Hyperion® Essbase® MaxL

Release 6.1.0

User's Guide

Hyperion Solutions Corporation
1344 Crossman Avenue
Sunnyvale, CA 94089

Contents

Appendix A: Error and Status Messages 67

Contents

Introduction to MaxL

This document is a guide for using MaxL, multi-dimensional database access language for Hyperion Essbase OLAP Server. MaxL is a flexible way to automate Hyperion Essbase administration and maintenance tasks.

MaxL and its components enable you to perform database-administrative operations by issuing MaxL statements to a Hyperion Essbase server.

This chapter provides an overview of MaxL and its components. It contains the following sections:

- "The MaxL Language" on page 7
- "The MaxL Command Shell" on page 8
- "The Essbase Perl Module (Essbase.pm)" on page 9

The MaxL Language

The MaxL language is a means for making administrative requests to Essbase using a few simple *statements*, instead of a long series of commands or functions. MaxL operates on the Hyperion Essbase system using an intuitive, SQL-like language paradigm.

Using MaxL, you can easily automate administrative operations on Hyperion Essbase databases.You can write MaxL scripts which are easy to customize and re-use. A MaxL script contains a login and a sequence of MaxL statements, each terminated by a semicolon.

MaxL statements begin with a verb, and consist of grammatical sequences of keywords and variables.

A single MaxL statement looks similar to an English sentence; for example,

```
create or replace user <user-name> identified by <password>;
```

In order for Hyperion Essbase to receive and parse the MaxL statements, you must "pass" them to the Essbase server using either the MaxL Command Shell (essmsh) or a Perl program that uses the Essbase Perl extension.

For more information about the MaxL language, see Chapter 2, "About the MaxL Language," and the online *MaxL Language Reference* in your DOCS directory.

The MaxL Command Shell

The MaxL Command Shell (essmsh in the bin directory) is a native command-line interpreter that can be run interactively or in batch mode. This shell has many advanced features similar to those found in UNIX shells, including script nesting, output logging, shell escapes, command-line arguments, variable interpolation, and the ability to accept input from the output of other programs.

As with ESSCMD, the MaxL Command Shell enables you to work interactively or in batch mode. The MaxL Command Shell has many advanced features to offer, and is easier to learn than ESSCMD. The MaxL Command Shell is an efficient choice for performing automated Essbase tasks using MaxL.

The following is an example of a MaxL script, sent to Essbase via the MaxL Command Shell. This script creates a user, creates a filter, and then assigns the filter to the user. Note that all MaxL scripts must begin with a login to the Essbase system, which must be running.

```
login admin identified by systempasswd on Esshost;
create user Fiona identified by sunflower;
create filter Sample.Basic.Diet read on '@idesc(Diet)';
grant filter Sample.Basic.Diet to Fiona;
logout;
exit;
```

For more information about the MaxL Command Shell, see Chapter 3, "The MaxL Command Shell," and the online *MaxL Language Reference* in your DOCS directory.

The Essbase Perl Module (Essbase.pm)

With the the aid of the Essbase Perl Module (Essbase.pm), the MaxL language becomes embeddable in Perl programs.

Essbase.pm, located in the `Perlmod` directory, enables beginning or advanced Perl programmers to wrap MaxL statements in Perl scripts. In this way, database administration with MaxL becomes as efficient and flexible as your Perl programs are.

Using Perl with MaxL enables you to take advantage of these and other programmatic features while you administer Hyperion Essbase databases:

● Conditional testing

● Inter-process communication

● Message handling

● E-mail notification

● Web scripting

The Perl module (`Essbase.pm`), contains methods that enable you to pass MaxL statements by means of Perl. These methods are:

connect ( ), which establishes a connection to Hyperion Essbase.

do ( ), which tells Perl to execute the enclosed MaxL statement.

pop_msg ( ), which navigates through the stack of returned MaxL messages.

disconnect ( ), which terminates the connection to Hyperion Essbase.

