 0. There are two dispatchers defined. Dispatcher (region2_dsp_flash) is in Standby state, and is not connected to the Sequencer (state: down). Dispatcher (region1_dsp_flash) is the Active dispatcher, and is connected to the Sequencer (state: up).
Alerts	<ul style="list-style-type: none"> <li>Count of critical, warning, and normal alerts open.</li> </ul>	There should be no critical alerts, but warning alerts can show up as part of normal system operation and don't need to be cleared. Normal alerts can appear during system startup. To see all open alerts in a system, use the <a href="#">ALERT LIST OPEN</a> command.

## Running Commands at the unityadmin Prompt

The unityadmin command provides all of the functionality available in the Unity portlets as well as additional control. The unityadmin command establishes a socket connection to the Sequencer process.

- At the command line prompt, type:

```
/opt/teradata/unity/bin # ./unityadmin -u user -P password
```

Variable	Description
<i>user</i>	Unity Management System user login name (If the -u option is not specified, the default values becomes the user name in the originating shell.)

Variable	Description
<i>password</i>	User password

**Tip:**

Use single quotes when running two-word commands. For example: `/opt/teradata/unity/bin # ./unityadmin -u user -P password -e 'SESSION LIST'`

**Related Information:**

[Running Commands from the unityadmin Shell](#)

[Syntax Conventions](#)

[Changing States with unityadmin](#)

## Syntax Conventions

The unityadmin commands use the convention of all capital letters to indicate specific unityadmin command names, such as SYSTEM LIST.

The following table lists annotations used to describe command syntax.

Object	Description	How used
	Vertical bars	Separates alternative, mutually exclusive elements
[ ]	Square brackets	Indicates optional elements
<b>bold</b>	Boldface type	Indicates typed exactly as shown
<i>italics</i>	Italics type	Indicates elements for which you supply the values
{ }	Curly braces	Indicates grouped parameters

**Related Information:**

[Running Commands from the unityadmin Shell](#)

[Running Commands at the unityadmin Prompt](#)

[Changing States with unityadmin](#)

## Changing States with unityadmin

### Teradata System State Changes

You can change the state of Teradata systems with unityadmin. The Teradata system state must be Active, Standby, or Read Only to start sessions on the Teradata system. The following table shows how to change Teradata system accessibility and Unity states.

From	To	Commands
Active	Standby	SYSTEM DEMOTE
Standby	Active	SYSTEM PROMOTE

## To Active State Commands

The following table describes how to place a Teradata system or table in Active state from another state.

To Active from	Commands
Out of Service	SYSTEM RECOVER
Standby	SYSTEM PROMOTE
Read Only	SYSTEM RECOVER

### Note:

If a table is in a Restore state, it automatically goes into an Active state when recovery is complete.

## From Active State Commands

From Active state, a Teradata system or table can be placed into the following states.

From Active to	Commands
Unrecoverable	SYSTEM DEACTIVATE
Out of Service	SYSTEM HALT
Read Only	SYSTEM FREEZE
Standby	SYSTEM DEMOTE

### Note:

From an Active state, a system is automatically placed in Restore state if the queue of pending Write requests is greater than the **RecoveryQueueLimit** parameter. This prevents the system from getting reads directed to it when it is far behind in writes, allowing the system time to catch up with the other systems in the cluster.

### Related Information:

[Running Commands from the unityadmin Shell](#)

[Running Commands at the unityadmin Prompt](#)

[Syntax Conventions](#)

[Supported State Transitions](#)

# ADMIN ADD

## Purpose

Adds a new Unity server administrator.

## Syntax

```
ADMIN ADD userid username password | LDAP admin | readonly ;
```

## Parameters

### *admin*

Adds user with full administrative privileges.

### *password*

User password.

- Must be at least five characters long
- Alphanumeric characters only.

### LDAP

Adds a new server administration user. If password is set to LDAP, LDAP authentication is used using the LDAP server set in the 'ldapServerURI' config setting.

### *readonly*

Adds user with read only privileges.

### *userid*

Unique user identifier (ID).

### *username*

Name of the user. Name can contain a maximum of 25 characters.

## Usage Considerations

This command results in an error if the *userid* value already exists in Unity.

## Example

```
unityadmin> ADMIN ADD adminExampleID Administrator 123456 ADMIN;
Successfully added the administrator 'adminExampleID' (role ADMIN).
```



# ADMIN CHANGE NAME

## Purpose

Allows an administrator to change the user's descriptive name associated with the specified *userid* . This command results in an error if the *userid* does not exist.

## Syntax

```
ADMIN CHANGE NAME userid newusername ;
```

## Parameters

*userid*

Unique user identifier (ID).

*newusername*

New user name.

## Usage Considerations

This command results in an error if a *username* corresponding with the *userid* does not exist in Unity.

## Examples

```
unityadmin> ADMIN CHANGE NAME adminExampleID Admin2;  
Successfully changed the name of administrator 'adminExampleID'.
```

# ADMIN CHANGE PASSWORD

## Purpose

Allows an administrator to change the password associated with a specific *userid* . This command results in an error if the *userid* does not exist.

## Syntax

```
ADMIN CHANGE PASSWORD userid password | LDAP ;
```

## Parameters

*userid*

Unique user identifier (ID).

***password***

New administrator password.

**LDAP**

Changes the password of the specified administrator. If password is set to LDAP, LDAP authentication is used using the LDAP server set in the 'ldapServerURI' config setting.

**Usage Considerations**

This command results in an error if the user id value does not exist in Unity.

**Examples**

```
unityadmin> ADMIN CHANGE PASSWORD adminExampleID 654321;
Successfully changed the password of administrator 'adminExampleID'.
```

## ADMIN CHANGE ROLE

**Purpose**

Changes the role associated with a *userid*.

**Syntax**

```
ADMIN CHANGE ROLE userid role ;
```

**Parameters*****userid***

Unique user identifier (ID).

***role***

Represents the user permissions.

Option	Description
<b>Admin</b>	Administrator access
<b>readonly</b>	view-only access

**Examples**

```
unityadmin> ADMIN CHANGE ROLE adminExampleID READONLY;
Successfully changed the role of administrator 'adminExampleID' to readonly.
```

## ADMIN LIST

### Purpose

Lists the current Unity administrators.

### Syntax

ADMIN LIST ;

### Example

```
unityadmin> ADMIN LIST;
Login:    admin
Name:     Main Administration User
Role:     Admin
-----
Login:    adminExampleID
Name:     Apollo
Role:     readonly
```

The output displays the following information.

Label	Description
Login	User ID
Name	Display name for the user
Role	Admin or Readonly

## ADMIN REMOVE

### Purpose

Removes the specified *userid* from the Unity-managed systems.

### Syntax

ADMIN REMOVE *userid* ;

### Parameters

*userid*

Unique user identifier (ID).

## Usage Considerations

- This command results in an error if the user id does not exist.
- The internal user admin cannot be removed.

## Examples

```
unityadmin> ADMIN REMOVE adminExampleID;
Successfully removed the administrator 'adminExampleID'.
```

# ALERT CLEAR

## Purpose

Clears a specified alert number and stores the reason for clearing it.

## Syntax

```
ALERT CLEAR alertNumber REASON 'reason' ;
```

## Parameters

### *alertNumber*

Number identifying the alert to be cleared.

### REASON

Reason for clearing. Use single quotation marks when the reason is more than one word.

## Example

```
unityadmin> ALERT CLEAR 1 'test';
The alert '1' was successfully cleared.
```

# ALERT CLEAR ALL

## Purpose

Clears all open alerts by severity or alert number with a common reason.

## Syntax

```
ALERT CLEAR ALL [CRITICAL 'reason'] | [LESS THAN alertNumber] | [WARNING 'reason'] ;
```

## Parameters

### CRITICAL '*reason*'

Enables clearing all critical alerts. The reason for clearing alerts specified as a single quoted string with a maximum length of 1024 characters.

### LESS THAN *alertNumber*

Enables clearing all open alerts with a number lower than the provided alert ID.

### WARNING '*reason*'

Enables clearing all open Warning severity alerts.

## Examples

Demonstrates how to clear all open alerts with Warning severity level.

```
unityadmin> ALERT CLEAR ALL WARNING 'Open warning alerts have been cleared';
Bulk clear of all alert warnings in progress
```

Demonstrates how to clear all open alerts with Critical severity level.

```
unityadmin> ALERT CLEAR ALL CRITICAL 'Open critical alerts have been cleared';
Bulk clear of all critical alerts in progress
```

Demonstrates how to clear all open alerts with an alert ID lower than 100.

```
unityadmin> ALERT CLEAR ALL LESS THAN 100 'Cleared all open alerts with an
alert ID lower than 100';
Performing bulk clear of alerts with ID less than 100
```

# ALERT CLEAR ALL BEFORE

## Purpose

Clears all alerts in bulk before time listed.

## Syntax

```
ALERT CLEAR ALL BEFORE YYYY-MM-DD HH:MM:SS 'reason'
```

## Parameters

### YYYY-MM-DD HH:MM:SS

Use a four-digit year number.

**REASON**

Reason for clearing. Use single quotation marks when the reason is more than one word.

**Examples**

To clear all alerts before January 1st, 2017 at 12:30:01, use the following command:

```
ALERT CLEAR ALL BEFORE 2017-01-01 12:30:01 'upgrade 16.00.00'
```

## ALERT CLEAR RANGE

**Purpose**

Clears a range of alerts between start and end alert numbers and tags them with a common reason.

**Syntax**

```
ALERT CLEAR RANGE starting alertNumber ending alertNumber 'reason' ;
```

**Parameters*****starting alertNumber***

Number identifying the alert number identifying the lowest-numbered alert in the range of alerts to be cleared.

***ending alertNumber***

Number identifying the alert number identifying the highest-numbered alert in the range of alerts to be cleared.

***'reason'***

Reason for clearing. Use quotation marks when the reason is more than one word.

**Example**

```
unityadmin> ALERT CLEAR RANGE 1 100 'clearing first 100 alerts';
Bulk alert clearing attempt complete.
```

## ALERT DISABLE

**Purpose**

Disables one or more alerts by ID.

**Syntax**

```
ALERT DISABLE Alert Type id [, Alert Type id, ...] ;
```

**Parameters****alerttypeid**

Represents the unique number assigned to an alert.

**Usage Considerations**

Lists as Alert no. if you run ALERT LIST TYPES.

**Example**

```
unityadmin> ALERT DISABLE 170;
Alert(s) have been disabled. See sequencer.log for details.
```

## ALERT ENABLE

**Purpose**

Enables one or more alerts by ID.

**Syntax**

```
ALERT ENABLE Alert Type id [, Alert Type id, ...] ;
```

**Parameters****alerttypeid**

Represents the unique number assigned to an alert.

**Usage Considerations**

Lists as Alert Type if you run ALERT LIST TYPES.

**Example**

```
unityadmin> ALERT ENABLE 170;
Alert(s) have been enabled. See sequencer.log for details.
```

## ALERT LIST ALL

## Purpose

Lists all raised and cleared alerts for the Unity server.

## Syntax

ALERT LIST ALL [NORMAL | WARNING | CRITICAL] ;

## Example

```
unityadmin> ALERT LIST ALL;
-----
The following alerts satisfy your request
-----
Alert No.           : 1
Alert Code          : 40001
Alert Description   : Unity sequencer has started
Alert Details       : Unity Sequencer started
Resource Type       : SEQUENCER
Resource ID         : cressida
Alert Severity      : Normal
Alert State         : Opened
Repeated           : 0
Raised Time         : 09/20 10:54:15
Updated Time        : 12/08 11:18:09
-----
Alert No.           : 12
Alert Code          : 40046
Alert Description   : Unity does not recognize one or more objects used in this
request.

Alert Details       : Dictionary lookup failure: Database object 'lod008_e2'
not
                                     found in Unity dictionary for
database 'TPTLOAD' (' DROP TABLE LOD008_e2; ')
Alert Category      : Database Operations
Resource Type       : Data_Dictionary
Resource ID         : cressida
Alert Severity      : Warning
Alert State         : Closed
Repeated           : 0
Raised Time         : 01/25 04:54:12
Updated Time        : 02/08 11:18:09
-----
```



```

unityadmin> ALERT LIST ALL CRITICAL;
-----
The following alerts satisfy your request
-----
Alert No.: 21
Alert Type: 40033
Alert Description: A recovery operation on an object has been interrupted.
Recovery can continue when the current condition is resolved. Refer to the
Teradata Unity User Guide for more information.
Alert Details: System 2 (db2) sent an inconsistent message, table:
dbtest.gs114892_a, session: 1000 are interrupted:
Transaction: 7
SQL To Unity: Unspecified / No
Expected response [success] row count 0
Code: 0
Message: Success
Actual response: row count 0
Code: 3803
Message: Table 'GS114892_a' already exists.

Alert Category: Consistency
Resource Type: Data_Sync
Resource ID: unity1
Alert Severity: Critical
Alert State: Raised
Repeated: 0
Raised Time: 03/27 09:59:06
Updated Time: 03/27 09:59:06
unityadmin>

```

## ALERT LIST OPEN

### Purpose

Lists all currently raised alerts for the Unity server.

Open alerts can be listed by type.

### Syntax

```
ALERT LIST OPEN [NORMAL | WARNING | CRITICAL] ;
```

**Example**

```

unityadmin> ALERT LIST OPEN;
-----
The following alerts satisfy your request
-----
Alert No.           : 1
Alert Type          : 40001
Alert Description   : Unity sequencer has started
Alert Details       : Unity Sequencer started
Alert Category      : System
Resource Type       : SEQUENCER
Resource ID         : cressida
Alert Severity      : Normal
Alert State         : Opened
Repeated           : 11
Raised Time         : 09/20 10:54:16
Updated Time        : 12/08 11:18:09
-----

```

The output displays in the following format.

Label	Description
Alert No.	A unique number identifying the alert.
Alert Type	A unique number corresponding to the alert type.
Alert Description	A description of the alert condition.
Alert Details	Additional alert information specific to this instance of the alert.
Alert Category	Available alert categories are: System, Sequencer, Managed Teradata, Dispatcher, Endpoint, Watchdog, Data Dictionary, Table State Change, Threshold Violation, Data Sync.
Resource Type	Specifies the resource affected by the alert, options include SEQUENCER, WATCHDOG, DISPATCHER, and ENDPOINT.
Resource ID	Name of the Teradata system associated with the affected resource type.
Alert Severity	Severity of the alert condition: <ul style="list-style-type: none"> <li>• Critical – Events indicating that the application cannot access some data. For example, one table is unrecoverable.</li> <li>• Warning – Events indicating non-serious failures that could impact Unity operation. For example, Unity configuration errors.</li> <li>• Normal – Informative messages. For example, the Sequencer has started. These alerts are typically used for startup.</li> </ul>
Alert State	<ul style="list-style-type: none"> <li>• Opened - The alert first appears.</li> </ul>

Label	Description
	<ul style="list-style-type: none"> <li>Cleared - The alert has been closed.</li> </ul>
Repeated	<p>The number of times the alert repeated since the original alert was raised. The alert count is based on the following rules:</p> <ul style="list-style-type: none"> <li>The first time an error condition is detected, an alert is raised with a count set to 0.</li> <li>After the initial alert is raised, if the same error condition is detected, the initial alert updates with a count based on the number of times the alert has been raised.</li> </ul>
Raised Time	The timestamp when the alert was opened.
Updated Time	The timestamp when the alert was last updated. This is equal to the raised time if the alert has not been updated.

## ALERT LIST TYPES

### Purpose

Lists all alerts and type IDs for ENABLE or DISABLE commands.

### Syntax

ALERT LIST TYPES ;

### Example

```
unityadmin> ALERT LIST TYPES;
-----
The following alerts are defined
Alert Type          : 1
Alert Description    : Unity sequencer has started
Alert Category       : System
Alert Severity       : Normal
Alert Status         : Enabled
Recommended Action   : Normal system operation. Alert must be manually cleared.
-----
Alert Type          : 2
Alert Description    : An error has prevented a Unity process from starting. See
log for details
Alert Category       : System
Alert Severity       : Critical
Alert Status         : Enabled
Recommended Action   : Correct the error and try starting the process again.
-----
```

The output displays in this format.

Label	Description
Alert Type	The alert number for this alert. Used when marking alerts as enabled or disabled.
Alert Description	A general description.
Alert Category	Which category/part of Unity the alert belongs to.
Alert Severity	The severity of the alert (Normal, Warning, Critical).
Alert Status	Enabled or disabled.
Recommended Action	A default recommended action. This can be updated.

## ALERT TEST

### Purpose

Use when testing with an enterprise Teradata system.

Use in conjunction with ALERT CLEAR command to clear an alert. The scripts called when an alert is opened or closed are /opt/teradata/unity/bin/alertopen and /opt/teradata/unity/bin/alertclear respectively. Samples alertopen and alertclear scripts are provided in /opt/teradata/unity/samples/ directory.

### Syntax

ALERT TEST *Alert Type id* ;

### Parameters

**alerttypeid**

Represents the unique number assigned to an alert.

### Example

```
unityadmin> ALERT TEST 170;
UE Alert type 170 raised
```

## AVAILABILITY STATS

### Purpose

Shows overall and system availability metrics for Unity.

## Syntax

AVAILABILITY STATS ;

## Example

```

unityadmin> AVAILABILITY STATS;
Availability Statistics:
-----
Total time since 'In Service'      : 6 days, 15 hours, 7 minutes
Unity Availability                  : 6 days, 15 hours, 7 minutes
( 100.000 % )
Managed system 'db1' availability : 6 days, 5 hours, 7 minutes ( 93.715 % )
Managed system 'db2' availability : 6 days, 5 hours, 16 minutes ( 93.810 % )

```

# CONFIG LIST

## Purpose

Lists all configuration parameters of a process type or specific registered process.

## Syntax

CONFIG LIST [ *process type* | *process id* ] [ KEY *key1* [ *key2*,[...] ] ] [ **USER** | **FULL** ] ;

## Parameters

### FULL | USER

Displays the full hierarchy of values: global default values, global user values, process type default values, process type user values, and process id user values.

Displays only user-set values (Unity system default values do not display).

### *processid*

Identifier assigned to a specific process.

### *process type*

If specified, displays configuration options for a specified Unity process type:

- Endpoint
- Dispatcher
- Sequencer
- Watchdog

**KEY**

Identifies configuration keys.

**Usage Considerations**

Use the `PROCESS STATUS` command to list all processes registered with Unity.

**Examples**

Demonstrates listing specific process configuration options for a dispatcher process with the id `dispatcher-1`.

```
unityadmin> CONFIG LIST dispatcher-1;
dispatcher-1
-----
Tdpid = system1
Region = toronto
AccessMode = read-write
LogErrorsize = 50
LogPath = /opt/Teradata Unity/dispatcherlogs
...
```

Demonstrates using the **FULL** parameter to list the hierarchy of values for a dispatcher process with the id `dispatcher-1`.

```
unityadmin> CONFIG LIST dispatcher-1 FULL;
Global Default Values
-----
LogErrorSize = 10
LogPath = /opt/Teradata Unity/logs
....
Global User Set Values
-----
AccessMode = read-write
Dispatcher User Set Values
-----
LogPath = /opt/Teradata Unity/dispatcherlogs
dispatcher-1
-----
Tdpid = system1
Region = toronto
AccessMode = read-write
LogErrorsize = 50
LogPath = /opt/Teradata Unity/dispatcherlogs
...
```

Demonstrates using the **USER** parameter to list user-set values for a dispatcher process with the id dispatcher-1.

```
unityadmin> CONFIG LIST dispatcher-1 USER;
dispatcher-1
-----
Tdpid = system1
Region = toronto
```

Demonstrates listing specific process configuration options for the Endpoint process type.

```
unityadmin> CONFIG LIST Endpoint;
Endpoint 'toronto_ept'
-----
HAFailoverMode: manual
EndpointDefaultPort: 1025
RecoveryTableName: unitymanagement.txntable
BlockedSessionThreshold_5: 120:20:40
RecoveryLogThreshold: 60:80
EndpointAddresses: any
ReadOverrideCport: 6001
SpaceUsageThreshold: 60:80
PacketTraceEnabled: 0
ProcessPingTimeout: 30
BlockedSessionThreshold_1: 10:60:100
BlockedSessionThreshold_2: 20:50:80
BlockedSessionThreshold_3: 30:40:60
BlockedSessionThreshold_4: 60:30:50
HashFuncDBC: 5
DictionaryAllowUnknownFunctions: false
```

Lists the hierarchy of values for the Endpoint process type.

```
unityadmin> CONFIG LIST Endpoint FULL;
System Set Globals
-----

HAFailoverMode: manual
RecoveryTableName: unitymanagement.txntable
BlockedSessionThreshold_5: 120:20:40
RecoveryLogThreshold: 60:80
SpaceUsageThreshold: 60:80
PacketTraceEnabled: 0
```

```

ProcessPingTimeout: 30
BlockedSessionThreshold_1: 10:60:100
BlockedSessionThreshold_2: 20:50:80
BlockedSessionThreshold_3: 30:40:60
BlockedSessionThreshold_4: 60:30:50
HashFuncDBC: 5
DictionaryAllowUnknownFunctions: false

```

System Set for Endpoint

-----

```

EndpointDefaultPort: 1025
EndpointAddresses: any
ReadOverrideCport: 6001

```

Lists values for the keys AutoSelectOnDatabaseScan and LoadNumSessions.

```

unityadmin> config list dispatcher key AutoSelectOnDatabaseScan,
LoadNumSessions full;
System Set Globals
-----
AutoSelectOnDatabaseScan: false
LoadNumSessions: 16

User Set Globals
-----
LoadNumSessions: 20

```

Lists dispatcher processes where the LoadNumSessions value changed and are awaiting restart.

```

unityadmin> config list dispatcher key LoadNumSessions full;
System Set Globals
-----
LoadNumSessions: 16

User Set Globals
-----
LoadNumSessions: change requires restart
-----
region1_dsp_jasonu2      current value: 20, after restart: 21
region1_dsp_jasonu4      current value: 20, after restart: 21
region2_dsp_jasonu4      current value: 20, after restart: 21
region2_dsp_jasonu2      current value: 20, after restart: 21
-----

```



## CONFIG RESET

### Purpose

Resets the parameter to the specified value.

### Syntax

```
CONFIG RESET [ process type | process id] value ;
```

### Parameters

#### *processid*

Identifier assigned to a specific process.

#### *process type*

If specified, resets the value parameter for a specified Unity process type:

- Endpoint
- Dispatcher

#### *value*

The configuration parameter value to be reset.

### Example

The following example show how to use the CONFIG UPDATE command to add a system wide TDPID definition with the config key UnityTDPID. Some configuration parameter changes require a Unity cluster restart as shown in the example. After the CONFIG UPDATE command, run the CONFIG RESET command to reset the value of the UnityTDPID to its original value.

```
unityadmin> CONFIG UPDATE UnityTDPID globalTDPID REASON 'New TDPID';
Requested config added/updated successfully.
Note that configuration changes require a restart of the affected processes
before taking effect.

CONFIG RESET UnityTDPID;
Reset request completed successfully.
```

## CONFIG UPDATE

### Purpose

Updates a parameter value.

## Syntax

CONFIG UPDATE [ *<process type>* | *<process id>* ] *<parameter>**<value>* REASON *<reason for configuration change>*;

## Parameters

### *processid*

Identifier assigned to a specific process.

### *process type*

Specifies Unity process type to update. Process types are:

- Endpoint
- Dispatcher
- Sequencer
- Watchdog

### *parameter*

Specifies the configuration key to update.

### *value*

Specifies the value for the configuration parameter.

## Usage Considerations

- To set a parameter globally, do not include *process type* or *process id* values.
- Certain configuration parameters can only be at certain levels where *process type* or *process id* is mandatory. Updating configuration parameters is recorded in the audit log.

The PROCESS LIST command displays process ids.

## Example

This shows how to update the Sequencer's recovery interval frequency to 300 seconds.

```
CONFIG UPDATE Sequencer RecoveryInterval 300 REASON 'Setting recovery interval to 5 min';
```

## SEQUENCER Configuration Parameters

The following parameters are available at the Sequencer level.

Parameter	Default	Valid Values	Scope	Description
<b>ExportMaxJobs</b>	5	0 to infinity, inclusive		Sets the maximum number of export jobs allowed on a Unity system.
<b>LoadRequestTimeout</b>	90 seconds	Value type: Integer Allowed value range: greater than or equal to 5	Only for sequencer	Sets the timeout length for a load request from the Sequencer. Use to prevent timeouts for the load request from the active sequencer, for example, when the two Unity regions have significant network delays.

## DISPATCHER Configuration Parameters

Parameter	Default	Valid Values	Scope	Description
<b>DatabaseConnectionTimeout</b>	120	30 to 1200	Process Type or Per Process	Period of time (in seconds) allowed to pass before the Dispatcher process detects a lost Gateway connection. The minimum value you can set is 30 seconds. The maximum value is 1200 seconds.
<b>DispatcherWorkpoolSizeMax</b>	null		Process Type or Per Process	Maximum number of worker threads in the Dispatcher worker pool. If the value is undefined (null) or less than the DispatcherWorkpoolSizeMin value, the Dispatcher uses the DispatcherWorkpoolSizeMin value or a multiple of five times the total number of CPU cores, whichever is greater.
<b>DispatcherWorkpoolSizeMin</b>	5		Process Type or Per Process	Minimum number of worker threads in the Dispatcher worker pool. The Dispatcher uses multiple threads in a worker pool to process SQL statements from the Unity server and forwards them to the Teradata Database system. The Dispatcher automatically adjusts the size of the worker pool based on the current number of sessions and the transaction rate.

Parameter	Default	Valid Values	Scope	Description
<b>ManagementPassword</b>	No default		Process Type or Per Process	ManagementUserId password.
<b>ManagementUserId</b>	No default		Process Type or Per Process	Identifier for the Unity server user. This is the name of the Teradata user through which the Dispatcher and Sequencer processes connect to the Teradata Database system. The <b>userid</b> must be configured for TD2 authentication, and be able to read Unity Data Dictionaries and to create database tables.
<b>MonitorDbConfigInterval</b>	3600	300 to infinity		The interval in seconds at which the database settings (specifically <b>HashFuncDBC</b> and <b>FixSRandomRange</b> ) are checked by the dispatcher. To disable the monitor, use the <b>MonitorHashFuncDBC</b> and <b>MonitorFixSRandomRange</b> parameters.
<b>MonitorFixSRandomRange</b>	True	True or False		Monitor control. This allows you to make sure the value of <b>FixSRandomRange</b> is consistent across the managed Teradata Database systems and Unity. If the value of <b>FixSRandomRange</b> is different, an alert is raised.
<b>MonitorHashFuncDBC</b>	True	True or False		Monitor control. This allows you to make sure the value of <b>HashFuncDBC</b> is consistent across the managed Teradata Database systems and Unity. If the value of <b>HashFuncDBC</b> is different, an alert is raised.
<b>MonitorInterval</b>	30		Process Type or Per Process	Period of time (in seconds) that the Dispatcher process takes to connect to each IP address and perform a configuration handshake which determines the availability of the IP address and the Gateway.

Parameter	Default	Valid Values	Scope	Description
<b>MonitorSystemSerialNumber</b>	True	True or False		Monitor control. This feature creates and assigns serial numbers to the Unity-managed systems, so that Unity can verify that system gateway addresses are pointing to the correct system. The verification happens periodically as well as on system activation and recovery.
<b>MonitorSystemSerialNumberInterval</b>	1800	300 to infinity		The interval in seconds at which the system serial number will be periodically checked by the dispatcher. To disable the monitor, use the <b>MonitorSystemSerialNumber</b> parameter.
<b>PrecedingMessageWaitTimeout</b>	30	5 to infinity	Process Type	Period of time (in seconds) the Dispatcher waits for a preceding lock information message for a specific Endpoint request. On timeout, if a request is missing the preceding message, the request fails with: 4533 - Timeout waiting for preceding message. Please resubmit the request.
<b>TDMSTPort</b>	1025		Process Type or Per Process	Port number on which the Dispatcher connects to the Teradata Database system.
<b>Tdpid</b>	No default		Process Type or Process	TDPID of the Teradata Database system managed by Unity that allows the Dispatcher to connect to the system. In a high-availability configuration, you must assign two Dispatchers the same TDPID value, one to each Region in the configuration.

## ENDPOINT Configuration Parameters

Parameter	Default	Scope	Option	Description
<b>EndpointAddresses</b>	No default	Per Process	Required	<p>List of IP addresses and optional ports on which the Endpoint process listens for client requests. Use commas to separate multiple IP addresses. The format of this parameter is:</p> <pre>any[:port]   &lt;ip&gt;[:&lt;port&gt;][,&lt;ip&gt;[:&lt;port&gt;], ...]</pre> <p>The keyword any indicates that the Endpoint listens on all defined IP addresses on the Teradata system. The EndpointDefaultPort parameter specifies the port value.</p> <p>If the keyword any is not used, you can provide specific IP address values. The &lt;port&gt; value is optional. Any IP addresses given without a port value default to the value defined in the EndpointDefaultPort parameter.</p>
<b>EndpointDefaultPort</b>	1025	Per Process	Optional	Port number for an IP address that is not defined as a port in EndpointAddresses. Client applications and users connect to this port.
<b>EndpointSIPAddresses</b>	No default	Per Process	Optional	<p>List of Standby IP (SIP) addresses that the Endpoint process enables if Endpoints in other Regions are not available. Use commas to separate multiple SIP addresses. The SIP address values cannot be the same as the IP address values defined in EndpointAddresses. If the EndpointAddresses value is set to any, the EndpointSIPAddresses are ignored.</p> <p>The format of this parameter is:</p>

Parameter	Default	Scope	Option	Description
				<pre>&lt;region&gt; &lt;ip&gt;[:&lt;port&gt;] [,&lt;ip&gt; [:&lt;port&gt;], ...][; &lt;region&gt; &lt;ip&gt;[:&lt;port&gt;] [,&lt;ip&gt; [:&lt;port&gt;], ...]]</pre> <p>Because this parameter can support the failover of Endpoint processes from multiple Regions, the <i>&lt;region&gt;</i> value specifies the name of the Regions for which the following IP address is enabled if the Endpoints in that Region are not available. For example:</p> <pre>toronto 192.168.0.1, 192.168.0.2; sandiego 192.168.0.3, 192.168. 0.4</pre> <p>This example shows that if all Endpoints in the Toronto Region are unavailable, then this Endpoint enables 192.168.0.1 and 192.168.0.2 for available connections from clients that were connecting to the Endpoints in Toronto. It also specifies that if all the Endpoints in San Diego Region are unavailable, then this Endpoint enables 192.168.0.3 and 192.168.0.4 for available connections from clients that were connecting to Endpoints in San Diego.</p> <p><b>Note:</b></p> <p>The <i>&lt;region&gt;</i> in this parameter cannot be the same as the Region value for the Endpoint process. The <i>&lt;ip&gt;[:&lt;port&gt;]</i> values cannot be the same as those defined in the <b>EndpointAddresses</b> parameter.</p> <p>If the EndpointAddresses value is set to any, then values in the EndpointSIPAddresses are ignored.</p>

Parameter	Default	Scope	Option	Description
<b>LoadCheckPointsLimit</b>	0 (Disabled)			<p>When the parameter is set to 0, Unity does not check the number of allowed checkpoints. When the parameter is set to a positive number and the number of checkpoints exceeds the number, Unity aborts the load to all systems and forces a rollback. Unity returns an error 4536 to client: "Number of checkpoints exceeded checkpoint limit setting." Restarted loads count from zero again. For example, if the LoadCheckPointsLimit is set to 100, and a load job takes about 250 checkpoints, the Unity forces a rollback at checkpoint 101, then restarts the load and only counts the checkpoints for the restart. Then aborts after 100 and starts the load again until the load is complete.</p> <ul style="list-style-type: none"> <li>• If a load client sends several checkpoint 0, Unity only counts it once.</li> <li>• If the MLoad client sends several CHECKPOINT interval 0 (or several same checkpoint interval value on restart), each of them is counted as a checkpoint seen. (The checkpoint interval value of the MLoad client can be less than the Checkpoint seen.)</li> </ul>
<b>LoadChunkTimeoutMS</b>	60000			Time (in microseconds) that the endpoint waits for a data chunk response from the managed system. By default, it is set to 60000, with a minimum value of 10000, and no maximum value limit. Unity will reject any values underneath the minimum value.
<b>LoadDfsFileCloseAsync</b>	False		Optional	With the default (False) value, loads to managed multiple database systems are not completely asynchronous. Load files are written to remote nodes asynchronously. However, at a load checkpoint



Parameter	Default	Scope	Option	Description
				<p>end or load end, when closing load data files, Unity waits for all outstanding remote load data writes to finish. Loads to multiple database systems are synchronized at the end of any checkpoints and the end of the loads.</p> <p>When the parameter is set to true, loads to managed multiple database systems are completely asynchronous. Load data are written to remote nodes asynchronously and, at a load checkpoint end or load end, when closing load data files, Unity does not wait for all the outstanding remote writes of load data to finish. With the True setting, if there is significant network delay or low network bandwidth between the two Unity regions, load clients complete the loads much faster, as loading to a single database system directly, right after completing the loads to the database system at the local region. Unity complete the loads to the database system at the remote region, in the background.</p>
<b>LoadLogoffTimeoutMS</b>	10000			Time (in microseconds) that the endpoint waits for a LOGOFF response from the managed system. By default, it is set to 10000, with a minimum value of 1000 and no maximum value limit. Unity will reject any values underneath the minimum value.
<b>LoadLogonTimeoutMS</b>	60000			Time (in microseconds) that the endpoint waits for a LOGON response from the managed system. By default, it is set to 60000, with a minimum value of 10000 and no maximum value limit. Unity will reject any values underneath the minimum value.
<b>LoadReadFromDiskThreshold</b>	5000	1000-to-infinity		Parameter that controls the queue size before compelling a system to disk during a client load. If a queue for the

Parameter	Default	Scope	Option	Description
				slower system during a client load exceeds this threshold, the system instead loads data from disk for the current checkpoint instead of continuing to load data parcels from memory. Note that this feature (moving a slower system to read from disk instead of in-memory queue) only applies if LoadSpeedThrottle is disabled.
<b>LoadSpeedThrottle</b>	Enabled	Enabled and disabled		Controls whether or not the the load throttling feature is enabled. If enabled, Unity throttles faster systems down to ensure that the slower system does not fall too far behind. By default, slower system can fall behind by 5000 data parcels before the faster systems throttle down. The LoadSpeedThrottleThresholds config parameter controls this threshold.
<b>LoadSpeedThrottleThreshold</b>	Max=5000			Optional configuration that controls both the maximum queue size before the throttle feature kicks in and how large the queue can be before delaying new data parcel queuing. Represented as a maximum minimum string, the max is set to 5000 by default. This means for a client load job, if the slower system queue is at 5000 data parcels or greater, the feature throttles the faster system. New data packets are held in memory until the queue for all systems has dropped below the minimum. The minimum, by default, is set to 4999. When the queue drops below 4999, the system can process new data parcels.
<b>ReadOvedrrideCport</b>	6001	Global	Optional	Port on which Unity listens for sockclient commands. However, sockclient commands are no longer supported.

## Global Configuration Parameters

Parameter	Default (Valid Values)	Description
<b>AutoSelfHealingInterval</b>	0 (0 to infinity)	This is the interval in seconds that the sequencer will attempt to validate, re-activate and resync any tables that have become unrecoverable due to data inconsistencies. (0 to disable).
<b>AutoSelfHealingOptions</b>	0 (0, 1)	This controls whether the self-healing loop will validate (0) or validate and resync (1).
<b>AutoSelfHealingThreads</b>	1 (1 to infinity)	This is the number of concurrent threads used to send table validation and resync requests in the self-healing loop.
<b>AutoTableValidateAttempts</b>	0 (-1 to infinity)	The number of times table validation will be attempted on a table unrecoverable due to data inconsistency. (-1 for unlimited, 0 to disable).
<b>CDMStreamSupportEnabled</b>	true (true/false)	When enabled and supported by the Teradata system, CDM Streaming protocol is used to extract and apply change data. This should only be disabled if a issue occurs when CDM Streaming is used.
<b>CDMMultiStreamSupportEnabled</b>	true (true/false)	When enabled and supported by the Teradata system, CDM Multi-Stream protocol is used to extract and apply change data. This should only be disabled if a issue occurs when CDM Multi-Stream is used.
<b>CDMSessionQueueLimit</b>	1000 (100 to infinity)	The maximum amount (in MB) of queued change data information held in memory for a session before Unity starts throttling the read of incoming CDM data.
<b>CDMMaxDataConnectionsPerGroup</b>	25 1 to 1024	Specifies the maximum number of data connections to connect per replication group.
<b>DataFileAsyncNonBlockingWrite</b>	true (true/false)	By default, data files are distributed across regions asynchronously and limited by network capacity. When a data file size exceeds the network buffer, the write continues or pauses, based on the threshold setting in the <b>DfsAsyncWriteMaxOutstandingMegaBytes</b> parameter. <ul style="list-style-type: none"> <li>Set to false, data file writes pause if the outstanding data on a network buffer to a remote node exceeds the</li> </ul>

Parameter	Default (Valid Values)	Description
		<p>threshold. The data writes continue after the outstanding data is consumed and is under the threshold.</p> <ul style="list-style-type: none"> <li>Set to <code>true</code>, data file writes continue when the amount of outstanding data on the network buffer to a remote node exceeds the threshold. Data writes proceed as fast as the local disk can write. Unity DFS reads the unsent data from the local file copies and sends the data to a remote node after it is within the threshold.</li> </ul> <p><b>Note:</b> Only GSC configures this parameter.</p>
<b>DefaultObjectResyncOption</b>	1 (1, 2, 5)	This controls the default option for resyncing unrecoverable tables. (1 - Never resync, 2 – Resync the entire table, 5 – Resync periodically). The option on individual tables overrides this setting.
<b>DefaultObjectResyncTimeout</b>	300 (30 to infinity)	This sets the default timeout (in seconds) for synchronizing tables using the table sync service.
<b>DefaultObjectValidationTimeout</b>	300 (30 to infinity)	This sets the default timeout (in seconds) for validating tables using the table validation service.
<b>DfsAsyncWriteMaxOutstandingMegaBytes</b>	50 megabytes (1 to infinity)	<p>Maximum amount of outstanding DFS file data on a network buffer for asynchronous writes to a remote node.</p> <p>If reached, data files writes either pause (blocking mode) or continue (non-blocking mode), depending on the <b>DataFileAsyncNonBlockingWrite</b> setting.</p>
<b>DirectSessionIdleTimeout</b>	0 (disabled) (0 to infinity)	The number of seconds a direct session remains idle until Unity closes the matching Unity managed session. This feature is disabled by default because matching Unity managed session for direct sessions is automatically closed on logoff. This parameter is required to close Unity managed session for direct sessions for sessions from a connection pool.
<b>dynamicDFSIOTimeouts</b>	2 (0, 1, 2)	<p>Controls the behavior of read and write timeouts in DFS.</p> <ul style="list-style-type: none"> <li>0 = Feature is disabled</li> </ul>

Parameter	Default (Valid Values)	Description
		<ul style="list-style-type: none"> <li>1 = Feature is enabled for both read and write timeouts in DFS. A DFS read monitors the size of the file and does not timeout if data continues to be written to the file. DFS writes take into account the validity of the file and remote socket when doing writes and waiting for responses.</li> <li>2 = Recommended setting. In addition to the parameters for setting 1, setting 2 allows reads to not timeout because of outstanding writes.</li> </ul>
<b>EnableActivityTrackingObfuscation</b>	false (true/false)	Prevents the collection of sensitive information in support bundles. The sensitive information includes SQL statements and IP addresses to be collected in log files, alerts, backtraces, core files, and so on. This value must be set to <code>true</code> for any of the other <b>EnableActivityTrackingObfuscation*</b> configuration values to take effect.
<b>EnableActivityTrackingObfuscationBacktraces</b>	false (true/false)	Prevents the collection of backtraces in support bundles.
<b>EnableActivityTrackingObfuscationCores</b>	false (true/false)	Prevents the collection of core files in support bundles.
<b>EnableActivityTrackingObfuscationIPAlerts</b>	false (true/false)	Prevents the collection of IP addresses in alerts.
<b>EnableActivityTrackingObfuscationIPLogs</b>	false (true/false)	Prevents the collection of IP addresses in log files.
<b>EnableActivityTrackingObfuscationSQLAlerts</b>	false (true/false)	Prevents the collection of SQL statements in alerts.
<b>EnableActivityTrackingObfuscationSQLLogs</b>	false (true/false)	Prevents the collection of SQL statements in log files.
<b>GatewayMonitorMinTimeout</b>	2 (Process Type or Per Process)	Minimum period of time (in milliseconds) for the Monitor to start for each Gateway. If the Gateway does not respond, it is marked as down, and the timeout is adjusted by the value in <b>GatewayMonitorTimeoutAdjust</b> .
<b>GatewayMonitorTimeoutAdjust</b>	2 (Process Type or Per Process)	Number of milliseconds the period of time allowed for the Gateway handshake increases.

Parameter	Default (Valid Values)	Description
<b>GatewayMonitorMaxTimeout</b>	3 (Process Type or Per Process)	Maximum period of time (in milliseconds) allowed when attempting a handshake with the Gateway.
<b>GlobalDeadlockDetectionInterval</b>	0 (0, 10 seconds to infinity)	The interval in seconds determines how often Unity detects a global deadlock between Unity and the Teradata database. Unity reads the session-level statistics from the Teradata database. The Teradata database updates its session-level statistics at an interval set by the session rate for the Teradata database. Accounting for both rates, the longest time to detect a deadlock between Unity and the Teradata database is the session rate for the database plus the time set in the <b>GlobalDeadlockDetectionInterval</b> . The default value 0 (zero) disables the setting.
<b>OperationPurgeTime</b>	168 hours	Sets the default expiration period in hours after which a completed operation can be purged. To purge operations immediately after they complete, set the expiration time to 0 (zero) hours. The last operation on a valid object, such as a table, or a system is not purged to ensure the persistence of a history for the object.
<b>PassiveRoutingFeature</b>	Enabled*	Enables the passive routing feature. For installations of Unity15.10 or later, the parameter is enabled. To use the passive routing feature, set this parameter to enabled once the entire cluster has been upgraded to 15.10 or later.
<b>SetReadOnlyOnWriteFailure</b>	false (true/ false)	With the default value ( true ), when a recoverable transient error occurs on a request that reads and writes to multiple tables, all the tables being read from are placed in the Read-Only state. All the tables being written to are placed in the Interrupted state. If the feature is turned off ( false ), all the tables being written are placed in an Interrupted state, while the tables being read remain Active.
<b>ValidationUsername</b>	dbc (*)	Database username used to validate tables. Must have the correct select permissions to read tables to validate their content.
*For installations being upgraded from Unity 14.11, the parameter is set to disabled by default.		