➤ To make the Perl methods available to Hyperion Essbase, you must include a reference to `Essbase.pm` in your Perl program. Place the following line at the top of each Perl script:

```
use Essbase;
```

Perl is not difficult to learn, especially if you have knowledge of UNIX shells or other programming languages. To download Perl and learn more about Perl, visit the Comprehensive Perl Archive Network Web site at `http://www.cpan.org/`.

For information and examples about installing and using Essbase.pm, see the online *MaxL Language Reference* in your `DOCS` directory, and the `README` file located in your `PERLMOD` directory.

About the MaxL Language

MaxL is the new data definition language for Hyperion Essbase OLAP Server. This intuitive, linguistic interface to Essbase helps you automate administrative operations on Hyperion Essbase databases. MaxL also enables programmatic management of Hyperion Essbase applications without a prerequisite knowledge of C, Visual Basic, or the Essbase API.

The MaxL Command Shell is an interpreter that enables you to log in and out of Essbase, issue MaxL statements at the command line, and direct the handling of process ouptut. For more information, see Chapter 3, "The MaxL Command Shell."

This chapter contains the following sections:

About Statements

The MaxL language, like English, has a lexicon of various types of words that can be strung together to create statements. A *statement* in MaxL is a complete direction telling Hyperion Essbase to perform an action. Statements are passed from the MaxL Command Shell or from a Perl program to the MaxL parser. The MaxL parser then "reads" the statements and passes them to the Hyperion Essbase server.

Here is one example of a MaxL statement:

```
create or replace user Fiona identified by sunflower;
```

Overview: Verbs and Objects

A MaxL statement always begins with a verb, such as *create* or *alter*. Verbs indicate what type of operation you want to perform. After the verb, you specify an object. Objects, such as *database* or *user*, indicate the database elements upon which you want to perform actions. To get an overall picture of what MaxL statements can do, see Figure 1, which illustrates verb-object combinations.

Figure 1: Verbs and their Objects in the MaxL Language
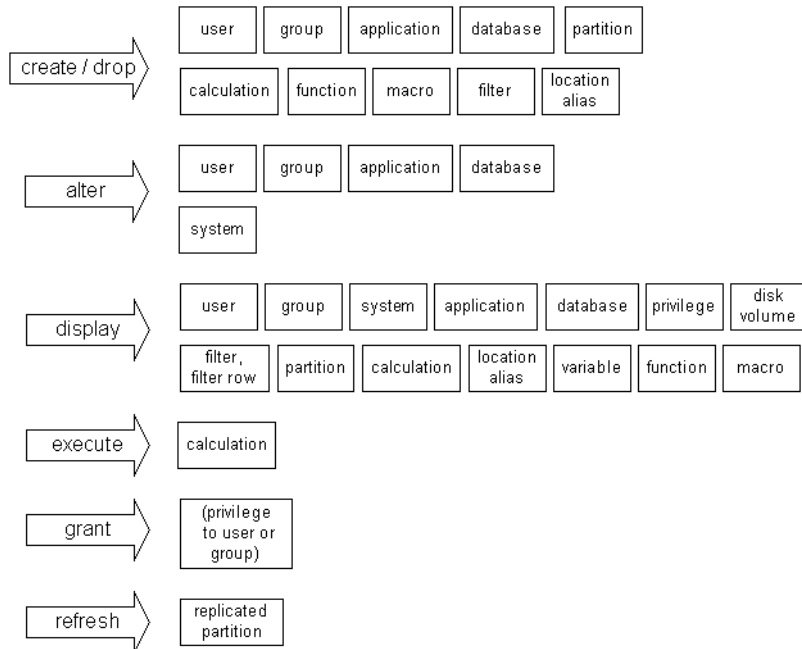
After the object specification, the rest of the MaxL statement is for giving more details about the action you wish to perform. This is done using a grammatically correct sequence of MaxL statement-parts, or *tokens*.

Construction of Statements

The MaxL parser recognizes and requires an ordered presentation of tokens, which are the constituents of statements. A token is a space-delimited sequence of valid characters that is recognized by MaxL as a single readable unit. Tokens can be any of the following:

- Keywords

- Names

- Strings

- Numbers

Keywords

Keywords are the reserved words that make up MaxL's regular vocabulary. These include a number of verbs, objects, and other words commonly used to construct statements. Keywords in MaxL are independent of your data: conversely, all other MaxL tokens (names, for example) must be defined by you.

The MaxL parser expects to see MaxL keywords and other tokens in their correct grammatical order, as diagrammed in the online *MaxL Language Reference*.

In the following sample grammar diagram from the online *MaxL Language Reference*, only the lower-cased words in the diagram represent keywords. The other elements are placeholders for names or values that you provide.

Note: Keywords are case-insensitive.

Figure 2: Example of MaxL Syntax Diagram: Create User
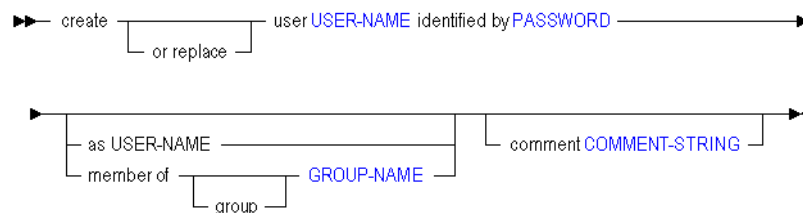
For a complete list of keywords, see the Index of Keywords in the online *MaxL Language Reference*.

Names

Names in MaxL are used to uniquely identify databases and database objects, such as users, applications, or filters.

Rules for Names

Unless you enclose it within single quotation marks, a MaxL name is a string that must begin with an alphabetic character or the underscore. Unquoted names may contain only alphabetic characters, numbers, and the underscore.

When enclosed in single quotation marks, a name may contain white space and any of the following special characters:

. , ; : % $ " ' * + - = < > [ ] { } ( ) ? ! / \ | ~ ` # & @ ^

Note: Any name that is also a MaxL keyword must be enclosed in single quotation marks. For a list of keywords, see the Index of Keywords in the online *MaxL Language Reference*.

Examples:

The following application names *do not* require single quotation marks:

Orange

Orange1

_Orange

The following application names *do* require single quotation marks:

`Orange County` (contains a space)

`1orange` (begins with a number)

`variable` (is a MaxL keyword)

Types of Names

Some Essbase objects have single names, and some require compound names known as *doubles* and *triples*, which express the nesting of namespaces. The three possible ways to name Essbase objects are with a singleton name, a double, or a triple.

A *singleton name* is a name that can stand alone and be meaningful in a system-wide context. This type of name is used for Hyperion Essbase entities that can be referred to independently. For example, an application has a singleton name.

A *double* is two names connected by a period, and a *triple* is three names connected by two periods. Doubles and triples show the inherited namespace of the named entity. For example, a database is always referred to using two names. The first name identifies the application in which the database resides. The second name is the database's own name.

Database objects, such as filters, are usually referred to using triples: the first two names identify the application and database, and the third name is the object's own name. For example, a filter might have the name `sample.basic.filter3`.

The following table shows what type of name is required for each database object, and provides an example of the name used in a statement.