# DATABASE ACTIVATE

## Purpose

Activates all unrecoverable tables within the specified database on one or more systems in a Unity configuration.

## Syntax

```
DATABASE ACTIVATE {< database>[ ON tdpid[, tdpid]]} [REASON' reason'];
```

## Parameters

### *database*

Name of database holding tables currently in Data Dictionary.

### *tdpid*

Unique identifier (TDPID) of a Teradata Database system.

### *reason*

Reason why objects on the database need to change.

## Usage Considerations

Use this command after adding a new database using the Data Dictionary Scanner to make all tables within the added database Active.

Only tables that are unrecoverable on all systems can be activated. Any tables that are not unrecoverable on the system they are managed will be skipped.

## Example

The following example shows how to activate a Teradata Database system.

```
unityadmin> database activate dbtest;
Operation Number : 535
Operation Name   : Activate database
User            : admin
User Name       : Main Administration User
Start Time      : 01/21 08:45:37
Updates:
  01/21 08:45:37 [-] Info: Successfully activated table 'dbtest.dblvl1' on
2 systems.
  01/21 08:45:37 [-] Info: Successfully activated table 'dbtest.dblvl2' on
2 systems.
```

```

01/21 08:45:37 [-] Info: Successfully activated table 'dbtest.dblvl3' on
2 systems.
01/21 08:45:37 [-] Info: Successfully activated table 'dbtest.dblvl4' on
2 systems.
01/21 08:45:37 [-] Info: Successfully activated table 'dbtest.dblvl5' on
2 systems.
01/21 08:45:37 [-] Info: Successfully activated table 'dbtest.dblvl6' on
2 systems.
01/21 08:45:37 [-] Info: Successfully activated table 'dbtest.dblvl7' on
2 systems.
01/21 08:45:37 [-] Info: Successfully activated table 'dbtest.dblvl8' on
2 systems.
01/21 08:45:37 [-] Info: Successfully activated table 'dbtest.dblvl9' on
2 systems.
01/21 08:45:37 [-] Info: Successfully activated table 'dbtest.dblvl10' on
2 systems.
01/21 08:45:37 [-] Info: Operation finished
Status      : Finished (1)
Finish Time : 01/21 08:45:37
unityadmin>

```

## DATABASE DEMOTE

### Purpose

Changes the state of all tables in the specified database from Active to Standby.

### Syntax

```
DATABASE DEMOTE{< database>[ ON tdpid[, tdpid]]} [REASON' reason'];
```

### Parameters

#### *database*

Name of database holding tables currently in Unity Data Dictionary.

#### *tdpid*

Unique identifier (TDPID) of a Teradata Database system.

#### *reason*

Description of why objects on the database need to be demoted.



## Usage Considerations

If a system is not specified when running this command, it sets all tables in the database to Standby on ALL systems, which is not usually correct. Specify a system to make the tables Standby only on one system.

## Example

The following example shows how to demote a Teradata Database system with TDPID fast in Standby state.

```
unityadmin> database demote dbtest on db1;
Operation Number : 537
Operation Name   : Demote database
User            : admin
User Name       : Main Administration User
Start Time      : 01/21 08:48:40
Systems:
  [1] db1
Updates:
  01/21 08:48:41 [-] Info: Successful demote of table dbtest.dblvl1 on
1 systems
  01/21 08:48:41 [-] Info: Successful demote of table dbtest.dblvl2 on
1 systems
  01/21 08:48:41 [-] Info: Successful demote of table dbtest.dblvl3 on
1 systems
  01/21 08:48:41 [-] Info: Successful demote of table dbtest.dblvl4 on
1 systems
  01/21 08:48:41 [-] Info: Successful demote of table dbtest.dblvl5 on
1 systems
  01/21 08:48:41 [-] Info: Successful demote of table dbtest.dblvl6 on
1 systems
  01/21 08:48:41 [-] Info: Successful demote of table dbtest.dblvl7 on
1 systems
  01/21 08:48:41 [-] Info: Successful demote of table dbtest.dblvl8 on
1 systems
  01/21 08:48:41 [-] Info: Successful demote of table dbtest.dblvl9 on
1 systems
  01/21 08:48:41 [-] Info: Successful demote of table dbtest.dblvl10 on
1 systems
  01/21 08:48:41 [-] Info: Operation finished
Status          : Finished (1)
Finish Time     : 01/21 08:48:41
Systems:
  [1] db1 - Finished (1)
unityadmin>
```

## DATABASE EXCLUDE ADD

### Purpose

Adds a database to the list of databases to be excluded by the Unity dictionary scanner.

### Syntax

```
DATABASE EXCLUDE ADD {< database> [, database, ...]} ;
```

### Parameters

*database*

Name of database to be added.

## DATABASE EXCLUDE LIST

### Purpose

Lists databases to be excluded by the Unity dictionary scanner.

### Syntax

```
DATABASE EXCLUDE LIST ;
```

## DATABASE EXCLUDE REMOVE

### Purpose

Removes database from the list of databases to be excluded by the Unity dictionary scanner.

### Syntax

```
DATABASE EXCLUDE REMOVE {< database> [, database, ...]} ;
```

### Parameters

*database*

Name of database to be removed.

## DATABASE FREEZE

## Purpose

Changes the state of all tables in the specified database from Active to Read-Only.

## Syntax

```
DATABASE FREEZE, < database> [ ON< tdpid> [, < tdpid>, ...]] [REASON '< reason>'];
```

## Parameters

### *database*

Name of database holding tables currently in Unity Data Dictionary.

### *tdpid*

Unique identifier (TDPID) of a Teradata Database system.

### *reason*

Description of why objects on the database need to change.

## Usage Considerations

If a system is not specified when running this command, it sets all tables in the database to Read-only on ALL systems, which is not usually correct. Specify a system to make the tables Read-Only on one system.

## Example

```
unityadmin> database freeze dbtest;
Operation Number : 1
Operation Name   : Freeze database
User            : admin
User Name       : Main Administration User
Start Time      : 12/21 06:32:17
Updates:
  12/21 06:32:17 [-] Info: Successfully froze table 'dbtest.test0' on 2 systems
  12/21 06:32:17 [-] Info: Successfully froze table 'dbtest.test1' on 2 systems
  12/21 06:32:17 [-] Info: Successfully froze table 'dbtest.test2' on 2 systems
  12/21 06:32:17 [-] Info: Successfully froze table 'dbtest.test3' on 2 systems
  12/21 06:32:17 [-] Info: Successfully froze table 'dbtest.test4' on 2 systems
  12/21 06:32:17 [-] Info: Successfully froze table 'dbtest.test5' on 2 systems
  12/21 06:32:17 [-] Info: Successfully froze table 'dbtest.test6' on 2 systems
  12/21 06:32:17 [-] Info: Successfully froze table 'dbtest.test9' on 2 systems
  12/21 06:32:17 [-] Info: Successfully froze table 'dbtest.test10' on
2 systems
```

# DATABASE HALT

## Purpose

Changes the state of all tables in the specified database from Active to Out-of-Service.

## Syntax

```
DATABASE HALT{< database> [ ON < tdpid> [, < tdpid>, ...]]} [REASON '< reason>'];
```

## Parameters

### *database*

Name of database holding tables currently in Unity Data Dictionary.

### *tdpid*

Unique identifier (TDPID) of a Teradata Database system.

### *reason*

Reason why the objects on the database need to change.

## Usage Considerations

If a system is not specified when running this command, it sets ALL tables in the database to Out-of-Service on all systems, which is not usually correct. Specify a system to make the tables Out-of-Service on one system.

When running this command, Unity holds any new transactions/statements until all inflight transactions close for the object. The **RecoveryCheckpointTimeout** parameter determines the wait time. If in-flight transactions are not closed within the time limit, the operation aborts, an error displays, and the state returns to the state prior to the attempt.

## Example

The following example shows how to halt a Teradata Database system.

```
unityadmin> database halt dbtest on db1;
Operation Number : 541
Operation Name   : Halt database
User            : admin
User Name       : Main Administration User
Start Time      : 01/21 08:50:32
Systems:
  [1] db1
Updates:
```

```

01/21 08:50:32 [-] Info: Requesting mgmt X lock on 'dbtest.dblvl1'
01/21 08:50:32 [-] Info: Mgmt X lock on 'dbtest.dblvl1' granted
01/21 08:50:32 [-] Info: Requesting mgmt X lock on 'dbtest.dblvl2'
01/21 08:50:32 [-] Info: Mgmt X lock on 'dbtest.dblvl2' granted
01/21 08:50:32 [-] Info: Requesting mgmt X lock on 'dbtest.dblvl3'
01/21 08:50:32 [-] Info: Mgmt X lock on 'dbtest.dblvl3' granted
01/21 08:50:32 [-] Info: Successfully halted table 'dbtest.dblvl1' on
1 systems
01/21 08:50:32 [-] Info: Successfully halted table 'dbtest.dblvl2' on
1 systems
01/21 08:50:32 [-] Info: Successfully halted table 'dbtest.dblvl3' on
1 systems
01/21 08:50:32 [-] Info: Successfully halted table 'dbtest.dblvl4' on
1 systems
01/21 08:50:32 [-] Info: Successfully halted table 'dbtest.dblvl5' on
1 systems
01/21 08:50:32 [-] Info: Operation finished
01/21 08:50:32 [-] Info: Releasing mgmt X lock on 'dbtest.dblvl1'
01/21 08:50:32 [-] Info: Releasing mgmt X lock on 'dbtest.dblvl2'
01/21 08:50:32 [-] Info: Releasing mgmt X lock on 'dbtest.dblvl3'
01/21 08:50:32 [-] Info: Releasing mgmt X lock on 'dbtest.dblvl4'
01/21 08:50:32 [-] Info: Releasing mgmt X lock on 'dbtest.dblvl5'
Status      : Finished (1)
Finish Time : 01/21 08:50:32
Systems:
[1] db1 - Finished (1)

```

## DATABASE MANAGE

### Purpose

Changes the managed location of all tables in the specified database to the specified systems.

### Syntax

```
DATABASE MANAGE {< database> [ON < tdpid>[,...]] [REASON '< reason>']}
```

### Example

```

unityadmin> database manage dbtest on db1;
          Table "dbtest.test0" de-activated on :
                    2 (db2)
          Table "dbtest.test1" de-activated on :

```

```

                2 (db2)
Table "dbtest.test2" de-activated on :
                2 (db2)
Table "dbtest.test3" de-activated on :
                2 (db2)
Table "dbtest.test4" de-activated on :
                2 (db2)
Location of all specified object(s) have been modified.

```

## DATABASE PROMOTE

### Purpose

Changes the state of all tables in the specified database from Standby to Active.

### Syntax

```
DATABASE PROMOTE {< database> [ ON tdpid [, tdpid]]} [REASON' reason'];
```

### Parameters

#### *database*

Name of database holding tables currently in Unity Data Dictionary.

#### *tdpid*

Unique identifier (TDPID) of a Teradata Database system.

#### *reason*

Reason why objects need to be Active.

### Usage Considerations

If a system is not specified when running this command, it sets ALL tables in the database to Active on all systems. Specify a system to make the tables Active on one system.

### Example

The following example shows how to promote a Teradata Database system with TDPID fast in Active state.

```

unityadmin> database promote dbtest on db1;
Operation Number : 539
Operation Name   : Promote database
User            : admin
User Name       : Main Administration User
Start Time      : 01/21 08:49:20

```

```

Systems:
  [1] db1
Updates:
  01/21 08:49:20 [-] Info: Successful promote of table dbtest.dblvl1 on
1 systems
  01/21 08:49:20 [-] Info: Successful promote of table dbtest.dblvl2 on
1 systems
  01/21 08:49:20 [-] Info: Successful promote of table dbtest.dblvl3 on
1 systems
  01/21 08:49:20 [-] Info: Successful promote of table dbtest.dblvl4 on
1 systems
  01/21 08:49:20 [-] Info: Successful promote of table dbtest.dblvl5 on
1 systems
  01/21 08:49:20 [-] Info: Successful promote of table dbtest.dblvl6 on
1 systems
  01/21 08:49:20 [-] Info: Successful promote of table dbtest.dblvl7 on
1 systems
  01/21 08:49:20 [-] Info: Successful promote of table dbtest.dblvl8 on
1 systems
  01/21 08:49:20 [-] Info: Successful promote of table dbtest.dblvl9 on
1 systems
  01/21 08:49:20 [-] Info: Successful promote of table dbtest.dblvl10 on
1 systems
  01/21 08:49:20 [-] Info: Operation finished
Status      : Finished (1)
Finish Time : 01/21 08:49:20
Systems:
  [1] db1 - Finished (1)

```

## DATABASE RECOVER

### Purpose

Changes the state of all tables in the specified database from Interrupted, Read-only or Out-of-service to Active.

### Syntax

```
DATABASE RECOVER {< database> [ ON < tdpid> [, < tdpid>, ...]]} [REASON '< reason>'];
```

## Parameters

### *database*

Name of database holding tables currently in Unity Data Dictionary.

### *tdpid*

Unique identifier (TDPID) of a Teradata Database system.

### *reason*

Reason why the objects on the database need to change.

## Usage Considerations

This command sets the state of all tables Read-only or Out-of-service in the specified database and systems to Restore. If the table cannot recover successfully due to a transient error, then the table remains in Interrupted state, which raises an alert.

After resolving the error, retry the recovery. If Unity cannot successfully recover the table due to data consistency issues, then the table enters Unrecoverable state. The command operation replays the Recovery Log to bring the Teradata Database server or table from Out of Service, Interrupted, or Read Only state to Active state without requiring manual intervention to restore the data. If the recovery operation is unsuccessful, the Teradata Database system or table becomes Interrupted or Unrecoverable based on the error returned.

Running this operation on a Teradata Database system places each table in that Teradata Database system from Out of Service or Interrupted into Restore state. When attempted at the system level, the system completes recovery within a short period while tables on the system continue recovery at an individual level. As each table completes recovery successfully, it becomes ACTIVE and accessible for client transactions.

## Example

The following example shows how to recover a Teradata Database system with TDPID.

```
unityadmin> database recover dbtest on db1;
Operation Number : 543
Operation Name   : Recover database
User            : admin
User Name       : Main Administration User
Start Time      : 01/21 08:53:20
Systems:
  [1] db1
Updates:
  01/21 08:53:20 [-] Info: Recovering table dbtest.dblvl1
  01/21 08:53:20 [-] Info: Recovering table dbtest.dblvl2
  01/21 08:53:20 [-] Info: Recovering table dbtest.dblvl3
```



```

01/21 08:53:20 [-] Info: Recovering table dbtest.dblvl4
01/21 08:53:20 [-] Info: Recovering table dbtest.dblvl5
01/21 08:53:20 [-] Info: Recovering table dbtest.dblvl6
01/21 08:53:20 [-] Info: Recovering table dbtest.dblvl7
01/21 08:53:20 [-] Info: Recovering table dbtest.dblvl8
01/21 08:53:20 [-] Info: Recovering table dbtest.dblvl9
01/21 08:53:20 [-] Info: Recovering table dbtest.dblvl10
01/21 08:53:20 [-] Info: Successfully started recovery for table on 1 systems
01/21 08:53:20 [-] Info: Operation finished
Status      : Finished (1)
Finish Time : 01/21 08:53:20
Systems:
  [1] db1 - Finished (1)

```

## DATABASE RESYNC

### Purpose

Synchronizes all the tables in the specified database from the source system to the destination system.

For each table within the database, if the replication and table sync options are set, they are used to determine if the table should be synchronized and how long that synchronization is allowed to take.

The synchronization may take a long time depending on the size of the tables or the transfer rate between the systems.

### Syntax

```
DATABASE RESYNC {< database>[ FROM < tdpid1> TO < tdpid2>][ TIMEOUT '< timeout (seconds)>']}] ;
```

### Parameters

#### *database*

Name of database holding tables to synchronize.

#### *tdpid1*

Unique identifier (TDPID) of a source Teradata Database system.

#### *tdpid2*

Unique identifier (TDPID) of a target Teradata Database system.

#### **TIMEOUT** *seconds*

Specifies the timeout value, in seconds.

**Example**

```

unityadmin> database resync db1 from sys1 to sys2;
Operation Number : 3
Operation Name   : Resyncing Object
User            : admin
User Name       : Main Administration User
Start Time      : 05/01 08:53:55
Updates:
    05/01 08:53:55 [-] Info: Resyncing table db1.table1 from [1] sys1 to [2]
    sys2 using timeout 150
...
    05/01 08:53:55 [-] Info: 2461 tables in the database were checked to see if
they could be synced.
    05/01 08:53:55 [-] Info: 99 tables were skipped, because they are set to
NOT sync.
    05/01 08:53:55 [-] Info: Syncing was attempted on 2362 tables.
    05/01 08:53:55 [-] Info: 2361 tables were successfully synced.
    05/01 08:53:55 [-] Info: Operation finished
Status         : Finished (1)
Finish Time    : 05/01 08:53:55
unityadmin>

```

## DATABASE SET SYNC USER

**Purpose**

Sets the user name for syncing tables in the specified database using the table sync service.

**Syntax**

```
DATABASE SET SYNC USER {< database> < sync user name>;}
```

**Parameters*****database***

Name of the target database.

***sync user name***

The name of the user to execute sync SQL statements.

**Example**

```
unityadmin> DATABASE SET SYNC USER db0 db1;
The syncUser metadata for tables in database db0 has been set to db1.
```

## DICTIONARY EXPORT

**Purpose**

Exports the catalog to a directory based on database or object criteria.

**Syntax**

```
DICTIONARY EXPORT { [DATABASES < databasename>] | [OBJECTS < objectname> | id: objecteid]} TO
<exportpath> ;
```

**Parameters**

***databasename* | *objectname***

Name of one or multiple databases or specific database objects to export.

***exportpath***

Name of the file path to which Unity exports the specified object information.

**Usage Considerations**

Use a comma to separate multiple database names or object names in a list. You must specify the directory location where you want the exported dictionary to be stored.

**Example**

```
unityadmin> DICTIONARY EXPORT DATABASES db0, db1 TO /exported_dictionaries;
```

## DICTIONARY IMPORT

**Purpose**

Imports object definitions from an export file created using the `DICTIONARY EXPORT` command.

This command validates imported objects against Teradata systems managed by Unity. It marks objects not matching underlying systems as invalid, and disallows dictionary deployment to the managed Unity Data Dictionary until the issues are resolved.

**Syntax**

```
DICTIONARY IMPORT FROM path [FORCE UPDATE] ;
```

**Parameters***path*

Specifies the path of the location to import from.

**Example**

```
unityadmin> DICTIONARY IMPORT FROM /exported_dictionaries FORCE UPDATE;
```

## DICTIONARY SCAN

**Purpose**

Scans Teradata systems managed by Unity for new table objects and for changes in table objects that apply to the Data Dictionary you specify.

**Syntax**

```
DICTIONARY SCAN { ALL [ tdpid1, tdpid2,..., tdpidN] } {[IN db1, db2.. , dbN] | OBJECT objectname} ;
```

**Parameters***tdpid1*, *tdpid2*, *tdpidN*

Unique identifier (TDPID) of a Teradata Database system.

IN *db1*,*db2*,*dbN*

Use to specify the Teradata Databases to scan within the Teradata systems managed by Unity. Running DICTIONARY SCAN without a database list scans all Teradata Databases that exist in the specified Data Dictionary (*dictionaryname*). If you are scanning a new Data Dictionary and exclude this parameter, Unity scans all Teradata Databases that it manages.

*object*

Name of a specific database object in the Teradata managed systems to scan.

**Usage Considerations**

If any Teradata system or Teradata Database you specify for scanning is inaccessible, the DICTIONARY SCAN operation fails and Unity raises an alert.

This command creates a scan log on the Unity server in `/var/opt/teradata/unity/logs` managed by the Active Sequencer. The file name is `scanner.log`. This log includes a full list of table objects added to the managed Unity Data Dictionary and also table objects rejected due to failure to comply with one or more object-matching criteria. For each rejected table object, the log lists and describes the mismatched elements within the object.

## ENDPOINT ALLOW NEW CONNECTIONS

### Purpose

Allows new session connections to an Endpoint. The command requires that you specify the name of the Endpoint to which you want new connections allowed. This command operation is asynchronous.

### Syntax

```
ENDPOINT ALLOW NEW CONNECTIONS endpoint_name ;
```

### Parameters

#### *endpointname*

Name of the Endpoint process.