Object	Name	Example
User	singleton	`create user `**`Fiona`**` identified by sunflower;`
Group	singleton	`alter user Fiona add to group `**`Gardening`**`;`
Host	singleton	`drop replicated partition Samppart.Company from Sampeast.East at `**`EastHost`**`;`
Application	singleton	`create application '`**`&New App`**`';`
Database	double	`display database '`**`&New App`**`'.`**`testdb`**`;`
Calculation	triple	`drop calculation `**`sampeast.'2nd'.calcA`**`;`
Filter	triple	`display filter row `**`sample.basic.filter1`**`;`
Function (local)	double	`drop function `**`sample.'@COVARIANCE'`**`;`
Function (global)	singleton	`create function `**`'@JSUM'`**` as 'CalcFnc.sum';`
Location alias	triple	`drop location alias `**`Main.Sales.EasternDB`**`;`

Object	Name	Example
Role	singleton	grant **designer** on database Sample.basic to Fiona;
Substitution variable	singleton	alter system add variable **Current_month**; alter system set variable **Current_month** July;
Disk volume	singleton	alter database AP.main1 add disk volume **G**; alter database AP.main1 set disk volume **G** partition_size 200mb;

Strings

Strings are used in MaxL statements to represent the text of comments, member expressions, calculation scripts, and file references. Strings can begin with any valid character. As with names, strings containing whitespace or special characters must be enclosed in single quotation marks.

See "Rules for Names" on page 14 for a list of valid special characters.

The following table shows examples of statement elements that are strings:

Type of String	Example
Password	create user Fiona identified by **sunflower**;
Comment	alter group Financial comment '**Reports due July 31**';
Member expression	create filter sample.basic.filt1 read on '**Sales, @ATTRIBUTE(Bottle)**'; create or replace replicated partition sampeast.east area '@IDESC(East), @IDESC(Qtr1)' to samppart.company mapped globally ('**Eastern Region**') to (**East**);

Body of a calculation	execute calculation
	
	  '"**Variance**"**=@VAR(Actual, Budget);**
	
	   "**Variance %**"**=@VARPER(Actual, Budget);**'
	
	on Sample.basic;

File reference	spool on to '**/homes/fiona/out.txt**';

Numbers

You use numbers in MaxL statements to change certain database settings in Hyperion Essbase. For example, you can change cache and buffer sizes, or set system-wide intervals such as the number of days elapsing before users are required to change their passwords. To change numeric settings, you can use positive and negative integers, positive and negative real numbers, and zero. Decimals and scientific notation are also permitted.

Examples:

```
1000
-22
645e-2
```

For size settings, units must follow the number. Spaces in between numbers and units are optional. Units are case-insensitive, and may include the following: `B/b` (bytes), `KB/kb` (kilobytes), `MB/mb` (megabytes), `GB/gb` (gigabytes), and `TB/tb` (terabytes). If no units are given, bytes are assumed.

Examples:

```
1000 b
5.5GB
645e-2 mb
145 KB
2,000e-3TB
```

Anatomy of Statements

The following diagrams are templates illustrating some possible MaxL statements and their constituent parts. In the diagrams, lower-cased words are keywords, and words in upper-case are to be replaced with the appropriate values as shown in the example following each illustration.

Altering a Database: Lock Timeout
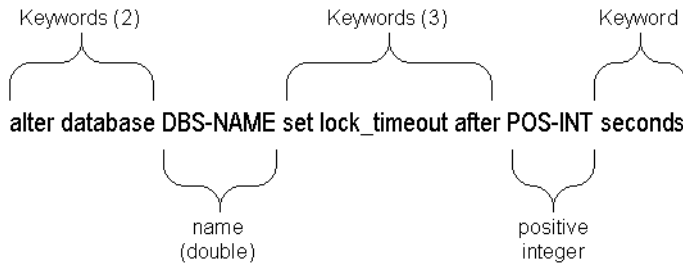
Figure 3: MaxL statement: setting the lock timeout interval

Example:

```
alter database Sample.Basic set lock_timeout after 180 seconds;
```

Altering a Database: Cache Size
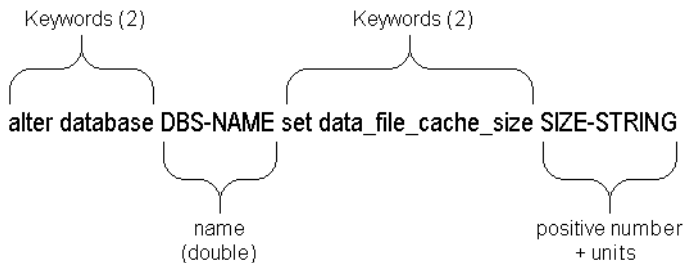
Figure 4: MaxL statement: Changing data file cache size

Example:

```
alter database Sample.Basic set data_file_cache_size 32768KB;
```

Granting Privileges
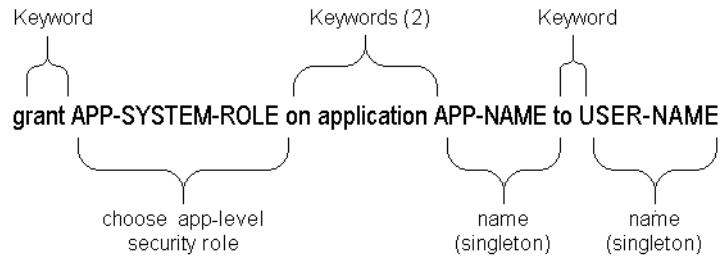
Figure 5: MaxL statement: Granting application privileges to a user

Example:

```
grant designer on application Sample to Fiona;
```

Creating a Calculation
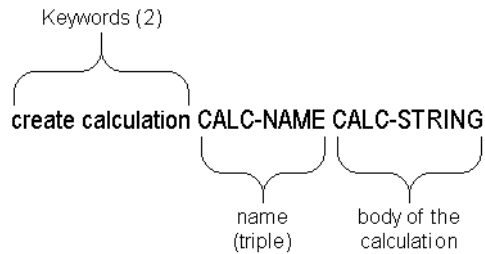
Figure 6: MaxL statement: Creating a stored calculation

Example:

```
create calculation sample.basic.varcalc
'"Variance"=@VAR(Actual, Budget);
"Variance %"=@VARPER(Actual, Budget);'

;
```

The MaxL Command Shell

The MaxL Command Shell (`essmsh` in your `Bin` directory) is a pre-processor to the MaxL language interface to Hyperion Essbase. You can use the MaxL Command Shell to run MaxL scripts, or to enter MaxL statements interactively at the command line. MaxL scripts automate many system-administrative operations for which you might have used ESSCMD in the past. In addition, within any MaxL session, you can:

- Include other MaxL scripts. See "Nesting MaxL Scripts" on page 26.

- Record the entire MaxL session by spooling output to a file. See "Spooling Output to a File" on page 27.

- Include operating-system commands. See "Including Operating-System Commands" on page 27.

- Reference ESSCMD scripts, if necessary. See "Using ESSCMD from within MaxL" on page 27.

- Use variables, so that the same MaxL script can be used for many situations. See "Using Variables to Customize MaxL Scripts" on page 28.

This chapter shows you how to get started using the basic features of the MaxL Command Shell. It contains the following sections:

- "Starting the MaxL Command Shell" on page 22

- "Logging In to Hyperion Essbase" on page 25

- "Command Shell Features" on page 26

- "Stopping the MaxL Command Shell" on page 28

Starting the MaxL Command Shell

The MaxL Command Shell can be invoked to take input in any of the following ways:

- Interactively, from the keyboard

- From a MaxL script file (statements are read from the file specified on the command line)

- From standard input that is piped to the MaxL Command Shell from the output of another program

The MaxL Command Shell also accepts any number of command-line arguments at invocation time. These can be used with positional parameters to represent any name, or a password.

Starting the Shell for Interactive Input

➤ To enter MaxL statements interactively at the command line, simply invoke the command shell at your operating-system prompt. Text you type is indicated by **bold** text.