### Usage Considerations

The `ENDPOINT ALLOW NEW CONNECTIONS` instructs the specified Endpoint to listen for new connections. You can use the `ENDPOINT STATUS` command to verify the status of global listeners on the Endpoint after you allow connections. When you allow a new connection, the specified Endpoint listens for connections on all default listeners and any SIP listeners currently running on one or more regions. To prevent an Endpoint from receiving new connections, use the `ENDPOINT DISALLOW NEW CONNECTIONS` command.

### Example

This example command requests that the Endpoint in Region1 listen for new connections.

```
unityadmin> ENDPOINT ALLOW NEW CONNECTIONS region1_ept;
Request successfully sent to endpoint
Use the 'endpoint status' command to check the endpoint's status
unityadmin> ENDPOINT STATUS;
-----
Endpoint   : region1_ept
Connected  : yes
Global listen status : Listening allowed
Listeners :
  Host:Port      : any:1025
  Is SIP         : no
```

```

Associated region : region1 (1)
State (Current)   : Listening
State (Wanted)    : Listening

```

## ENDPOINT DISALLOW NEW CONNECTIONS

### Purpose

Prevents a new session connection to an Endpoint. The command requires that you specify the name of the Endpoint to which new connections are disallowed. This command operation is asynchronous.

### Syntax

```
ENDPOINT DISALLOW NEW CONNECTIONS endpoint_name ;
```

### Parameters

#### *endpointname*

Name of the Endpoint process.

### Usage Considerations

This ENDPOINT DISALLOW NEW CONNECTIONS command instructs the specified Endpoint to stop listening for new connections. You can use the ENDPOINT STATUS command to view the status of global listeners on an Endpoint before disallowing connections. When you disallow connections, the Endpoint stops listening for default listeners and any defined SIP listeners. The command does not affect connections that are already open. When you are ready, use the ENDPOINT ALLOW NEW CONNECTIONS command to change the Endpoint status to receiving new connections again.

### Example

This example command prevents any new sessions on the Endpoint named region1\_ept.

```

unityadmin> ENDPOINT DISALLOW NEW CONNECTIONS region1_ept;
Request successfully sent to endpoint
Use the 'endpoint status' command to check the endpoint's status
unityadmin> ENDPOINT STATUS;
-----
Endpoint   : region1_ept
Connected  : yes
Global listen status : Listening disallowed
Listeners :
  Host:Port      : any:1025
  Is SIP         : no
  Associated region : region1 (1)

```

```
State (Current)    : Not Listening
State (Wanted)     : Not Listening
```

## ERRORLIST DELETE

### Purpose

Removes the indicated error list profile.

### Syntax

```
ERRORLIST DELETE ErrorListName ;
```

### Parameters

#### *errorlistname*

Unique TDPID that identifies the Unity error list name.

### Example

```
unityadmin> ERRORLIST DELETE myErrorList1;
```

## ERRORLIST EXPORT

### Purpose

Exports the specified error list into the specified file which then can be edited and imported.

### Syntax

```
ERRORLIST EXPORT ErrorListName filename ;
```

### Parameters

#### *errorlistname*

Unique TDPID that identifies the Unity error list name.

#### *filename*

Name of the export details file that you create using the ERRORLIST EXPORT command.

If the filename is specified without a full path, the exported error list file is created in the directory from which unityadmin was launched.

**Example**

```
unityadmin> ERRORLIST EXPORT myErrorList1 myExportedList.txt;
```

## ERRORLIST LIST

**Purpose**

Provides connection, region, and status of the listeners (defined using the CONFIG UPDATE command) on each Endpoint.

**Syntax**

```
ERRORLIST LIST ErrorListName ;
```

**Parameters*****errorlistname***

Unique TDPID that identifies the Unity error list name.

**Example**

```
unityadmin> ERRORLIST LIST;
DefaultErrorList
```

## ERRORLIST UPDATE

**Purpose**

Creates or updates a routing error list by specific codes.

**Syntax**

```
ERRORLIST UPDATE <ErrorListName> [RESUBMIT | RETRY | RETRY THEN RESUBMIT | EXIT | REMOVE
{ <code> | <start>-<end> } [ 'comment' ] ] ;
```

**Parameters*****errorlistname***

Unique TDPID that identifies the Unity error list name.

EXIT



**file**

Filename of the error list to import.

The format must match the format of the file created using `ERRORLIST EXPORT`.

If the full path is not specified, the file is read in the directory from which unityadmin was launched.

**filename**

Filename of the error file to import. The format of this file must match the original error code file created using sockclient.

The format of this file is: `<error code> [# <comment>]`

Example of a RESUBMIT file:

```
1068 # Resubmittable error due to lost connection to underlying system
2507
4505 # Resubmittable error due to nonexistent session on
underlying system
```

**code**

The error code to update.

**IMPORT**

Imports Errorlist.

**RESUBMIT**

Tries the query on another system.

**RETRY THEN RESUBMIT**

Combination of retry and resubmit.

**EXIT**

Terminates the session.

**REMOVE**

Removes the query.

**start**

Start of the error code range when updating a rule that affects a range of error codes.

**end**

End of the error code range when updating a rule that affects a range of error codes.

## Examples

At the unityadmin prompt:

Input	Description
ERRORLIST UPDATE from_QD IMPORT RESUBMIT err_pref.txt EXIT err_ exit.txt;	Import an existing rule created using an earlier version of sockclient into Unity.
ERRORLIST UPDATE myErrorList1 IMPORT myEditedList.txt;	Import changes that were made to the error list file that was exported using "ERRORLIST EXPORT."
ERRORLIST UPDATE myErrorList1 RESUBMIT 2543-2545 'Operation not allowed error';	Add a new failover rule for a range of error code to an existing error list profile.
ERRORLIST UPDATE myErrorList1 REMOVE 2507;	Remove a rule from an existing error list profile.

## LOAD LIST

### Purpose

Displays information on all client loads that are currently loading data through Unity.

### Syntax

```
LOAD LIST [LSN] ;
```

### Parameters

#### LSN [optional]

Optional LSN command to get more details about a specific client job that is currently in progress.

### Examples

Lists all current loads.

```
unityadmin> Load List;
Client Loads In Progress: 2
-----
LSN:      1000
```

Table: testDB.testtable3

```
Transaction ID           : 61
Latest client checkpoint : 21
Systems participating in load :
  managed1: processing checkpoints/data (last checkpoint taken: 21)
  managed2: processing checkpoints/data (last checkpoint taken: 20)
-----
LSN:      1004
```

Table: testDB.testtable2

```
Transaction ID           : 85
Latest client checkpoint : 3
Systems participating in load :
  managed1: processing checkpoints/data (last checkpoint taken: 3)
  managed2: processing checkpoints/data (last checkpoint taken: 3)
```

Lists specific LSN

```
unityadmin> LOAD LIST 1004;
```

```
-----
LSN:      1004
```

Table: testDB.testtable2

Error tables: testDB.testtable2\_error\_a testDB.testtable2\_error\_b

```
Transaction ID           : 85
Load Type                 : fastload
Latest client checkpoint : 11
Endpoints participating in load:
  region1_ept
  region2_ept

Load Size                  : 990000 bytes
Load rows applied          : 110000 rows applied
```

System managed1:

2 data connections

State: processing checkpoints/data (last checkpoint taken: 11)

Approximate data loaded: 990000 bytes

Approximate rows applied: 110000

System managed2:

2 data connections

```
State: processing checkpoints/data (last checkpoint taken: 10)
Approximate data loaded: 900000 bytes
Approximate rows applied: 100000
```

## LOAD INFO RECOVERY

### Purpose

Displays information on all loads that are currently being recovered as well as loads that are in the recovery queue (and ready to be recovered once preceding requests have completed).

- 1 - Session id, tid and load type [MLOAD session 1000 tid 1234]
- 2 - System being recovered
- 3 - Checkpoints applied
- 4 - Total checkpoints
- 5 - MLoad: Will exec be run

### Syntax

```
LOAD RECOVERY LIST [LSN]
```

### Parameters

#### LSN [optional]

Optional LSN command to get more details about a specific load job in recovery.

### Examples

Lists all recovering loads.

```
unityadmin> load recovery list;
Recovery Loads In Progress: 1
-----
LSN:      1003
Table: testDB.mloadA
Table: testDB.mloadB

Transaction ID           : 286
Latest checkpoint applied : 0
System being recovered   :
managed2: processing checkpoints/data (last checkpoint taken: 0)
```

Lists specific LSN.

```

unityadmin> load recovery list 1003;
Recovery Loads In Progress: 1
-----
LSN:      1003

Table: testDB.mloadA
  Error tables: testDB.mloadA_et1 testDB.mloadA_et2
  Work table: testDB.mloadA_wt
Table: testDB.mloadB
  Error tables: testDB.mloadB_et1 testDB.mloadB_et2
  Work table: testDB.mloadB_wt
Transaction ID           : 286
Load Type                 : mload
Latest checkpoint applied : 0
System managed2:
  2 data connections
  State: processing checkpoints/data (last checkpoint taken: 0)
  Approximate data loaded: 110 bytes
  Approximate rows applied: 10

```

## LOAD RECOVERY LIST

### Purpose

Returns a list of recovery loads in progress or ready to start.

### Syntax

```
LOAD RECOVERY LIST LSN ;
```

### Parameters

#### *LSN*

Load session number. Enter an optional session number to see the details for the specified session.

### Examples

This shows all recovery loads in progress.

```

unityadmin> load recovery list;
Recovery Loads In Progress: 1
-----
LSN:      1003

```

```
Table: testDB.mloadA
```

```
Table: testDB.mloadB
```

```
Transaction ID           : 286
Latest checkpoint applied : 0
System being recovered    :
  managed2: processing checkpoints/data (last checkpoint taken: 0)
```

This shows load session number 1003 in progress.

```
unityadmin> load recovery list 1003;
Recovery Loads In Progress: 1
-----
LSN:      1003

Table: testDB.mloadA
  Error tables: testDB.mloadA_et1 testDB.mloadA_et2
  Work table: testDB.mloadA_wt
Table: testDB.mloadB
  Error tables: testDB.mloadB_et1 testDB.mloadB_et2
  Work table: testDB.mloadB_wt
Transaction ID           : 286
Load Type                 : mload
Latest checkpoint applied : 0
System managed2:
  2 data connections
  State: processing checkpoints/data (last checkpoint taken: 0)
  Approximate data loaded: 110 bytes
  Approximate rows applied: 10
```

## OBJECT ACTIVATE

### Purpose

Activates the specified object on the specified systems in the cluster.

### Syntax

```
OBJECT ACTIVATE { tablename [ON tdpid [, tdpid, ...]] [ REASON ' reason' ] [SYNCED FROM tdpid] ;
```

## Parameters

### ON

Change a table on a specific Teradata Database system.

### *tdpid*

Unique identifier (TDPID) of a Teradata Database system.

### REASON *reason*

User description of why the object is having its state changed.

### *tablename*

The fully qualified table name if you are attempting to change the state of a table.

## SYNCED FROM

An object in an *Unrecoverable* state on one of the managed systems can be activated if the data has been manually synced up between managed systems that are in *Active*, *Out-of-service*, or *Read-only* states. During OBJECT ACTIVATE using the SYNCED FROM clause, the source table must be in either *Read-Only* or *Out of Service*. The object becomes *Out of Service* on the managed system where it was *Unrecoverable*, and then can be recovered to *Active*. This allows you to use an online ARC to minimize the write-outage on the source table.

## Usage Considerations

The OBJECT ACTIVATE command changes all specified Teradata Database systems and the table objects on those systems to an *Active* state. The specified table objects must be in an *Unrecoverable* state before being activated. The data in the specified table objects must be synchronized across the specified Teradata Database systems to successfully be changed to an *Active* state. The OBJECT ACTIVATE command fails if data is inconsistent or if table objects are not in the *Unrecoverable* state prior to running the command.

## Example

```
unityadmin> OBJECT ACTIVATE  upgrade_test.active_table;
Operation Number : 29
Operation Name   : Activating Table
User            : admin
User Name       : Main Administration User
Start Time      : 12/24 14:06:49
Systems:
  [1] sd117772
  [2] sd117773
Updates:
```

```

12/24 14:06:49 [-] Info: Activating table upgrade_test.active_table
12/24 14:06:49 [-] Info: Successfully activated table on 2 systems
12/24 14:06:49 [-] Info: Operation finished
Status      : Finished (1)
Finish Time : 12/24 14:06:49
Systems:
  [1] sdll7772 - Finished (1)
  [2] sdll7773 - Finished (1)
unityadmin>

```

## OBJECT DEACTIVATE

### Purpose

Changes state of a table object to the Unrecoverable state. The table object must be managed by Unity.

### Syntax

```
OBJECT DEACTIVATE { tablename [ON tdpid [, tdpid, ...]] [ REASON reason] ;
```

### Parameters

#### ON

Change a table on a specific Teradata Database system.

#### *tdpid*

Unique identifier (TDPID) of a Teradata Database system.

#### **REASON** *reason*

User description of why the object is having its state changed.

#### *tablename*

The fully qualified table name if you are attempting to change the state of a table.

### Usage Considerations

Unless you specify a table object name, running this command sets all table objects to the Unrecoverable state.

---

#### Note:

The table object becomes permanently inaccessible after running this operation.

---



## Example

The following changes a table `temporal_user.mc_tab6` to the Unrecoverable state across all managed Teradata Database systems.

```
unityadmin> OBJECT DEACTIVATE temporal_user.mc_tab6;
The request is currently processing as operation number 2.

Operation Number : 2
Operation Name   : Deactivating Table
User            : admin
User Name       : Main Administration User
Progress (%)     : 100
Status          : Finished (1)
Start Time      : 01/17 10:45:46
Finish Time     : 01/17 10:45:47
Systems:
  [1] bltor - Finished (1)
  [2] fast - Finished (1)
Updates:
  01/17 10:45:47 [-] Info: Deactivating table temporal_user.mc_tab6
  01/17 10:45:47 [-] Info: Successfully deactivated table on 2 systems
  01/17 10:45:47 [-] Info: Operation finished
```

## OBJECT FREEZE

### Purpose

Changes state of a table object to the Read-Only state. The table must be managed by Unity.

### Syntax

```
OBJECT FREEZE { tablename [ON tdpid [, tdpid, ...]] [ REASON reason] ;
```

### Parameters

#### ON

Change a table on a specific Teradata Database system.

#### *tdpid*

Unique identifier (TDPID) of a Teradata Database system.

**REASON *reason***

User description of why the object is having its state changed.

***tablename***

The fully qualified table name if you are attempting to change the state of a table.

**Usage Considerations**

- Specify a single table object to mark it Read-Only.
- When a Unity managed table object is frozen across multiple systems, the freeze process completes on each system as the table object on that system becomes current with the freeze point. After a successful operation, the data is the same on each copy of the table object across the systems on which it was frozen.
- If you are making a state change for a Unity managed table object, then the table object on all Teradata Database systems becomes Read-Only. If you are making a state change for a Unity managed table object on a specific Teradata Database system, only the table object on that Teradata Database system becomes Read-Only.

**Example**

The following example demonstrates placing a table `table4` from a Teradata Database `temporal_user` in Read-Only state.

```
unityadmin> OBJECT FREEZE temporal_user.table4
The request is currently processing as operation number 9.

Operation Number : 9
Operation Name   : Freezing Table
User            : admin
User Name       : Main Administration User
Progress (%)     : 100
Status          : Finished (1)
Start Time      : 01/14 16:35:49
Finish Time     : 01/14 16:35:50
Systems:
  [1] bltor - Finished (1)
  [2] fast - Finished (1)
Updates:
  01/14 16:35:50 [-] Info: Freezing table temporal_user.table4
  01/14 16:35:50 [-] Info: Requesting mgmt R lock on 'temporal_user.table4'
  01/14 16:35:50 [-] Info: Mgmt R lock on 'temporal_user.table4' granted
  01/14 16:35:50 [-] Info: Releasing mgmt R lock on 'temporal_user.table4'
```

```
01/14 16:35:50 [-] Info: Successfully froze table on 2 systems
01/14 16:35:50 [-] Info: Operation finished
```

## OBJECT HALT

### Purpose

Changes state of a table object to the Out-of-Service state. The table must be managed by Unity.

### Syntax

```
OBJECT HALT { tablename [ON tdpid [, tdpid, ...]] [ REASON reason] ;
```

### Parameters

#### ON

Change a table on a specific Teradata Database system.

#### *tdpid*

Unique identifier (TDPID) of a Teradata Database system.

#### REASON *reason*

User description of why the object is having its state changed.

#### *tablename*

The fully qualified table name if you are attempting to change the state of a table.

### Usage Considerations

Restricts an object state change to an object on a specific Teradata Database system. If making a state change for an object on all the Teradata Database systems then the object has the same data across all Teradata Database systems when this operation completes.

When running this command, Teradata Unity holds any new transactions/statements until all in-flight transactions close for the object. **RecoveryCheckpointTimeout** determines the duration of wait time. If in-flight transactions are not closed within this time limit, the operation aborts. An error displays, and the state returns to its state prior to the attempt.

If a multi-object Write runs, and an object is in Out of Service or Interrupted state, then all associated objects in the Write automatically become Out of Service or Interrupted.

### Example

The following example demonstrates placing an object `t4_cp` from a Teradata Database `cp_tpp004u` in Out-of-Service state on the Teradata Database system `cricket`.

```
unityadmin> OBJECT HALT cp_tpp004u.t4_cp ON cricket;
```

## OBJECT LIST

### Purpose

Lists all Unity-managed table objects in the state you specify.

### Syntax

```
OBJECT LIST [ state] [ ,state,... ] [ ON tdpid [,tdpid,...]] ;
```

### Parameters

The following states are available:

#### ACTIVE

Lists table objects in *Active* state.

#### INTERRUPTED

Lists table objects in *Interrupted* state.

#### OOS

Lists table objects in *Out of Service* state.

#### READONLY

Lists table objects in *Read Only* state.

#### RESTORE

Lists table objects in *Restore* state.

#### STANDBY

Lists table objects in *Standby* state.

#### UNRECOVERABLE

Lists table objects in *Unrecoverable* state.

#### *tdpid*

Unique identifier of the Teradata Database system.

## Example

The following example lists table objects in the *Active* state on the Teradata Database systems named TDsysA and TDsysB.

```
unityadmin> OBJECT LIST ACTIVE;
State of orders.customers on preferred system 1 (TDsysA) active
State of orders.customers on system 2 (TDsysB) active
State of orders.employee on preferred system 1 (TDsysA) active
State of orders.employee on system 2 (TDsysB) active
State of orders.location on system 1 (TDsysA) active
State of orders.location on system 2 (TDsysB) active
```

### Note:

The word 'preferred' only appears on an output line that shows the preferred system for the table.

Tables that do not have a preferred system, as in the `orders.location` output in this example, do not have the 'preferred' specification.

## OBJECT MANAGE

### Purpose

Sets the systems to manage the object.

### Syntax

```
OBJECT MANAGE <object name> ON <tdpid> [, <tdpid>,...]
```

### Parameters

#### *object\_name*

Fully qualified name of object

#### *tdpid*

Unique identifier of a Teradata Database system.

### Usage Considerations

This command allows you to change the definition of the object within Unity to a subset of systems. This does not remove the object from the system. It tells Unity where the object should be managed.

**Example**

```
unityadmin> OBJECT MANAGE db1.obj8080 ON sys1;
```

## OBJECT RECOVER

**Purpose**

Replays the Recovery Log to bring the Unity managed table object from an Out of Service, Interrupted, or Read Only state to Active state without requiring manual intervention by an administrator to restore the data. If the recovery operation is unsuccessful, the table object changes to the Interrupted or Unrecoverable state depending on the error.

**Syntax**

```
OBJECT RECOVER { tablename [,tablename,...] [ ON tdpid [, tdpid, ...]] } [ REASON reason] ;
```

**Parameters*****tablename***

The fully qualified table name if you are attempting to change the state of a table. You may enter multiple table names.

***tdpid***

Unique identifier (TDPID) of a Teradata Database system.

**REASON**

Reason for clearing. Use single quotation marks when the reason is more than one word.

**Usage Considerations**

- To restore a Unity managed table object to *Active* state, you must synchronize the data outside of Unity or if all writes were done through Unity, synchronize the table through recovery processing.
- If the Unity managed table object cannot recover successfully due to a transient error, then the table object remains in Interrupted state, which raises an alert. After resolving the error, retry the recovery.
- If Unity cannot successfully recover the Unity-managed table object due to data consistency issues, then the table object enters *Unrecoverable* state.

**Example**

```
unityadmin> OBJECT RECOVER db1.obj9333 ON sys1;
```

## OBJECT REMOVE

## Purpose

Removes an object for immediate reclaim after deactivation, which removes the object from the data dictionary.

## Syntax

**OBJECT REMOVE** <*object\_name*>

## Example

```
unityadmin> object remove dbtest.test199;
Operation Number : 5
Operation Name   : Removing Object
User            : admin
User Name       : Main Administration User
Start Time      : 12/21 06:38:09
Updates:
  12/21 06:38:10 [-] Info: No references found for Object: dbtest.test199
  12/21 06:38:10 [-] Info: Marking provided item: dbtest.test199 for
immediate reclaim
  12/21 06:38:10 [-] Info: Successfully marked object dbtest.test199 for
immediate reclaim
  12/21 06:38:10 [-] Info: Operation finished
```

# OBJECT RESYNC

## Purpose

Synchronizes the specified table from the source system to the destination system.