For example,

essmsh

```
Hyperion Essbase MaxL Command Shell Version 6.1.0.
(c) Copyright 2000 Hyperion Solutions Corporation.
All rights reserved.
```

➤ To enter MaxL statements interactively after logging in at invocation time, use the -l flag. For example,

essmsh -l Fiona sunflower

```
Hyperion Essbase MaxL Command Shell Version 6.1.0.
(c) Copyright 2000 Hyperion Solutions Corporation.
All rights reserved.

49 - User logged in: [fiona].
```

➤ To enter MaxL statements interactively and also supply command-line arguments to represent variables you will use in your interactive session, use the -a flag. For example,

`essmsh -a Fiona sunflower Sample Basic`

```
Hyperion Essbase MaxL Command Shell Version 6.1.0.
(c) Copyright 2000 Hyperion Solutions Corporation.
All rights reserved.
```

`login $1 $2;`

```
49 - User logged in: [fiona].
```

`alter database $3.$4 enable aggregate_missing;`

```
72 - Database altered: ['sample'.'basic'].
```

In the above example, $1, $2, $3, and $4 are positional parameter variables: they represent whatever arguments were entered after **essmsh** at invocation time, in the order they were entered.

Starting the Shell for File Input

➤ To invoke the MaxL Command Shell to take input from a MaxL script file, type **essmsh** followed by the name of a MaxL script in the current directory, or, the full path and file name of a MaxL script in another directory.

If you provide only a file name, the MaxL Command Shell assumes that the file is in the current directory (the directory the operating-system command prompt was in when essmsh was invoked). In the following invocation example, the file maxlscript.msh must be in C:\.

```
C:\> essmsh maxlscript.msh
```

If the MaxL script is not in the current directory, provide a path to the MaxL script. You can use absolute paths or relative paths.

For example,

```
$ essmsh ../hyperion/essbase/test.msh
```

Note: MaxL scripts are not required to have any particular file extension, or any file extension at all. This document uses .msh.

In UNIX shells, you should place single quotation marks around the path to avoid file-reading errors.

In the Windows command prompt, if the path to the script contains a space, you may have to use double quotation marks around the entire path and file name to avoid file-reading errors.

Starting the Shell for Programmatic Input

➤ To invoke the MaxL Command Shell to take input from the standard output of another program or process, use the -i flag. For example,

```
program.sh | essmsh -i
```

The shell script program.sh may generate MaxL statements as output. The shell script's output is piped to essmsh -i, which uses that output as its input. This allows for efficient co-execution of scripts.

Windows Example

The following Windows NT batch script generates a login statement and a MaxL display statement as its output. The -i flag enables that output to be used by essmsh, the MaxL Command Shell, as input.

```
echo login Fiona sunflower on localhost; display privilege
user;|essmsh -i
```

User Fiona is logged in, all user privileges are displayed, and the MaxL session is terminated.

UNIX Example

The following portion of a shell script ensures that there are no applications on the system, by testing whether **display application** returns zero applications.

```
if [ $(echo "display application;" | essmsh -l admin password1
-i | \
       awk '/53 - Records returned:/ {print $5}' ) != "[0]." ]
then
    print "This test requires that there be no applications on
the system"
    print "Quitting"
   exit 2

fi
```

Here is how the above example works:

1. MaxL grammar is piped to a MaxL Command Shell invocation and login, as the output of the UNIX `echo` command.

2. The results of the MaxL session are tested by `awk` for pattern-matching with the MaxL status message you would get if you entered **display application** on an empty system: `53 - Records returned: [0]`.

3. `Awk` matches the string `'Records returned: '`, and then it checks to see if that is equal to `'[0].'`

4. If `$5` (a variable representing the fifth token `awk` finds in the status string) is equal to `'[0].'`, then there are no applications on the system; otherwise, `$5` would equal `'[1].'` or whatever number of applications exist on the system.

For more information and examples on invocation options, see the online *MaxL Language Reference* in your `DOCS` directory. This information is also contained in the essmsh "man page." To view the man page, enter **essmsh -h | more** at the operating-system command prompt.