## Syntax

**OBJECT RESYNC** {< *tablename* | *id:object id*>} [**FROM** *tdpid1*] [**TO** *tdpid2*] [**TIMEOUT** '< *timeout* (seconds)>'] ;

## Parameters

### *tablename*

Fully qualified name of the target object.

### *object id*

Managed object id of the target object.

***tdpid1***

Unique identifier (TDPID) of a source Teradata Database system.

***tdpid2***

Unique identifier (TDPID) of a target Teradata Database system.

***timeout (seconds)***

The timeout, in seconds, to use for the operation.

**Example**

```
unityadmin> OBJECT RESYNC db0.t1000 FROM sdt29630 TO sdt29631 TIMEOUT 60;
Operation Number : 3
Operation Name   : Resyncing Object
User            : admin
User Name       : Main Administration User
Start Time      : 07/22 10:29:24
Updates:
  07/22 10:29:24 [-] Info: Resyncing table db0.t1000 from sdt29630 to sdt29631
using timeout 60
  07/22 10:29:26 [-] Info: Activating table on target system ( setting
to Active )
  07/22 10:29:26 [-] Info: Successfully activated the table on 2 systems
  07/22 10:29:26 [-] Info: Operation finished
  07/22 10:29:26 [-] Info: Table resync completed
Status          : Finished (1)
Finish Time     : 07/22 10:29:26
```

## OBJECT SET DETERMINISM

**Purpose**

Set the override level for the determinism of an object.

**Syntax**

```
OBJECT SET DETERMINISM < object_name > [NO OVERRIDE | FORCE DETERMINISTIC |
FORCE NONDETERMINISTIC]
```

**Parameters****FORCE DETERMINISTIC**

Treats an object as deterministic



**FORCE NONDETERMINISTIC**

Treats an object as nondeterministic

**NO OVERRIDE**

Allows Unity to decide the object determinism

**objectname**

Fully qualified name of the target object

**Example**

At the unityadmin prompt:

Input	Result
OBJECT SET DETERMINISM databasename.objectname FORCE NONDETERMINISTIC;	Forces Unity to use CDM replication for SQL statements which <ul style="list-style-type: none"> <li>• Perform a write; and</li> <li>• Reference the named object</li> </ul>
OBJECT SET DETERMINISM databasename.objectname FORCE DETERMINISTIC;	Forces Unity to not use CDM replication due to the named object being referenced in the SQL statement. This does not guarantee that the SQL statement will not result in CDM replication due to the determinism level of other objects referenced within the same SQL statement.
OBJECT SET DETERMINISM databasename.objectname NO OVERRIDE;	Allows Unity to decide which replication type to use in regards to the named object being referenced within a SQL statement.

## OBJECT SET REPLICATION

**Purpose**

Defines if a table is replicated automatically using Unity self-healing feature and DATABASE SYNC command.

**Syntax**

```
OBJECT SET REPLICATION {< objectname>} [DEFAULT | NEVER | IMMEDIATE] ;
```

**Parameters****objectname**

Fully qualified name of the target object.

**DEFAULT | NEVER | IMMEDIATE**

With NEVER, the table is not replicated.

With IMMEDIATE, the table is not immediately replicated from a source system to a target system.

With DEFAULT, the table is replicated based on the value of the configuration setting **DefaultObjectResyncOption**:

- 1 - Never replicate the table.
- 2 - Resync the table immediately.

**Examples**

The replication option for [4143] db1.table1 has been set to IMMEDIATE.

```
unityadmin> OBJECT SET REPLICATION db1.table1 IMMEDIATE;
```

The replication option for [4143] db1.table1 has been set to NEVER.

```
unityadmin> OBJECT SET REPLICATION db1.table2 NEVER;
```

## OBJECT SET SYNC USER

**Purpose**

Sets the user name for syncing this table using the table sync service.

**Syntax**

```
OBJECT SET SYNC USER{< objectname>} [ '< sync user name>' ] ;
```

**Parameters*****objectname***

Fully qualified name of the target object.

***sync user name***

The name of the user to execute sync SQL statements.

**Example**

```
unityadmin> OBJECT SET SYNC USER db0.t0 db1;
The sync user for [1001] db0.t0 has been set to db1.
```

## OBJECT SET SYNC TIMEOUT

### Purpose

Sets the timeout for syncing this table using the table sync service.

### Syntax

```
OBJECT SET SYNC TIMEOUT{< objectname>} [ '< sync timeout (seconds)>' ] ;
```

### Parameters

#### *objectname*

Fully qualified name of the target object.

#### *sync timeout (seconds)*

Specifies the sync timeout value, in seconds.

### Example

```
unityadmin> OBJECT SET SYNC TIMEOUT db1.table1 30;
The sync timeout for [4143]db1.table1 has been set to 30.
```

## OBJECT SET SYNC WHERE CLAUSE

### Purpose

Sets the sync WHERE clause for a table.

### Syntax

```
OBJECT SET SYNC WHERE CLAUSE {< objectname>} [ '< whereclause >' ] ;
```

### Parameters

#### *objectname*

Fully qualified name of the target object.

#### *whereclause*

WHERE clause to be used with the table sync SQL.

**Example**

```
unityadmin> OBJECT SET SYNC WHERE CLAUSE db0.t0 "WHERE c0 > 1";
The sync WHERE clause for table 'db0.t0' has successfully been set to: 'WHERE c0 > 1'.
```

## OBJECT SET VALIDATION WHERE CLAUSE

**Purpose**

Sets the validation WHERE clause for a table.

**Syntax**

```
OBJECT SET VALIDATION WHERE CLAUSE{< objectname>} ['< whereclause>'] ;
```

**Parameters*****objectname***

Fully qualified name of the target object.

***whereclause***

WHERE clause to be used with the table validation query.

**Example**

```
unityadmin> OBJECT SET VALIDATION WHERE CLAUSE db0.t0 "WHERE c0 > 1";
The validation WHERE clause for table 'db0.t0' has successfully been set to:
'WHERE c0 > 1'.
```

## OBJECT SHOW

**Purpose**

Shows the system the object is being managed on.

**Syntax**

```
OBJECT SHOW < object_name> MANAGED ON
```

**Parameters****<object\_name>**

Fully qualified name of object

## Usage Considerations

Use to determine the systems that are managing an object.

## Example

```
unityadmin> OBJECT SHOW db1.obj8080 MANAGED ON sys1;
```

# OBJECT VALIDATE

## Purpose

Validates that the specified table is in-sync (on all systems or on the specified systems). The optional ACTIVATE flag requests the table activation.

```
OBJECT VALIDATE < tablename> | id:< object id> ON < tdpid1>, < tdpid2>[,...]TIMEOUT '< timeout  
(seconds)>'] [ACTIVATE]
```

## Parameters

### tablename

The fully qualified name of the target object.

### object id

The managed object id of the target object.

### tdpid

Unique identifier (TDPID) of a target Teradata Database system.

### timeout

The timeout, in seconds, for the operation.

### ACTIVATE

Requests table activation.

## Table in sync

```
unityadmin> OBJECT VALIDATE admin_db.ttt on 1, 2:
Operation Number   : 5
Operation Name     : Validating Object
User               : admin
User Name          : Main Administration User
Start Time         : 02/22 11:06:17
Updates:
```

```

02/22 11:06:17 [-] Info: Validating table admin_db.tttt using timeout 600
02/22 11:06:24 [-] Info: Validation metric: ROWCOUNT
02/22 11:06:24 [-] Info: Validation time: 2019-02-22T19:01:43:784Z
02/22 11:06:24 [-] Info: systems: sdt13140( 1 ), sdt13141( 0 )
02/22 11:06:24 [-] Info: Table is not in Sync
02/22 11:06:24 [-] Info: Validation complete on 2 systems
02/22 11:06:24 [-] Info: Operation finished
Status      : Finished (1)
Finish Time : 02/22 11:06:24

```

## OPERATION CHECK

### Purpose

Displays the current status and messages associated for the specified operation number.

### Syntax

`OPERATION CHECK operation identifier ;`

### Parameter

#### *operation identifier*

Checks status and operation messages for a specific operation number.

### Example

```

unityadmin> OPERATION CHECK 5;
Operation Number : 5
Operation Name   : Activating Systems
User            : admin
User Name       : Main Administration User
Progress (%)    : 100
Status          : Finished (1)
Start Time      : 10/02 14:22:23
Finish Time     : 10/02 14:22:24
Systems:
  [1] sdll3777 - Finished (1)
  [2] sdll3778 - Finished (1)
Updates:
  10/02 14:22:23 [-] Info: Beginning activation of systems
  10/02 14:22:24 [1] Info: Recovery table was successfully created.
  10/02 14:22:24 [2] Info: Recovery table was successfully created.
  10/02 14:22:24 [-] Info: Activated system 1 (sdll3777)

```

```
10/02 14:22:24 [-] Info: Activated system 2 (sdll3778)
10/02 14:22:24 [-] Info: Activated 2 systems
10/02 14:22:24 [-] Info: Successfully activated 2 systems
10/02 14:22:24 [-] Info: Operation finished
```

## OPERATION CLEAR

### Purpose

Clears logging messages associated with a specific Unity management operation.

### Syntax

```
OPERATION CLEAR {ALL | CANCELLED | FAILED | FINISHED | operation identifier} ;
```

### Parameters

#### ALL

Clears all operation messages for completed operations.

#### Example:

```
unityadmin> OPERATION CLEAR ALL;
1 operations cleared successfully.
```

#### CANCELLED

Clears canceled operation messages from the operations list.

#### Example:

```
unityadmin> OPERATION CLEAR CANCELLED;
1 operations cleared successfully.
```

#### FAILED

Clears operation messages for failed operations.

#### Example:

```
unityadmin> unityadmin> OPERATION FAILED;
1 operations cleared successfully.
```

#### FINISHED

Clears messages marked Finished from the operations list.

#### Example:

```
unityadmin> OPERATION CLEAR FINISHED;
2 operations cleared successfully.
```

**operation identifier**

Clears all operation messages for a specific operation number. The operation must be in the Failed or Finished state to be successful. You can run OPERATION LIST to determine the operation status.

**Example:**

```
unityadmin> OPERATION CLEAR 1;
Operation 1 successfully removed.
```

## OPERATION LIST

**Purpose**

Lists the current operations performed in Unity or unityadmin.

**Syntax**

```
OPERATION LIST ;
```

**Example**

```
unityadmin> OPERATION LIST;
The following operations are in progress or finished:
-----
Operation Number : 1
Operation Name   : Activating Systems
User            : admin
User Name       : Main Administration User
Progress (%)    : 100
Status          : Finished (1)
Start Time      : 10/02 00:58:08
Finish Time     : 10/02 00:58:08
Systems:
  [1] sd113777 - Finished (1)
  [2] sd113778 - Finished (1)
-----
Operation Number : 2
Operation Name   : Dictionary Reload
User            : admin
```



```

User Name       : Main Administration User
Progress (%)    : 100
Status         : Finished (1)
Start Time     : 10/02 14:06:14
Finish Time    : 10/02 14:06:19
...

```

## PROCESS ADD

### Purpose

Registers dispatcher, endpoint, and watchdog processes with Unity.

### Syntax

```
PROCESS ADD process_id process_type region_name ;
```

### Parameters

#### *processid*

Identifier assigned to a specific process.

#### *process type*

Represents one of the following values:

Process Type	Description
endpoint	One endpoint per region is automatically added during installation.
dispatcher	Systems configured during setup have two dispatchers defined (HA). When adding new systems, two dispatchers per system must be added for HA environments.
watchdog	Unity allows one watchdog per region. By default, Unity is installed with two regions in an HA environment, and each region has a watchdog defined. If more regions are added, add watchdogs to these regions. These additional watchdogs act as process monitors only and do not participate in determining /resolving sequencer HA status.

#### *region\_name*

Unique character string that identifies a region managed by a Unity server.

### Usage Considerations

You can use the REGION LIST command to view a list of current region names.

Including *region\_name* which enables Unity to route queries based on its proximity to the Teradata Database system.

### Example

```
unityadmin> PROCESS ADD eptTest endpoint pluto;
Successfully added process 'eptTest'.
```

## PROCESS REMOVE

### Purpose

Removes processes from Unity.

### Syntax

```
PROCESS REMOVE process_id ;
```

### Parameters

#### *processid*

Identifier assigned to a specific process.

### Example

```
unityadmin> PROCESS REMOVE eptTest;
Successfully removed process 'eptTest'.
```

## PROCESS STATUS

### Purpose

Lists each process registered with Unity and its status.

### Syntax

```
PROCESS STATUS ;
```

### Example

The PROCESS STATUS command displays the following output.

Label	Description
Process Identifier	Name of the process.
Process Type	Can be Endpoint, Dispatcher, Sequencer, or Watchdog

Label	Description
Region Name	Name of the region associated with this process.
Current Status	Can be Up or Down. The Sequencer that the current unityadmin session is connected to shows its status as: Up - Processing this UnityAdmin connection.

```

unityadmin> PROCESS STATUS;
-----
Process Identifier : region1_dsp_fast
Process Type : Dispatcher
Region Name : region1
Current Status : Up
-----
Process Identifier : region1_ept
Process Type : Endpoint
Region Name : region1
Current Status : Down
-----
Process Identifier : region1_ept2
Process Type : Endpoint
Region Name : region1
Current Status : Down
-----
Process Identifier : region1_seq2
Process Type : Sequencer
Region Name : region1
Current Status : Up
-----
Process Identifier : region1_wdg2
Process Type : Watchdog
Region Name : region1
Current Status : Down
-----
Process Identifier : region1_seq
Process Type : Sequencer
Region Name : region1
Current Status : Up - Processing this Unityadmin connection

```

## PROCESS STOP

## Purpose

Stops any Unity registered process.

## Syntax

**PROCESS STOP** [-diag] [-immediate | -timeout secs] { *process\_id* | *process\_type* } *reason* ;

## Parameters

### -diag

Generates a Teradata Database system diagnostic log file before the process stops.

### -immediate

Stops processes forcefully. Does not shut down processes cleanly.

### -timeout secs

Cleanly shuts down processes in the number of seconds you specify (*secs*). If it does not exit in the indicated time, the process stops forcefully.

### *processid*

Identifier assigned to a specific process.

### *process type*

Type of process to be stopped:

- endpoint
- dispatcher
- sequencer
- watchdog

### *reason*

The reason for stopping the process. Teradata Unity saves this message in the process log. If the *<reason>* contains white spaces, enclose it in double or single quotes.

## Examples

### Dispatcher Example

```
unityadmin> process stop region2_dsp_fast 'Stopping standby dispatcher';
Successfully sent stop message for process 'region2_dsp_fast'.
```

### Watchdog Example

```
unityadmin> process stop -diag -immediate region1_wdg 'Stopping a watchdog';
Successfully sent stop message for process 'region1_wdg'.
```

## RECOVERY LOG USAGE

### Purpose

Provides a total of recovery log , data used by recovery per day for the previous 8 days, and data used by load per day for the last 8 days.

### Syntax

```
RECOVERY LOG USAGE;
```

### Example

```
unityadmin> recovery log usage;
-----
Recovery log and load data usage for last 8 days :
12/14, Recovery log Usage: 4 GB, Data load Usage: 856 GB
12/15, Recovery log Usage: 5 GB, Data load Usage: 163 GB
12/16, Recovery log Usage: 3 GB, Data load Usage: 342 GB
12/17, Recovery log Usage: 4 GB, Data load Usage: 945 GB
12/18, Recovery log Usage: 6 GB, Data load Usage: 645 GB
12/19, Recovery log Usage: 7 GB, Data load Usage: 145 GB
12/20, Recovery log Usage: 3 GB, Data load Usage: 243 GB
12/21, Recovery log Usage: 2 GB, Data load Usage: 234 GB

-----
Load size statistics for last 8 days :

12/14, Largest load by size: 14.305 MB, Largest load by row count: 100000 rows
12/15, Largest load by size: 25500 bytes, Largest load by row count: 500 rows
12/16, Largest load by size: 715.256 MB, Largest load by row count: 5000000 rows
12/17, Largest load by size: 255 bytes, Largest load by row count: 5 rows
12/18, Largest load by size: 7.153 MB, Largest load by row count: 50000 rows
12/19, Largest load by size: 2550 bytes, Largest load by row count: 50 rows
12/20, Largest load by size: 71.526 MB, Largest load by row count: 500000 rows
12/21, Largest load by size: 71 MB, Largest load by row count: 5000 rows

Largest load seen since last sequencer start, by size: 0 bytes, by row count:
0 rows
Largest load seen in last 8 days, by size: 715.256 MB, by row count: 5000000 rows
```

This table explains the output.

Output	Description
Recovery log and load data usage for last 8 days	Shows the amount of recovery log used, and the amount of space used in /data, per day.
Load size statistics for last 8 days	Shows the largest load by size of data packets, as well as by row count, per day. Additionally, also shows the largest load since last time this sequencer was started/became active, as well as largest load seen in last 8 days.

## REGION ADD

### Purpose

Adds a region to the Unity configuration.

### Syntax

```
REGION ADD region_name ;
```

### Parameters

#### *region\_name*

Unique character string that identifies a region managed by a Unity server.

### Usage Considerations

The audit log tracks REGION ADD changes.

### Example

```
unityadmin> REGION ADD sydney;
Successfully added region 'sydney'.
```

## REGION DISABLE LOGONS

### Purpose

Disables session log-on requests on all endpoints in the specified region.

### Syntax

```
REGION DISABLE LOGONS ON region_name ;
```

## Parameters

### *region\_name*

Unique character string that identifies a region managed by a Unity server.

## Usage Considerations

You must specify the name of the region where you want logons disabled. This command stops the listening by endpoints in the specified region for log-on requests on Standby IP (SIP) addresses and default addresses.

When you run this command, it only affects the endpoints already connected to the sequencer. If an endpoint in the specified region later connects to the sequencer, it continues to listen using its current addresses. You may safely re-issue the command to stop listeners for endpoints that connected after you issued the last REGION DISABLE LOGONS command.

## Example

This example command disables session logons on all endpoints in region1.

```
unityadmin> REGION DISABLE LOGONS ON region1;
Command sent successfully to all endpoints. This command is asynchronous.
Use 'ENDPOINT STATUS' command to check each endpoint's global listening status
unityadmin> ENDPOINT STATUS;
-----
Endpoint   : region1_ept
Connected  : yes
Global listen status : Listening disallowed
Listeners :
  Host:Port      : any:1025
  Is SIP         : no
  Associated region : region1 (1)
  State (Current)  : Not Listening
  State (Wanted)   : Not Listening
```

# REGION ENABLE LOGONS

## Purpose

Enables session log-on requests on all endpoints in the specified region.

## Syntax

```
REGION ENABLE LOGONS ON region_name ;
```

## Parameters

### *region\_name*

Unique character string that identifies a region managed by a Unity server.

## Usage Considerations

You must specify the name of the region where you want logons enabled. This command starts endpoint listening on default and standby IP (SIP) addresses that are currently active. If any SIP addresses are defined and running for the specified region on endpoints in other regions, the sequencer instructs those endpoints holding the SIP definitions to stop their SIP listeners.

The REGION ENABLE LOGONS command is asynchronous and only sends a command request to endpoints connected to the sequencer. When the command runs, if there is an endpoint in the specified region that is not connected to the sequencer, this endpoint is not enabled for logons. After an endpoint successfully connects to the sequencer, you can run the command again. Running the command multiple times does not affect those endpoints that are actively listening.

## Example

This example command enables session log-on requests on all endpoints in region1.

```
unityadmin> REGION ENABLE LOGONS ON region1;
Command sent successfully to all endpoints. This command is asynchronous.
Use 'ENDPOINT STATUS' command to check each endpoint's global listening status.
unityadmin> ENDPOINT STATUS;
-----
Endpoint   : region1_ept
Connected  : yes
Global listen status : Listening allowed
Listeners:
  Host:Port      : any:1025
  Is SIP         : no
  Associated region : region1 (1)
  State (Current)  : Listening
  State (Wanted)   : Listening
```

## REGION LIST

### Purpose

Lists all configured Regions and processes in each Region.



**Syntax**

```
REGION LIST ;
```

**Example**

```
unityadmin> REGION LIST;
```

```
-----
```

```
Region Id      : 1
```

```
Region Name    : sandiego
```

## REGION REMOVE

**Purpose**

Removes a region from the Unity configuration.

**Syntax**

```
REGION REMOVE region_name ;
```

**Parameters*****region\_name***

Unique character string that identifies a region managed by a Unity server.

**Usage Considerations**

The command fails if the region does not exist or there are processes or systems associated with it.

**Example**

```
unityadmin> REGION REMOVE toronto;
```

```
Successfully removed region 'toronto'.
```

## REGION RENAME

**Purpose**

Renames an existing Region.

**Syntax**

```
REGION RENAME originalRegionName newRegionName ;
```

## Parameters

### **originalRegionName**

The region name to change.

### **newRegionName**

The new name of the region. The new name cannot currently exist in Unity.

## Example

```
unityadmin> REGION RENAME sydney toronto;
Successfully renamed region.
```

# REWRITE ADD

## Purpose

Adds query rewrite patterns.

## Syntax

```
REWRITE ADD id input regexp output regexp ;
```

## Parameters

### *id*

A unique string to identify the specific Rewrite Rule.

### *input regexp*

The Tool Command Language (TCL) regular expression that enables matching SQL from a client.

To make sure that only the SQL required is matched, make the *input regexp* pattern be as specific as possible. Include a ^ at the beginning and a \$ at the end of each pattern.

### *output regexp*

The SQL regular expression that Unity generates and sends to Teradata.

## Example

The following rewrite pattern forces all selected files on table myDb.myTable that does not have a where clause to use ACCESS locks.

```
unityadmin> REWRITE ADD forceAccessOnMyTable "^SELECT(.*)FROM myDb.myTable$"
"LOCKING TABLE myDb.myTable FOR ACCESS SELECT \1 FROM myDb.myTable";
Rewrite rule add request completed successfully
```

## REWRITE LIST

### Purpose

Lists the current query rewrite patterns configured in Unity.

For each pattern, this command lists:

- id
- input string
- output string

### Syntax

```
REWRITE LIST ;
```

### Example

```
unityadmin> REWRITE LIST;
No rewrite rules were in effect in the database cluster
```

## REWRITE REMOVE

### Purpose

Removes a region from the Unity configuration.

### Syntax

```
REWRITE REMOVE id ;
```

### Parameters

*id*

Represents an existing rewrite rule identifier.

Unity raises an alert if the pattern the *id* value references does not exist.

**Example**

```
unityadmin> rewrite remove example;
Rewrite rule remove request completed successfully
```

## ROUTING DELETE

**Purpose**

Removes the indicated routing definition.

**Syntax**

```
ROUTING DELETE [ RoutingName ] ;
```

**Example**

```
unityadmin> ROUTING DELETE myRouting1;
```

## ROUTING LIST

**Purpose**

List of existing routing rules and the properties that define each rule.

**Syntax**

```
ROUTING LIST < RoutingName > ;
```

**Parameters*****RoutingName***

Name of the routing rule. If you do not specify a routing rule name, all existing routing rule definitions display.

**Examples**

Routing rule definitions display in the following format:

```
<RoutingName>
```

```
(REJECT LOGON) |
```

```
([READ [ALL | < tdpid>[, < tdpid>, ...] [PREFERRED]]
```

```
[WRITE [ALL | < tdpid>[, < tdpid>, ...] [CREATE {PREFERRED | BALANCED}]]]
```

[**OPTIONS** < *OptionName*>[, < *OptionName*>, ...]]

[**ERRORLIST** < *ErrorListName*>)] |

(**PASSIVE** [**ALLOW MANAGED WRITES**]

([**LU**] < *tdpid*>[, < *tdpid*>, ...] | **RR** < *tdpid*>[, < *tdpid*>, ...] [**WEIGHTED** < *n1*>[, < *n2*>, ...])

[**ERRORLIST** < *ErrorListName*>))

This example shows output for a ROUTING LIST command, where the existing rules are DefaultRouting, rejectRouting, managedRouting, and passiveRouting.

```
unityadmin> ROUTING LIST;
DefaultRouting REJECT LOGON
readSys1Only READ sys1
prefReadOnly READ PREFERRED
readSys1WriteAll READ sys1 WRITE sys1, sys2
MSIUser READ WRITE CREATE BALANCED ERRORLIST MsiErrorList
accessUpgrade READ WRITE OPTIONS ACCESS UPGRADE
prPref PASSIVE sys1, sys2
prLU PASSIVE ALLOW MANAGED WRITES LEAST USED sys1, sys2
prRR PASSIVE ROUND ROBIN sys1, sys2
prWRR PASSIVE ROUND ROBIN sys1, sys2 WEIGHTED 2,1
```

## ROUTING UPDATE

### Purpose

Allows you to create routing rules:

#### REJECT LOGON

Use this routing rule when you want to reject a connection.

#### MANAGED

Use this routing rule when you want a managed routing connection.

#### PASSIVE

Use this routing rule when you want a passive routing connection.

### Syntax

**ROUTING UPDATE** < *RoutingName*>

(REJECT LOGON) |

([READ [ALL | < *tdpid*>[, < *tdpid*>, ...] [PREFERRED]]

[WRITE [ALL | < *tdpid*>[, < *tdpid*>, ...] [CREATE {PREFERRED | BALANCED}]]

[OPTIONS < *OptionName*>[, < *OptionName*>, ...]]

---

**Note:**

You must specify options.

---

[ERRORLIST < *ErrorListName*>)] |

(PASSIVE [ ALLOW MANAGED WRITES]

([LU] < *tdpid*>[, < *tdpid*>, ...] | RR < *tdpid*>[, < *tdpid*>, ...] [WEIGHTED < *n1*>[, < *n2*>, ...])

[ERRORLIST < *ErrorListName*>)]

;

## Common Parameters

### *RoutingName*

Name of the routing rule to update.

---

**Note:**

If a routing name does not exist, this command creates the routing name.

---

### *tdpid*

Unique TDPID identifier of a Teradata Database.

### *ErrorListName*

Specifies the error list profile for this routing rule to use.

---

**Note:**

If this parameter is not specified, Unity uses the **DefaultErrorList**.

---

## Reject Logon Parameters

### REJECT LOGON

Creates a routing rule that rejects any new connections that are initiated by the user you specify.

## Managed Routing Parameters

### READ [ALL | <tdpid>[, <tdpid>, ...] ]

Sets the READ attributes for this routing rule by specifying the Teradata Database systems to use when submitting read requests. If you specify the ALL option, all systems managed by Unity become available for read requests. If you specify a list of Teradata Database systems (tdpid), Unity sends read requests to only the systems that you list. By default, the READ parameter uses automatic routing among the systems in the list. If you specify the PREFERRED parameter, the order of the systems in the list defines the preferred routing order for read requests. All routing rules must include this READ parameter.

### PREFERRED

Specifies that read requests adhere to the preferred routing algorithm. A preferred routing rule sends reads to the first TDPID specified. If a system failure or failover occurs, reads are then sent to the next TDPID specified.

---

#### Note:

If this option is not provided, the Default Read Routing behavior is used. The Default Read Routing rule applies auto-routing behavior using the shortest queue algorithm.

---

### WRITE [ALL | <tdpid>[, <tdpid>, ...] ]

Sets the WRITE attributes for the routing rule by specifying the Teradata Database systems to use when submitting write requests. At logon, Unity opens database write sessions only on the systems specified in the WRITE parameter.

---

#### Note:

This parameter is optional. If the **WRITE** parameter is not specified, the routing rule is created as read-only.

---

Parameter	Description
WRITE	Routing rule submits write requests to all managed systems with tables that are affected by the request.
WRITE ALL	Routing rule submits write requests to all systems currently managed by Unity. If you add a new Teradata Database system in the future, this

Parameter	Description
	routing rule sends write requests to the newly added system without you performing additional updates to the rule.
WRITE <tdpid>[, <tdpid>, ...]	Routing rule submits write requests only to the Teradata Database systems (tdpid) in the list. If you specify CREATE PREFERRED, the order of the systems in the list defines the preferred routing order, and Unity sends CREATE requests to the first available system in the list.

## CREATE {PREFERRED | BALANCED}

Specifies the routing method to use for CREATE statements.

### Note:

If the CREATE command is not specified, CREATE statements are sent to all systems and are treated as standard write requests.

Parameter Combination	Description
CREATE PREFERRED	Sends CREATE statements to the first TDPID specified. If Unity detects a failure or failover, CREATE statements are sent to the next TDPIP specified, and all in-flight work is lost.
CREATE BALANCED	Sends all CREATE statements for the given session to the system Unity determines to be least busy. If Unity detects a failure or failover, CREATE statements are sent to the next least busy system, and all in-flight work is lost.

## OPTIONS

Specifies a list of additional options for this routing rule. Options specified by [OPTIONS <OptionName>[, <OptionName>, ...]] are on/off type flags. Each option is turned off for this routing rule by default. The whole [OPTIONS <OptionName>[, <OptionName>, ...]] sub-option is also optional. If not specified, all supported options are turned off for this routing rule. The following <OptionName> is available.

<OptionName>	Description
ACCESS UPGRADE	Automatically upgrade ACCESS locks to READ locks in user request.



## Passive Routing Parameters

### PASSIVE

This keyword indicates the rule is passive. This keyword must be specified for every passive routing rule; otherwise Unity considers the rule managed.

### ALLOWED MANAGED WRITES

If specified, connections that match this rule are allowed to write to objects managed in the Unity dictionary. By default, this option is not enabled.

### LU <tdpid>[,<tdpid>, ...]

Least used. If LU is specified, connections are opened on the tdpid with the least number of connections through Unity.

### RR <tdpid>[,<tdpid>, ...]

Round robin. If RR is specified, connections are opened on the tdpids specified in round robin style. Alternating based on systems listed.

### WEIGHTED <n1>[,<n2>, ...]

Weighted round robin. If WEIGHTED is specified, you can designate the number (0-9999) of open connections on the first system (weights) before going to second system and designating a different number of open connections, and so on. Weighted round robin is useful if your network has different system sizes, allowing you to open more connections on the bigger system. The value is numbers of consuming (accepting) sessions in one round. A zero value indicates an infinite number.

## Usage Considerations

When the Sequencer processes a ROUTING UPDATE command, it updates the routing rules.

At the unityadmin prompt:

Input	Description
ROUTING UPDATE DefaultRouting REJECT LOGON;	Updates the default routing rule to reject incoming connections. This prevents connections from any user without an explicit routing rule definition.
ROUTING UPDATE readSys1Only READ sys1;	Creates a read-only routing rule that only accesses system 1.
ROUTING UPDATE prefReadOnly READ PREFERRED;	Creates a read-only routing rule with a read-preferred algorithm.
ROUTING UPDATE readSys1WriteAll READ sys1 WRITE ALL;	Creates a routing rule that only reads from system 1, but allows writes to all Teradata Database systems.

Input	Description
ROUTING UPDATE MSIUser WRITE CREATE BALANCED ERRORLIST MsiErrorList;	Creates a routing rule that reads and writes to all systems, but only creates volatile tables to one system. Unity chooses the least busy system.
ROUTING UPDATE accessUpgrade READ WRITE OPTIONS ACCESS UPGRADE;	Create a routing rule that reads and writes to all systems, but all ACCESS locks in the user request are upgraded to READ locks.
ROUTING UPDATE prPref PASSIVE sys1, sys2;	Creates a passive routing rule that allows connections to sys1 and sys2. All connections are initially opened to sys1. Connections cannot write to Unity-managed objects.
ROUTING UPDATE prLU PASSIVE ALLOW MANAGED WRITES LU sys1, sys2;	Creates a passive routing rule that allows connections to sys1 and sys2. Each connection is opened on the system with the lowest number of connections through Unity at the time. The connection is allowed to write to managed objects.
ROUTING UPDATE prRR PASSIVE RR sys1, sys2;	Creates a passive routing rule that allows connections to sys1 and sys2. Each connection is opened on the next system in the specific order.
ROUTING UPDATE prWRR PASSIVE RR sys1, sys2 WEIGHTED 2,1;	Creates a passive routing rule that allows connections to sys1 and sys2. The connections are opened in the order of sys1, sys1, and sys2.

## SEQUENCER STATUS

### Purpose

Displays information on the High-Availability (HA) status of the Unity servers.

### Syntax

SEQUENCER STATUS ;

### Output Values

The following table describes output values that display after running SEQUENCER STATUS.

Label	Description
HA Status	<ul style="list-style-type: none"> <li>• Not Enabled</li> <li>• In Peer</li> <li>• Not in Peer</li> </ul>
Log Replication Mode (HA only)	<p>This label only displays for HA installations. The output can be one of the following:</p> <ul style="list-style-type: none"> <li>• Asynchronous</li> <li>• Near synchronous</li> </ul>

Label	Description
	<ul style="list-style-type: none"> <li>• Synchronous</li> </ul>
Individual Sequencer States	State of each Sequencer, which can be one of the following: <ul style="list-style-type: none"> <li>• ACTIVE</li> <li>• DISABLED</li> <li>• DOWN</li> <li>• INACTIVE</li> <li>• STANDBY (SYNCHRONIZING/UNSYNCHRONIZED)</li> <li>• STANDBY (SYNCHRONIZED)</li> <li>• UNKNOWN</li> <li>• WAITING</li> </ul>
Log Percentage to Sync	The percentage of log pending synchronization from the active sequencer to the standby sequencer. Possible values: <ul style="list-style-type: none"> <li>• Percentage pending</li> <li>• Unavailable</li> </ul>
Sequencer State Counts (HA only)	Number of Sequencers in Active and Standby.
Process Identifier	State of each Sequencer provided when at least one Watchdog is connected.

## Examples

The following example shows output if High Availability is enabled.

```

unityadmin> SEQUENCER STATUS;
HA Status           : In Peer
Log Replication Mode : Asynchronous
Log Percentage to Sync: 0
Failover Mode       : Manual

Individual Sequencer States
-----
unity1:5344          : ACTIVE
unity2:5344          : STANDBY (SYNCHRONIZED)

Sequencer State Counts
-----

Active Count          : 1
Standby (Synchronized) Count : 1

```

```

-----
Process Identifier   : region2_seq
Process Type        : Sequencer
Region Name         : region2
Current Status      : Up - Processing this Unityadmin connection
-----
Process Identifier   : region1_seq
Process Type        : Sequencer
Region Name         : region1
Current Status      : Up
-----
Process Identifier   : region1_wdg
Process Type        : Watchdog
Region Name         : region1
Current Status      : Up
-----
Process Identifier   : region2_wdg
Process Type        : Watchdog
Region Name         : region2
Current Status      : Up

```

The following example shows output if High Availability is not enabled.

```

unityadmin> SEQUENCER STATUS;
HA Status           : Not enabled
Log Replication Mode : Asynchronous
Failover Mode       : Manual
-----
Process Identifier   : region1_seq
Process Type        : Sequencer
Region Name         : region1
Current Status      : Up - Processing this Unityadmin connection

```

The following example shows the percentage of log pending synchronization from the active sequencer to the standby sequencer.

```

unityadmin> sequencer status;
HA Status           : Synchronizing
Log Replication Mode : Asynchronous
Log Percentage to sync : 1.84
Failover Mode       : Automatic

```

---

**Note:**

The Log Percentage to Sync is displayed at the active sequencer only if another peer sequencer is connected to the cluster.

---

## SESSION ABORT

### Purpose

Allows for asynchronous abort of currently running or blocked transaction.

### Syntax

```
SESSION ABORT <sessionid> ;
```

### Example

```
unityadmin> session abort 1000;  
Asynchronous abort initiated.
```

## SESSION FAILOVER

### Purpose

For managed routing sessions, the command changes the designated Teradata system for Preferred session routing rule to the next available system in the ordered list.

For passive routing sessions, the command causes the session to move from the current system to the next system, as per the ordered system list associated with the routing rule.

### Syntax

```
SESSION FAILOVER sessionId | routing ;
```

### Parameters

#### *sessionid*

Unique session identifier (ID).

#### *routing*

Unique routing rule identifier.

### Usage Considerations

On sessions tagged for failover, the failover only occurs on the next request.

**Example**

At the unityadmin prompt:

Input	Description
SESSION FAILOVER 1001;	To force a failover of session 1001
SESSION FAILOVER prefReadOnly;	To force a failover of all sessions using the prefReadOnly routing rule

## SESSION KILL

**Purpose**

Requests termination of the specified session.

**Syntax**

```
SESSION KILL unitysessionid ;
```

**Parameters*****unitysessionid***

Specifies the session ID for termination.

**Usage Considerations**

For sessions associated with managed routing rules, this command does not abort in-flight requests. If there is an in-flight request, the session kill occurs once the current request is complete. Session kill commands are tracked in the audit log.

For sessions associated with passive routing rules, the command issues an async abort if there is an in-flight request. The session is closed when the current running query (if any) returns.

**Example**

```
SESSION KILL 1001;
```

## SESSION LIST

**Purpose**

Lists all active client session connections to the Unity server.

## Syntax

```
SESSION LIST ([CONNECTING | IDLE | INTRANS | RECONNECTABLE | DISCONNECTING] [SORT BY
TIME {IDLE | TXN | CONNECTED}]) | (INTERRUPTED [ALL]) | (RECOVERING [<session id>, ...] ON
<tdpid>[, <tdpid>,...]) ;
```

## Parameters

### INTERRUPTED

Displays interrupted sessions with database errors.

### INTERRUPTED ALL

Displays all interrupted sessions.

### RECOVERING

Displays all recovering sessions.

## Examples

Lists managed and passive sessions to the Unity server.

```
unityadmin> session list;
Unity Session Number      : 1000
Client TDP Session Number : 1050
Associated endpoint        : region1_ept (1)
Username                  : testuser1
Logon Source               : fast          634      testuser1  bteq  01 LSS
Client IP Address          : 153.65.194.177
Connected Since           : 10/28 16:58:53
Mode                      : Read-Only (managed)
Error List                 : DefaultErrorList
Session Routing           : managedrouting1
Session State              : Available (in transaction)
Session in transaction     : 1 day 15 hr 54 min 17 sec
Ordered Read Systems       : [1*] flash [2] blaze
System Sessions:
    [1] flash - 5316060 (Active)
    [2] blaze - 5223551 (Active)
-----
Unity Session Number      : 40001
Client TDP Session Number : 1049
Associated endpoint        :          region1_ept (1)
Username                  :      testuser1
```

```

Logon Source      : fast      630      testuser1  bteq  01 LSS
Connected Since   : 10/28 16:58:50
Mode              :          Passive
Error List        :          DefaultErrorList
Session Routing   :          passiverouting1
Session State     :          Available (not in transaction)
System Sessions   :          flash - 5316059 (Active)

```

Output	Description
Unity Session Number	Number of the Unity session.
Client TDP Session Number	Associated Unity Session Number.
Associated endpoint	Name of the Endpoint associated with the session.
Username	Name of the Teradata user.
Account	Account string if specified.
Group	Group name for the user if specified. <b>Note:</b> Define a group in the routing files or with the USER commands.
Client IP Address	IP Address
Connected Since	Time stamp set when a session begins.
Mode	Mode indicates Read-only, Read-Write, or Passive.
Error List	Error list associated with the session routing rule.
Session Routing	Routing rule session is using.
Session State	Availability for sessions.
Session in Transaction	Length of time session has been in transaction (valid only for Managed Routing rules).
Ordered Read Systems	[Optional] Read system associated with session routing rule. Only shown when the routing rule has the read system specified.
Ordered Write Systems	[Optional] Write system associated with session routing rule. Only shown when the routing rule has the write system specified.
System Sessions	Session names and Teradata session numbers. Shows which Teradata Database system is in use and status.

List sessions that are established on the managed servers for recovery.

```

unityadmin> session list recovering all on 1,2;
-----

```



```

System TDP ID                : sdt14336
Outstanding Requests/RecoverybatchSize : 1 / 1000
Restoring Sessions Count     : 1
Time Since Last Activity/RecoveryTimeout : 0 sec / 300 sec
-----
Unity Session Number         : 1002
Session Recovery Status      : Completed (requests pending
at dispatcher)

Sequencer Info:
Previous Error Txn           : 92945
Previous Error Text          : Duplicate row error in bharath.t2.
Previous Error Code          : 2802
Session Established Time     : 12/23 21:23:28
Last Sent Transaction ID     : 92945
Last Expected Commit         : 92945
Requests Sent To Dispatcher  : 1

Dispatcher Info:
System Session Number        : 13678
Session Queue Size           : 0
Session Type                 : DBCSQL
Last Committed Transaction ID : 92944
Current Transaction ID       : 92945
Current Request              : ins t2(2); [Insert]
Run Time (locks-grant + execution) : 30 (0 + 30)
-----
-----
System TDP ID                : sdt14319
Outstanding Requests/RecoverybatchSize : 1 / 1000
Restoring Sessions Count     : 1
Time Since Last Activity/RecoveryTimeout : 0 sec / 300 sec
-----
Unity Session Number         : 1001
Session Recovery Status      : Completed (requests pending
at dispatcher)

Sequencer Info:
Previous Error Txn           : 92938
Previous Error Text          : Table 't3' already exists.
Previous Error Code          : 3803
Session Established Time     : 12/23 21:23:28
Last Sent Transaction ID     : 92938
Last Expected Commit         : 92938

```

```

Requests Sent To Dispatcher          : 1

Dispatcher Info:
System Session Number                : 10482
Session Queue Size                   : 0
Session Type                         : DBCSQL
Last Committed Transaction ID        : 92899
Current Transaction ID               : 92938
Current Request                      : ct t3(i int); [DDL]
Run Time (locks-grant + execution)   : 30 (0 + 30)
-----
Connecting sessions                  : 0
IDLE sessions                        : 2
In transaction sessions              : 0
Reconnectable sessions              : 0
Disconnecting sessions              : 1
Interrupted sessions                : 0
Recent attempted connects            : 3
Recent failed attempts               : 0

```

Output	Description
System TDP ID	System ID
Outstanding Requests /RecoverybatchSize	How many requests have been sent to dispatcher/how many requests can be sent to dispatcher without waiting for response
Restoring Sessions Count	The number of sessions that need to be recovered
Time Since Last Activity /RecoveryTimeout	The amount of time (in seconds) since the last request was sent or the last response was received/The amount of time sequencer would wait for a response before timeout
Unity Session Number	The associated Unity session number
Session Recovery Status	Possible values: <ul style="list-style-type: none"> <li>• Waiting</li> <li>• In Progress</li> <li>• Completed</li> </ul>
System Session Number	The associated TDP session number on the managed server
Session Queue Size	The number of requests yet to be processed at the dispatcher
Session Type	The type of session (DBCSQL, MLOAD, FASTLOAD, EXPORT, UNKNOWN)
Current Transaction ID	The ID of the current transaction at the dispatcher

Output	Description
Current Request	The SQL text and statement type of the current request at the dispatcher
Run Time (locks-grant + execution)	The time (in seconds) taken by the current request at the dispatcher (waiting for locks grant and database server)
Pending Locks	The object IDs of the locks waiting for grant at the dispatcher
Previous Error Txn	The transaction ID of previous failure
Previous Error Text	Error text explaining the previous failure
Previous Error Code	Error code of previous failure
Session Established Time	The time the session was established
Last Sent Transaction ID	The transaction ID of the last request sent to the dispatcher
Last Committed Transaction ID	The last transaction ID committed by this session
Last Expected Commit	The last transaction ID that this session has to commit to complete recovery

Lists information on an interrupted session with a database error. In this example, Unity is trying to access a database that does not exist.

```
unityadmin> session list INTERRUPTED;
-----
Unity Session Number      : 1000
Unity Transaction ID      : 0
System                    : jasonu4
Root Cause                : Yes
Interrupt Time            : 09/29 16:17:16
Time of Original Query    : 09/29 16:07:09
Error Code                : 3802
Error Text                : Database 'jason2' does not exist.
SQL Text                  : database jason2;
-----
```

Output	Description
Unity Session Number	Number of the Unity session.
Unity Transaction ID	Number associated with transaction.
System	System associated with interrupted session.
Root Cause	<b>Yes</b> indicates the cause was from a database issue. <b>No</b> indicates the session was interrupted by another session.