Logging In to Hyperion Essbase

The MaxL language interpreter requires a connection to a Hyperion Essbase session before it can begin parsing MaxL statements. Use the MaxL Command Shell to establish the connection to Hyperion Essbase.

➤ To log in to Hyperion Essbase after the command shell has been started, use the shell's **login** grammar. Text you type is indicated by **bold** text.

For example,

essmsh

```
Hyperion Essbase MaxL Command Shell Version 6.1.0.
(c) Copyright 2000 Hyperion Solutions Corporation.
All rights reserved.
```

MAXL>**login Fiona identified by sunflower on hostname;**

If a host name is not specified, localhost is assumed.

➤ To log in when you invoke the shell, use the **-l** option. To log in to a server besides `localhost` at invocation time, use the **-s** option. For example,

```
essmsh -l fiona sunflower -s dtemp1
```

Note: You can log out and change users without quitting the shell.

For more information about the syntax for invocation options, see the online *MaxL Language Reference* in your DOCS directory.

Command Shell Features

The MaxL Command Shell includes command-line argument processing, environment variable processing, nesting of MaxL scripts, and shell escapes. These features offer the flexibility needed to create a highly automated Essbase production environment.

For more information on the syntax and usage of the following features, see the online *MaxL Language Reference* in your DOCS directory.

Nesting MaxL Scripts

As a database administrator, you may wish to save your separate automated tasks in several MaxL scripts, rather than executing many operations from a single script. Putting the pieces together is a simple task if you know how to reference one MaxL script from another.

➤ To reference or include other MaxL scripts within the current MaxL session, use the following MaxL Command Shell syntax:

```
msh <scriptfile>;
```

Here is an example of nested MaxL scripts:

```
/* unarchive.msh */
login admin password9;
alter database sample.basic
 end archive;
msh calculate.msh;

alter database sample.basic
 begin archive to file bak;
logout;
```

```
/* calculate.msh */
login admin password9;
 execute calculation
 'SET MSG ERROR; CALC ALL;'
 on sample.basic;
```
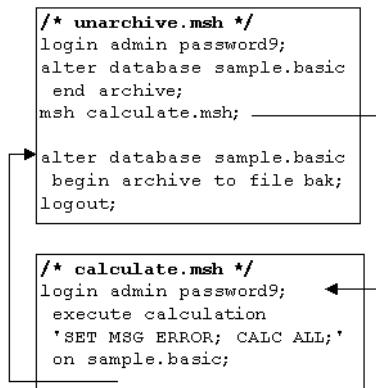
Figure 7: Nesting MaxL Scripts

Spooling Output to a File

You can create a log file of all or part of a MaxL session and its associated messages by spooling output to a file.

➤ To record a MaxL session, complete the following tasks:

1. Log in to Hyperion Essbase. For example,

```
login fiona sunflower;
```

2. Begin spooling output, using **spool on to <filename>**. For example,

```
spool on to 'c:\output\display.txt';
```

3. Enter as many MaxL statements as you want recorded.

4. Stop spooling output, using **spool off**;

Including Operating-System Commands

You can issue operating-system commands directly from a MaxL session. The operating-system output becomes part the MaxL Command Shell's output, which may be logged to a file. When the operating system finishes executing whatever commands are issued, it returns control to essmsh.

➤ To escape to the operating system from within a MaxL session, use **shell**. For example,

```
login fiona sunflower;
shell date;
logout;
```

Using ESSCMD from within MaxL

You can reference ESSCMD scripts from within MaxL scripts, if needed. You might do this to perform outline management tasks, which are not yet available in Release 6.1.0 of MaxL.