Output	Description
Interrupt Time	Timestamp when system became interrupted.
Time of Original Query	Timestamp when session attempted to access database.
Error Code	Code associated with error.
Error Text	Text string indicating error.
SQL Text	SQL Text string showing error.

## SESSION MOVE

### Purpose

For sessions associated with a managed routing rule, this command changes the Teradata system for the Preferred session routing rule to another system by specifying the TDPID of that system.

For sessions associated with passive routing rules, this command moves the sessions from the current system to the one specified. The TDPID specified must be part of the routing rule associated with the session.

### Syntax

```
SESSION MOVE { sessionid | routing } TO tdpid ;
```

### Parameters

#### *sessionid*

Unique session identifier (ID).

#### *routing*

Unique routing rule identifier.

#### *tdpid*

Unique identifier (TDPID) of a Teradata Database system.

### Usage Considerations

On sessions tagged for move, the move only occurs on the next request. The audit log tracks session changes.

### Examples

At the unityadmin prompt:

Input	Description
SESSION MOVE 1001 TO sys2;	The example demonstrates how to force a move of session 1001 to system sys2.
SESSION MOVE prefReadOnly TO sys2;	The example demonstrates how to force a move of all sessions using the prefReadOnly routing rule to system sys2.

## SESSION RECOVER

### Purpose

Starts session-level recovery operation. Enables auto-recovery before starting recovery of halted sessions.

### Syntax

SESSION RECOVER { *sessionid list* } ON { *systems list* } ;

### Example

```
unityadmin> session recover 1000 on 1;
Operation Number : 13
Operation Name   : Recover Session
User            : admin
User Name       : Main Administration User
Start Time      : 04/15 22:35:11
Systems:
  [1] sdt14336
Updates:
  04/15 22:35:11 [-] Info: Successfully started recovery on system 1
  04/15 22:35:11 [-] Info: Operation finished
Status          : Finished (1)
Finish Time     : 04/15 22:35:11
Systems:
  [1] sdt14336 - Finished (1)
```

## SESSION RECOVER HALT

### Purpose

Disables auto-recovery of sessions. Object and system-level recovery operations will not recover these sessions.

**Syntax**

```
SESSION RECOVER HALT { sessionid list } ON { system list } REASON { reason } ;
```

**Example**

```
unityadmin> session recover halt 1000 on 1 reason "demo";
Operation Number : 15
Operation Name   : Recover Session
User            : admin
User Name       : Main Administration User
Start Time      : 04/15 22:38:58
Systems:
  [1] sdt14336
Updates:
  04/15 22:38:58 [-] Info: 1 session halted on system 1
  04/15 22:38:58 [-] Info: Operation finished
Status          : Finished (1)
Finish Time     : 04/15 22:38:58
Systems:
  [1] sdt14336 - Finished (1)
```

## SESSION RECOVER RESUME

**Purpose**

Enables auto-recovery of sessions.

**Syntax**

```
SESSION RECOVER RESUME { sessionidlist | ALL } | ON { systems list } ;
```

**Example**

```
unityadmin> session recover resume 1000 on 1;
Operation Number : 9
Operation Name   : Recover Session
User            : admin
User Name       : Main Administration User
Start Time      : 04/15 22:05:24
Systems:
  [1] sdt14336
Updates:
  04/15 22:05:24 [-] Info: Auto-recovery enabled for session 1000.
```

```

04/15 22:05:24 [-] Info: Operation finished
Status      : Finished (1)
Finish Time : 04/15 22:05:24
Systems:
  [1] sdt14336 - Finished (1)

```

## SESSION SKIP APPLY

### Purpose

Allows Unity to skip a session that is interrupted.

### Syntax

```
SESSION SKIP APPLY < Unity Session ID> < transaction ID> ON < tdpid> [ ACTIVE];
```

### Parameters

#### *Unity Session ID*

Assigned Unity session ID.

#### Transaction ID

Assigned Transaction ID.

#### tdpid

Unique identifier (TDPID) of a Teradata Database system.

#### ACTIVE

Disables the default behavior which sets all tables to unrecoverable. Instead, checks to see if the objects can be set to Active.

### Usage Considerations

This command informs the recovery engine that the specified transaction on the specified session and TDPID should be skipped during recovery.

---

#### Note:

Skipping a transaction can cause tables to become out of sync immediately and other tables to become unrecoverable in subsequent transactions.

---

**Note:**

The transaction ID specified must be the transaction ID that is currently blocking recovery of the specified session. If the transaction ID does not match the session's interrupt transaction ID, the following error is returned: Specified transaction ID does not match expected interrupted transaction ID for session.

**Example**

```
unityadmin> SESSION SKIP APPLY 1000 437 on flash;
Warning: Skipping this transaction can cause tables to
become out of sync immediately and other tables to
become unrecoverable in subsequent transactions.
```

To see a list of affected tables refer to the SESSION SKIP LIST command.

Enter 'Yes' to skip this transaction: yes

Tables affected by transaction

```
db2inst2.doc_test3
db2inst2.doc_test4
```

2 tables

## SESSION SKIP LIST

**Purpose**

Retrieves a list of tables that are affected by a skip transaction request.

**Syntax**

```
SESSION SKIP LIST < Unity Session ID> < transaction ID> ON < tdpid>;
```

**Parameters****Unity Session ID**

Assigned Unity session ID.

**Transaction ID**

Assigned Transaction ID.



**tdpid**

Unique identifier (TDPID) of a Teradata Database system.

**Usage Considerations**

This command returns the list of objects that will be affected if the specified transaction ID is skipped during recovery.

**Note:**

This transaction ID specified must be the transaction ID that is currently blocking recovery of the specified session. If the transaction ID does not match the session's interrupt transaction ID, the following error is returned: "Specified transaction ID does not match expected interrupted transaction ID for session."

**Example**

```
unityadmin> SESSION SKIP LIST 100 436 on flash;
Tables affected by transaction

_____
db2inst2.doc_test3
db2inst2.doc_test4
_____
2 tables
```

## SESSION UNTAG

**Purpose**

Reverts or cancels any pending or unapplied session moves or failovers for the specified session routing.

**Syntax**

```
SESSION UNTAG sessionId | routing ;
```

**Parameters*****sessionId***

Unique session identifier (ID).

***routing***

Unique routing rule identifier.

## Examples

At the unityadmin prompt:

Input	Description
SESSION UNTAG 1001;	The following example demonstrates how to untag pending actions for session 1001.
SESSION UNTAG prefReadOnly;	The following example demonstrates how to untag pending actions for all sessions using the prefReadOnly routing rule.

# SESSIONMAPPING EXPORT

## Purpose

Exports the current routing information to the specified file.

## Syntax

```
SESSIONMAPPING EXPORT filename ;
```

## Parameters

### *filename*

Names the export details file created after running SESSIONMAPPING EXPORT.

If you specify the filename without a full path, Unity creates the exported session mapping file in the directory from which unityadmin was launched.

## Example

```
unityadmin> SESSIONMAPPING EXPORT sessionMapping.txt;
Successfully exported to file 'sessionMapping.txt'
```

# STATUS

## Purpose

To provide an overall status on all components.

## Syntax

```
STATUS
```

## Usage Considerations

The status command allows you to see the states of the sequencers, systems, endpoints, dispatchers, gateways, status, and session counts at the same time without having to view the dashboard. This command is only available on the active sequencer.

## Example

```
unityadmin> status;
Sequencer: ranch1_seq-sdlc0751(active)
Sequencer: diego1_seq-sdlc0960(standby-synchronized)

Endpoint ranch1_ept: any(listening):1025
Endpoint diego1_ept: any(listening):1025

System sdlc0797 (active) Tables: OOS 0, standby 0, unrecoverable 0, interrupted
0, restore 0, read-only 0, active 0
Dispatcher status: diego1_dsp_sdlc0797(standby) - up, ranch1_dsp_sdlc0797
(active) - up
Gateway status: sdlc0797 - sdlc0797cop1(up)
Connected sessions: 1

System sdlc0962 (active) Tables: OOS 0, standby 0, unrecoverable 0, interrupted
0, restore 0, read-only 0, active 0
Dispatcher status: diego1_dsp_sdlc0962(standby) - up, ranch1_dsp_sdlc0962
(active) - up
Gateway status: sdlc0962 - sdlc0962cop1(up)
Connected sessions: 2

Alerts: 0 critical, 0 warning, 2 normal
```

# SYSTEM ACTIVATE

## Purpose

Activates all systems on initial setup or activates one or more specified systems, creates the transaction table, and verifies the existence of the logging tables. If logging tables do not exist, the command creates the tables on the target system under the user that holds the transactions table. Use for an initial setup or installation where all Teradata Database systems are in the Unrecoverable state and for activating a system when there are other Teradata Database systems already in the Active state.

Running SYSTEM ACTIVATE with the **SYNCD FROM** clause changes table states to Active.

## Syntax

SYSTEM ACTIVATE **ALL** | { *tdpid* [, *tdpid*, ...] } | [ **NO TABLES**] [ REASON '<reason>' | *destination tdpid* SYNCED FROM *source tdpid* ;

## Parameters

### ALL

Places all Teradata Database systems and tables into Active state.

---

#### Important:

All data must be synchronized across all Teradata Database systems before running the **ALL** clause.

---

### *tdpid*

Unique identifier (TDPID) of a Teradata Database system.

### NO TABLES

With specifying NO TABLES, target systems specified by <tdpid> [, <tdpid>, ...] are marked Active and all tables in those systems will be Unrecoverable. Without this option, by default all tables are marked Active.

### SYNCED FROM *source tdpid*

Places the Teradata Database system and all common tables between *destination tdpid* and *source tdpid* into Active state.

## Usage Considerations

When using the SYNCED FROM clause, if the <*destination tdpid*> Teradata Database system is Unrecoverable, the Teradata Database system must be placed into the Active state before the tables can become Active.

If specific Teradata Database systems are listed, the SYSTEM ACTIVATE command places all specified Teradata Database systems and tables on those Teradata Database systems into Active state. This assumes that all the data on the specified Teradata Database systems has been synchronized across the specified Teradata Database systems.

The **NO TABLES** option is in conflict with ALL and SYNCED FROM and cannot be used at the same time.

With the **NO TABLES** option, systems can be activated without SYNCED FROM even though an Active system exists. SYSTEM ACTIVATE with option **NO TABLES** can be applied to Active systems as well. If a system is Active, applying SYSTEM ACTIVATE with **NO TABLES** keeps the system Active and all tables in the system are Unrecoverable.

To run the unsynced `SYSTEM ACTIVATE` command, all Teradata Database systems must be Unrecoverable.

## Examples

At the unityadmin prompt:

Input	Description
<code>SYSTEM ACTIVATE ALL;</code>	The following example demonstrates how to activate all systems that are currently connected to the Sequencer
<code>SYSTEM ACTIVATE tda;</code>	The following example demonstrates how to activate the system tda. All other systems are marked unrecoverable
<code>SYSTEM ACTIVATE sys2 SYNCED FROM sys1;</code>	The following example demonstrates how to activate synchronized tables in system sys2 from source system sys1  <b>Note:</b> Running <code>SYNCED FROM</code> does not copy data from one system to another. In the above example, the data on sys1 and sys2 must be the same when the command is run.

## SYSTEM ADD

### Purpose

Adds a Teradata Database system to the list of databases available to Unity by adding the appropriate system TDPID. You will also need to add a dispatcher process, modify the `unity.properties` file, and add the mgmtuser to the system being added.

For Lowest Common Denominator (LCD) purposes, new Teradata Database systems are added to the end of the Teradata Database system list. You can use the `SYSTEM ORDER` command to change the order.

### Syntax

```
SYSTEM ADD tdpid region_name OVERRIDE ;
```

### Parameters

#### *region\_name*

Unique character string that identifies a region managed by a Unity server.

#### *tdpid*

Unique identifier (TDPID) of a Teradata Database system.

**OVERRIDE**

Each system is assigned a system serial number. This serial number is stored on the system in a table. The OVERRIDE option overwrites an existing serial number on the system, if one exists.

**Example**

This example command adds a Teradata system named newSys1 to region2.

```
unityadmin> SYSTEM ADD newSys1 region2;
Successfully added system 'newSys1'.
```

## SYSTEM DEACTIVATE

**Purpose**

Changes all Teradata Database systems or a specified system to the Unrecoverable state.

**Syntax**

```
SYSTEM DEACTIVATE | { tpdid [, tpdid, ...] } | { database.tablename [ ON tpdid [, tpdid, ...]] } ;
```

**Parameters*****database.tablename***

Name of a database and table currently in the Unity Data Dictionary. The table name that you specify must be fully qualified.

***tpdid***

Unique identifier (TDPID) of a Teradata Database system.

**Usage Considerations**

Unless a table is specified, running this command sets all tables to Unrecoverable. Specify a single table to mark it Unrecoverable.

**Example**

The following example demonstrates placing a Teradata Database system with the TDPID bltor in Unrecoverable state, followed by running OPERATION CHECK to view status.

```
unityadmin> SYSTEM DEACTIVATE bltor;
The request is currently processing as operation number 1.
You may check its status using the command 'operation check 1'.
unityadmin> OPERATION CHECK 1;
```

```

Operation Number : 1
Operation Name   : Deactivating Systems
User            : admin
User Name       : Main Administration User
Progress (%)     : 100
Status          : Finished (1)
Start Time      : 11/14 16:03:35
Finish Time     : 11/14 16:03:36
Systems:
  [1] bltor - Finished (1)
Updates:
  11/14 16:03:36 [-] Info: Beginning deactivation of systems
  11/14 16:03:36 [-] Info: Resetting system 1 (bltor)
  11/14 16:03:36 [-] Info: Reset system 1 (bltor)
  11/14 16:03:36 [-] Info: Successfully deactivated 1 systems
  11/14 16:03:36 [-] Info: Operation finished

```

## SYSTEM FREEZE

### Purpose

Changes a Teradata Database system or table from Active state to Read-Only state.

### Syntax

```
SYSTEM FREEZE { tpdid [, tpdid,...] } | { database.tablename [ ON tpdid [, tpdid,...] ] } [" reason"] ;
```

### Parameters

#### *database.tablename*

Name of a database and table currently in the Unity Data Dictionary. The table name that you specify must be fully qualified.

#### *tpdid*

Unique identifier (TDPID) of a Teradata Database system.

#### "*reason*"

User description of why the object is having its state changed.

### Usage Considerations

Unless a table is specified, running this command sets all tables to Read-Only. Specify a single table to mark it Read-Only.

If making a state change for a specific table, then the table on all Teradata Database systems becomes Read Only.

### Example

The following example demonstrates placing a Teradata Database system with the TDPID fast in Read Only state.

```
unityadmin> system freeze fast;
The request is currently processing as operation number 3.
You may check its status using the command 'operation check 3'.
unityadmin> operation check 3;
Operation Number : 3
Operation Name   : Freezing Systems
User            : admin
User Name       : Main Administration User
Progress (%)     : 0
Status          : In Progress (0)
Start Time      : 01/09 12:49:47
Finish Time     : 12/31 16:00:00
Systems:
  [2] fast - Pending (2)
Updates:
  01/09 12:49:48 [-] Info: Freezing selected systems
```

## SYSTEM GATEWAYS

### Purpose

Displays the status for all gateways for a Teradata Database system.

### Syntax

```
SYSTEM GATEWAYS [ tdpid [, tdpid, ...] ] ;
```

### Parameters

***tdpid***

Unique identifier (TDPID) of a Teradata Database system.

### Example

```
unityadmin> SYSTEM GATEWAYS bltor;
System ID           : 1
System TDP ID       : bltor
```



```

System Monitor Status : Monitored
Gateways:
  COP Name           : bltorcop1
  Gateway Status     : Up
  Addresses:
    IP Address       : 153.64.210.148
    Network Status   : Up
    Last Status Change Time : 10/05 16:35:22
    LCC Interval Timer (ms) : 1
    Session Connect Count : 1
  COP Name           : bltorcop2
  Gateway Status     : Up
  Addresses:
    IP Address       : 153.64.210.149
    Network Status   : Up
    Last Status Change Time : 10/05 16:35:22
    LCC Interval Timer (ms) : 1
    Session Connect Count : 0
  Dispatcher 1: Not available
  Dispatcher 2: Not available

```

```

-----
System ID           : 2
System TDP ID       : fast
System Monitor Status : Unmonitored
  Dispatcher 1: Not available
  Dispatcher 2: Not available

```

**Note:**

Dispatcher information always displays Not available. To display Dispatcher information for Teradata Database systems, run the `ENDPOINT LIST` command.

## SYSTEM HALT

### Purpose

Changes a table or Teradata Database system in the Active state to the Out-of-Service state.

**Note:**

If a multi-table write runs, and a table is in Out of Service or Interrupted state, then all associated tables in the write automatically become Out of Service or Interrupted.

## Syntax

```
SYSTEM HALT tdpid | tdpid [ON database.tablename[, tdpid,...]] ;
```

## Parameters

### *database.tablename*

Name of a database and table currently in the Unity Data Dictionary. The table name that you specify must be fully qualified.

### *tdpid*

Unique identifier (TDPID) of a Teradata Database system.

## Usage Considerations

If making a state change for a table on all the Teradata Database systems, the table has same data across all Teradata Database systems when this operation completes.

When running this command, Unity holds any new transactions/statements until all in-flight transactions close for the object. **RecoveryCheckpointTimeout** determines the duration of wait time. If in-flight transactions are not closed within this time limit, the operation aborts, an error displays, and the state returns to the state prior to the attempt.

When making the state change to the Teradata Database system, all tables Unity manages on that Teradata Database system become Out of Service.

If applied to the Teradata Database system successfully, all tables in the Teradata Database system become Out of Service. If this operation is applied to a Teradata Database system, all tables must be Active, else the operation results in an error, and no action takes place.

## Example

The following example shows using SYSTEM HALT to place a table t4\_cp\_tpp from a Teradata Database cp\_tpp004u in Out of Service state. The administrator then runs OPERATION CHECK to confirm operation completion.

```
unityadmin> system halt cp_tpp004u.t4_cp_tpp;
The request is currently processing as operation number 2.
You may check its status using the command 'operation check 2'.
unityadmin> operation check 2;
Operation Number : 2
Operation Name   : Halting Table
User            : admin
User Name       : Main Administration User
Progress (%)    : 100
Status          : Finished (1)
```

```

Start Time      : 01/08 13:13:33
Finish Time     : 01/08 13:13:36
Systems:
  [1] bltor - Finished (1)
  [2] fast - Finished (1)
Updates:
  01/08 13:13:36 [-] Info: Halting table cp_tpp004u.t4_cp_tpp
  01/08 13:13:36 [-] Info: Requesting mgmt X lock on 'cp_tpp004u.t4_cp_tpp'
  01/08 13:13:36 [-] Info: Mgmt X lock on 'cp_tpp004u.t4_cp_tpp' granted
  01/08 13:13:36 [-] Info: Releasing mgmt X lock on 'cp_tpp004u.t4_cp_tpp'
  01/08 13:13:36 [-] Info: Successfully halted table on 2 systems
  01/08 13:13:36 [-] Info: Operation finished

```

## SYSTEM LIST

### Purpose

Lists Teradata Database systems added to Unity in the order they used for lowest common denominator (LCD) calculation.

### Syntax

```
SYSTEM LIST ;
```

### Example

```

unityadmin> SYSTEM LIST;
unityadmin> sd113754, sd113776

```

## SYSTEM LIST STATES

### Purpose

Provides state information for the specified object and all child objects.

### Syntax

```
SYSTEM LIST STATES { tdpid [, tdpid, ...] } | { database.tablename [ ON tdpid [, tdpid, ...]] } ;
```

### Parameters

#### *database.tablename*

Name of a database and table currently in the Unity Data Dictionary. The table name that you specify must be fully qualified.

***tdpid***

Unique identifier (TDPID) of a Teradata Database system.

**Usage Considerations**

Lists table states on one or more specific Teradata Database systems.

**Examples**

The following example shows the state of a Teradata Database called sdll3776 with a portion of the output.

```
unityadmin> SYSTEM LIST STATES sdll3776;
State of b00323a_usr0.j2:
  State of b00323a_usr0.j2 on system 2 (sdll3776): active
State of b00323a_usr0.j3:
  State of b00323a_usr0.j3 on system 2 (sdll3776): active
State of b00324a_usr0.t1:
  State of b00324a_usr0.t1 on system 2 (sdll3776): active
State of b00325a_usr0.tbl:
  State of b00325a_usr0.tbl on system 2 (sdll3776): active
...
```

The following example shows the state of a table called b00383a\_usr0.billings on the Teradata Database systems called sdll3754 and sdll3776.

```
unityadmin> SYSTEM LIST STATES b00383a_usr0.billings ON sdll3754,sdll3776;
State of b00383a_usr0.billings:
  State on system 1 (sdll3754): active
  State on system 2 (sdll3776): active
```

## SYSTEM ORDER

**Purpose**

Specifies the order of Teradata Database systems that Unity uses to calculate lowest common denominator (LCD) information.

**Syntax**

```
SYSTEM ORDER < tdpid>, < tdpid>[, tdpid, ...] ;
```

## Parameters

### *tdpid*

Unique identifier (TDPID) of a Teradata Database system.

## Usage Considerations

- All current Teradata Database systems (use `SYSTEM LIST` to list all current Teradata Database systems) must be included in the list. For example, if `td1`, `td2` and `td3` have been added to Unity, the `SYSTEM ORDER` command must include all three TDPID values. The `SYSTEM ORDER` command cannot just specify `td1` and `td2`.
- Executing this command multiple times overwrites existing information. If changed, the ordering only affects new sessions. Existing sessions already have the LCD information calculated and do not change if this command is run.
- If this command is not run, the default order of the Teradata Database systems results in the order with which they were added to Unity (using the `SYSTEM ADD` command, and using the order in which the systems were added during installation).

## Example

The following example uses the `SYSTEM ORDER` to place `fast` as the first system. You can run the `SYSTEM LIST` to show the change in system order.

```
unityadmin> SYSTEM ORDER fast,bltor;
Successfully ordered systems.
unityadmin> SYSTEM LIST;
fast, bltor
```

# SYSTEM RECOVER

## Purpose

Recovers Teradata Database systems or tables that are in Out-of-Service, Interrupted, or Read-Only state.

## Syntax

```
SYSTEM RECOVER < tdpid > [, < tdpid >, ...] [REASON '< reason >']
```

## Parameters

### *tdpid*

Unique identifier (TDPID) of a Teradata Database system.

**REASON**

Reason for clearing. Use quotation marks when the reason is more than one word.

**Usage Considerations**

All Unity managed tables for the specified tdpid(s) change to **RESTORE** state and go through recovery.

If the table cannot recover successfully due to a transient error, then the table remains in *Interrupted* state, which raises an alert. After resolving the error, retry the recovery.

If Unity cannot successfully recover the table due to data consistency issues, then the table enters *Unrecoverable* state.

The command operation replays the Recovery Log to bring the Teradata Database server from *Out of Service*, *Interrupted*, or *Read Only* state to *Active* state without requiring manual intervention to restore the data. If the recovery operation is unsuccessful, the Teradata Database system or table becomes *Interrupted* or *Unrecoverable* based on the error returned.

Running this operation on a Teradata Database system places each table in that Teradata Database system from *Out of Service* or *Interrupted* into *Restore* state. The system completes recovery within a short period while tables on the system continue recovery at an individual level. As each table completes recovery successfully, it becomes *Active* and accessible for client transactions.

**Example**

The following example demonstrates system level recovery after running **SYSTEM FREEZE**.

```
unityadmin> system freeze bltor;
Operation Number : 4
Operation Name   : Freezing Systems
User            : admin
User Name       : Main Administration User
Start Time      : 10/19 12:34:06
Systems:
  [1] bltor
Updates:
  10/19 12:34:08 [-] Info: Freezing selected systems
  10/19 12:34:08 [-] Info: Successfully froze 1 systems
  10/19 12:34:08 [-] Info: Operation finished
Status          : Finished (1)
Finish Time     : 10/19 12:34:08
Systems:
  [1] bltor - Finished (1)

unityadmin> system recover bltor;
Operation Number : 5
Operation Name   : Recovering Systems
```

```

User           : admin
User Name      : Main Administration User
Start Time     : 10/19 12:34:38
Systems:
  [1] bltor
Updates:
  10/19 12:34:38 [-] Info: Recovering selected systems
  10/19 12:34:38 [-] Info: Successfully started recovery on 1 systems
  10/19 12:34:38 [-] Info: Operation finished
Status        : Finished (1)
Finish Time   : 10/19 12:34:38
Systems:
  [1] bltor - Finished (1)

```

## SYSTEM REMOVE

### Purpose

Removes a Teradata Database system from the Unity-managed configuration.

### Syntax

```
SYSTEM REMOVE tdpid ;
```

### Parameters

#### *tdpid*

Unique identifier (TDPID) of a Teradata Database system.

### Usage Considerations

- After you remove a Teradata Database system, the system becomes unavailable to Unity for Read or Write requests.
- The TDPID that you specify must be valid within Unity. The TDPID must previously be added to Unity.
- The TDPID cannot be referenced in routing rules. To successfully run this command, first remove all references to the Teradata Database system from the routing rules.
- The Teradata Database system must be in one of the following states:
  - Disconnected
  - Unrecoverable

### Example

The following scenario includes two Active systems, *bltor* and *fast*. The example first runs SYSTEM DEACTIVATE to deactivate the system *bltor*.

```
SYSTEM DEACTIVATE bltor;
```

```
unityadmin> Operation Number : 6
Operation Name   : Deactivating Systems
User            : admin
User Name       : Main Administration User
Start Time      : 10/19 12:51:06
Systems:
  [1] bltor
Updates:
  10/19 12:51:06 [-] Info: Beginning deactivation of systems
  10/19 12:51:06 [-] Info: Resetting system 1 (bltor)
  10/19 12:51:07 [-] Info: Reset system 1 (bltor)
  10/19 12:51:07 [-] Info: Successfully deactivated 1 systems
  10/19 12:51:07 [-] Info: Operation finished
Status          : Finished (1)
Finish Time     : 10/19 12:51:07
Systems:
  [1] bltor - Finished (1)
```

After the command finishes, the system *bltor* is placed in the Unrecoverable state and ready for removal.

```
SYSTEM REMOVE bltor;
Successfully removed system 'bltor'.
```

## TIME WINDOW CLOSE

### Purpose

Removes the specified TIME WINDOW.

### Syntax

```
TIME WINDOW CLOSE <name>
```

### Example

```
unityadmin> time window close 'Upgrade Outage';
Time Window has been closed.
```

## TIME WINDOW DISABLE



**Purpose**

Opens the specified TIME WINDOW now.

**Syntax**

TIME WINDOW DISABLE <name>

**Example**

```
unityadmin> time window disable 'Nightly backups';
Time window 'Nightly backups' was disabled.
```

## TIME WINDOW ENABLE

**Purpose**

Removes the specified TIME WINDOW.

**Syntax**

TIME WINDOW ENABLE <name>

**Example**

```
unityadmin> time window enable 'Nightly backups';
Time window 'Nightly backups' was enabled.
```

## TIME WINDOW LIST

**Purpose**

Lists all defined TIME WINDOWS.

**Syntax**

TIME WINDOW LIST <name>

**Example**

```
unityadmin> time window list;
-----
Name           : Upgrade Outage (ENABLED)
Status        : CLOSED
Open Action    :
Close Action   :
```

```

Year          : 2018
Month         : 12
Day of Month  : 16
Time          : 0900 - 1400
-----
Name          : Nightly backups (ENABLED)
Status        : CLOSED
Open Action   :
Close Action  :
Time          : 2200 - 0100
2 Time windows defined.

```

## TIME WINDOW OPEN

### Purpose

Closes the specified TIME WINDOW now.

### Syntax

```
TIME WINDOW OPEN <name>
```

### Example

```

unityadmin> time window open 'Upgrade Outage';
Time Window has been opened.

```

## TIME WINDOW REMOVE

### Purpose

Removes the specified TIME WINDOW.

### Syntax

```
TIME WINDOW REMOVE <name>
```

### Example

```

unityadmin> time window remove 'Upgrade Outage';
Time window 'Upgrade Outage' was removed.

```

## TIME WINDOW UPDATE

**Purpose**

Adds or updates a TIME WINDOW.

**Syntax**

TIME WINDOW UPDATE

*<name>* [YEAR *<Start>* [ to *<End>*]] [MONTH *<Start>*[ to *<End>*] [MDAY *<Start>*[ to *<End>*] [WDAY *<Start>*[ to *<End>*]] [HOUR *<Start>*[ to *<End>*]] [MIN *<Start>*[ to *<End>*]]

**Example**

```
unityadmin> time window update 'Upgrade Outage' year 2018 month 12 mday 16 time
0900 to 1400 enable;
Successfully added time window 'Upgrade Outage'. It is currently ENABLED.
```

## UNITYMGMT PASSWORD RELOAD

**Purpose**

This command re-reads the value of the Unity Management password from the database or Teradata Wallet.

**Syntax**

UNITYMGMT PASSWORD RELOAD [ REASON'*< reason>*' ] ;

**Parameters****REASON**

Enter a reason for the password reload.

**Usage Considerations**

This command results in an error if the Unity Management Password value does not exist in Unity. It is recommended that the new password be first set in Teradata Wallet and loaded into Unity before it is changed on the managed systems.

**Examples**

```
unityadmin> UNITYMGMT PASSWORD RELOAD REASON update
```

## UNITYMGMT PASSWORD SET

## Purpose

This command allows you to change the UNITYMGMT password.

## Syntax

```
UNITYMGMT PASSWORD SET < newpassword> ;
```

## Parameters

### *newpassword*

New administrator password.

## Examples

```
unityadmin> UNITYMGMT PASSWORD SET passExampleID;
Successfully changed the password of UNITYMGMT 'passExampleID'.
```

# USER DELETE

## Purpose

Removes the referenced user mapping from the list. The user mapping is referenced using the Unity-assigned user mapping ID. You can also use user mapping properties, such as region, user name, account string, and so forth, to specify criteria against which to match user mappings for deletion.

## Syntax

```
USER DELETE < mapId> | < username> < accountstring> < role> < profile> < region> ;
```

## Parameters

### *mapId*

Identifier for the user mapping to delete.

You can retrieve the ID by executing the USER LIST command. The user mapping ID is also available from the Unity Configuration portlet interface.

### *username*

Name of the user. Name can contain a maximum of 25 characters.

### *accountstring*

Account string for the user mapping to delete. The account string you specify must be an exact match for the target user mapping.

**role**

Role of the user mapping to delete. This must be an exact match to the target user mapping.

**profile**

Profile of the user mapping to delete. This must be an exact match to the target user mapping.

**region**

Specifies either the name of a Unity region or asterisk (\*) to indicate any region. Use this parameter to determine a match with the logon Endpoint's region. If you do not specify a region, the default asterisk (\*) is used to specify any region.

**Usage Considerations**

Username of the user mapping to delete must be an exact match to the target user mapping.

**Examples**

The following example demonstrates how to delete the user mapping mapID 4.

```
unityadmin> USER DELETE 4;
Successfully removed user map 4.
```

The following example deletes the user mapping for user dbUser2, account string acct\*, role \*, and profile \* values.

```
unityadmin> USER DELETE dbUser2 acct* * *;
Successfully removed user map 3.
```

## USER LIST

**Purpose**

Lists the defined user mappings and optionally filters by matching criteria.

**Syntax**

```
USER LIST [ < username> [ < accountstring> [ < role> [ < profile> [ < region> ] ] ] ] ] ;
```

**Parameters****username**

Name of the user. Name can contain a maximum of 25 characters.

**accountstring**

Account string of the user mapping to list.

**role**

Role of the user mapping to delete. This must be an exact match to the target user mapping.

**profile**

Profile of the user mapping to delete. This must be an exact match to the target user mapping.

**region**

Specifies either the name of a Unity region or asterisk (\*) to indicate any region. Use this parameter to determine a match with the logon Endpoint's region. If you do not specify a region, the default asterisk (\*) is used to specify any region.

**Usage Considerations**

The user ID determines the order in which the user mappings are matched. For each new connection, Unity chooses the user mapping with the lowest ID matching the session user name, region, account string, role, and profile.

**Examples**

The following example executes the USER LIST command without parameters.

```
unityadmin> USER LIST;
ID          User      Account Role      Profile Routing
=====
1           *          *RB*    *          *          rbRouting
2          mark          *        *          *          readOnlySys1
3         *mktg*        *        *          *          readSys1
4         robert      acct4    dba         *          prefReadOnly
99999       *          *        *          *          DefaultRouting
```

The following example performs a search for a specific user mapping.

```
unityadmin> USER LIST mark;
2          mark          *        *          *          readOnlySys1
```

The following example performs a search for a user mapping that matches wildcards.

```
unityadmin> USER LIST report_mktg_3 acct2 rpt_role;
3         *mktg*        *        *          *          readSys1
```

The following example performs a search with more details.

```
unityadmin> USER LIST robert acct4 dba;
4          robert  acct4  dba      *          prefReadOnly
```

The following example performs a search with more details.

```
unityadmin> USER LIST robert local dba;
99999      *          *          *          *          DefaultRouting
```

## USER UPDATE

### Purpose

Updates or adds a user mapping to a routing rule.

### Syntax

USER UPDATE < *username* > < *accountstring* > < *role* > < *profile* > < *region* > < *routing* > [ TOP | AFTER < *mapId* > | BEFORE < *mapId* > ] ;

### Wildcards

Wildcard Pattern Example	Description
*	Matches any text for the parameter using this wildcard ( <b>username</b> , <b>accountstring</b> , <b>role</b> , <b>profile</b> ).
*usr	Matches any pattern ending with usr.
*usr*	Matches any username containing the text usr.

### Parameters

#### *username*

Name of the user. Name can contain a maximum of 25 characters.

#### *accountstring*

Account string expression to use to determine a match with the account string explicitly stated at logon. It must be an exact match to the target user mapping.

#### *role*

Role of the user mapping that is used to determine the match to the role assigned to the logon by the authorization provider (only applicable to external authentication mechanisms) .

You can use the wildcard \* to match arbitrary role values.

### ***profile***

Profile of the user mapping that is used to match to the profile assigned to the logon by the authorization provider (only applicable to external authentication mechanisms).

### ***region***

Specifies either the name of a Unity region or asterisk (\*) to indicate any region. Use this parameter to determine a match with the logon Endpoint's region. If you do not specify a region, the default asterisk (\*) is used to specify any region.

### ***routing***

Name of the routing rule that identifies the routing assigned to the session that corresponds to the logon matching criteria. You can use `ROUTING LIST` to obtain a list of available routing rule definitions.

### **AFTER *mapid***

Adds the user mapping in the priority list after the given *usermapid*. Shift all other user mapping ID numbers accordingly.

### **BEFORE *mapid***

Adds the user mapping in the priority list before the given *usermapid*. Shifts all other user mapping ID numbers accordingly.

### **TOP *mapid***

Positions the user mapping at the top of the user mapping priority list, with ID 1. Shifts all other user mappings ID numbers accordingly. The top user mapping is the first mapping considered when determining which routing rule to use for a given user.

## **Usage Considerations**

- If either **TOP**, **AFTER**, or **BEFORE** is not specified, the user mapping is added to the end of the user mapping priority list, but before the default mapping values.
- Creates a new user mapping rule that consists of an optional **REGION** parameter, four logon matching field values, the routing assigned to the user mapping, and a mapping priority option. If another rule with the same matching criteria (region, username, accountstring, role, and profile) already exists, the existing rule is updated with the routing and priority you specify in the command.
- **TOP | AFTER *mapid* | BEFORE *mapid*** is a designation of the mapping rules precedence in the list. Using TOP gives the rule the highest precedence. Using AFTER/BEFORE *mapid* places it either after or before the rule identified by the <mapid> parameter. If no precedence is indicated then the user mapping is placed at the end of the list but with precedence over the default user mapping.



- You can create a default rule for a region using a specific region and wildcards for all other criteria. For example, `USER UPDATE REGION r1 * * * * Region1DefaultRouting BEFORE 9999;`

You can use the wildcard `*` to match arbitrary values.

## Example

At the unityadmin prompt:

Input	Description
<code>USER UPDATE mark * * * readOnlySys1;</code>	The following example demonstrates how to create a user mapping for user Mark to use the read-only routing rule:
<code>USER UPDATE *mktg* * * * readSys1;</code>	The following example demonstrates how to create a user mapping for all marketing users to user the "read from system 1" routing rule:
<code>USER UPDATE * *RB* * * rbRouting TOP;</code>	The following example demonstrates how to create a user mapping for all account string containing RB to use the rbRouting routing rule. This should be the first rule to match:
<code>USER UPDATE robert acct4 dba * prefReadOnly AFTER 3;</code>	The following example demonstrates how to create a user mapping after the current user map 3:

# USERMAPPING EXPORT

## Purpose

Exports the current user mapping information to the indicated file.

## Syntax

`USERMAPPING EXPORT< filename>`

## Example

```
unityadmin> usermapping export 'mappings.unityadmin';
Successfully exported to file 'mappings.unityadmin'
```

# Additional Information

## Audience

This guide is intended for use by:

- Database administrators
- System administrators
- Software developers, production users, and testers

The following prerequisite knowledge is required for this product:

- Dual-active systems
- Teradata Database
- Teradata system hardware

## Changes and Additions

Date and Release	Release	Description
September 2020	17.00	<ul style="list-style-type: none"> <li>• Added support for CDM Streaming.</li> <li>• Added support for authentication with LDAP.</li> <li>• Added OBJECT RESYNC, OBJECT SET SYNC USER, DATABASE SET SYNC USER, and OBJECT SET SYNC WHERE CLAUSE commands.</li> </ul>
May 2020	16.51	<ul style="list-style-type: none"> <li>• Added the description of the @xplain feature which provides various pieces of information about a query.</li> <li>• Added support for Data Obfuscation Mode.</li> <li>• Added DATABASE EXCLUDE ADD, DATABASE EXCLUDE LIST, and DATABASE EXCLUDE REMOVE commands.</li> <li>• Added OBJECT SET REPLICATION command.</li> </ul>
January 2020	16.50	<ul style="list-style-type: none"> <li>• Added DICTIONARY MIGRATE TO FILE SYSTEM, LOAD INFO DETAILED, LOAD INFO RECOVERY, and RECOVERY LOG USAGE commands.</li> <li>• Added SSO support.</li> </ul>
December 2019	16.20.46	Removed support for REGION MOVE command.
May 2019	16.20.33	<ul style="list-style-type: none"> <li>• Added Dispatcher Failover considerations.</li> <li>• Added additional ENDPOINT configuration parameters.</li> <li>• Updated CDM considerations.</li> <li>• Added recommendations for <b>dynamicDFSIOTimeouts</b> parameter.</li> </ul>

Date and Release	Release	Description
		<ul style="list-style-type: none"> <li>Updated SESSION RECOVER HALT syntax.</li> </ul>
December 2018	16.20.27	Added Time Windows and support for additional data types.
October 2018	16.20.23.00	Added support for using the Unity UI.
April 2018	16.20.07	Added support for Teradata Database 16.20.

## Teradata Links

Link	Description
<a href="https://docs.teradata.com/">https://docs.teradata.com/</a>	Search Teradata Documentation, customize content to your needs, and download PDFs. Customers: Sign in to access Orange Books.
<a href="https://support.teradata.com">https://support.teradata.com</a>	One-stop source for Teradata community support, software downloads, and product information. Log in for customer access to: <ul style="list-style-type: none"> <li>Community support</li> <li>Software updates</li> <li>Knowledge articles</li> </ul>
<a href="https://www.teradata.com/University/Overview">https://www.teradata.com/University/Overview</a>	Teradata education network
<a href="https://support.teradata.com/community">https://support.teradata.com/community</a>	Link to Teradata community

## Related Documentation

Title	Publication ID
<i>Teradata® Viewpoint Installation, Configuration, and Upgrade Guide for Customers</i> Describes how to install Viewpoint software, configure settings, and upgrade a Teradata Viewpoint server.	B035-2207
<i>Teradata Vantage™ - Advanced SQL Engine Security Administration</i> Describes how to formulate, implement, and audit security policy for a Teradata system.	B035-1100