➤ To escape to an ESSCMD session, use **shell**. For example,

```
login fiona sunflower;
shell esscmd ../scripts/test.scr;
```

Using Variables to Customize MaxL Scripts

In the MaxL Command Shell, you can use variables as placeholders for any data that is subject to change or that you refer to often; for example, the name of a computer, user names, and passwords. You can use variables in MaxL scripts as well as during an interactive session. Using variables in MaxL scripts eliminates the need to create many customized scripts for each user, database, or host. Variables can be environment variables (for example, $ARBORPATH, which references the directory Essbase is installed to), or positional parameters (for example, $1, $2, etc.). A variable always begins with a $ (dollar sign).

For more information about syntax and usage of variables in the MaxL Command Shell, please refer to the online *MaxL Language Reference* in your DOCS directory.

Stopping the MaxL Command Shell

You can log out of a MaxL session, and optionally log in as another user, without quitting the shell. You should include a logout command at the end of MaxL scripts. It is not necessary to exit at the end of MaxL script files, or after a session using stream-oriented input from another program's output.

➤ To log out without exiting the MaxL Command Shell, enter

```
logout;
```

➤ To exit from the MaxL Command Shell after using interactive mode, enter

```
exit;
```

Security Management With MaxL

To effectively manage Hyperion Essbase security using MaxL, you should develop for your system the best combination of the following approaches:

- Grant privileges and roles to users and groups. When higher than global access settings applied to applications or databases, these user-assigned privileges and roles take precedence.

- Set minimum global permissions at the application or database level. Users and groups inherit these minimum settings, except when their granted privileges are higher, or when they have been granted higher access by means of a filter.

- Grant filters to users and groups. Filters define database access levels for particular database members, down to the individual data value (cell). Because they are more granular, filters take precedence over user-assigned and global settings.

This chapter contains the following sections:

About Privileges and Roles

Hyperion Essbase permissions consist of *privileges*, which are system access types, and *roles*, which are logical groups of privileges that typically belong together. Roles can be thought of as privilege levels that are grantable to users and groups. Some privileges can also be granted, independently of roles. For example,

- The role of **write** includes write privilege and read privilege. Granting **write** to user Fiona gives her read and write access.

- The **create_user** privilege is grantable independently of any role (but is also included in the **supervisor** role). Granting **create_user** to Fiona enables her to create and delete users, but not to access or modify any databases.

The *scope* of a permission refers to the area of data it encompasses. A permission's scope can be the system, an application, or a database. For example,

- The role of **supervisor** applies to the entire system; therefore, it also applies to all applications and databases.

- The role of application **designer** applies to an entire application; therefore, it also applies to all databases within the application.

For more information about

Permissions Available

The following table describes all security privileges and roles, and their scopes.

Table 1: Roles and System Privileges that can be Assigned

Role or Privilege	Affected Scope	Access Description
no_access (role)	Entire system, application, or database	No access of any kind to the assigned scope, unless access is available globally or by means of a filter.
read (role)	Database	Ability to read data values.
write (role)	Database	Ability to read and update data values.
execute (role)	Entire system, application, database, or single calculation	Ability to calculate, read, and update data values for the assigned scope, using the assigned calculation. Supervisors, application designers for the application, and database designers for the database can run calculations without being granted execute access.

Role or Privilege	Affected Scope	Access Description
designer (role)	Application, if assigned at that level	Ability to create, delete, and modify databases within the application (Application Designer).
designer (role)	Database, if assigned at that level	Ability to modify outlines, create and assign filters, alter database settings, and remove locks on the database (Database Designer).
create_application (system privilege)	Entire system	Ability to create and drop applications.
create_user (system privilege)	Entire system	Ability to create and drop users and groups.
supervisor (role)	Entire system	Full access to the entire system and all users and groups

The only *atomic* privileges (privileges you can grant directly, rather than as part of a role) are the system privileges **create_application** and **create_user**.

Privileges Contained in Security Roles

To help you understand which privileges you are granting when you grant a role to a user or group, the following table illustrates the hierarchy of security roles and shows the Essbase system privileges that are contained in each role.

The left column, MaxL System Roles, contains keywords in **bold**. For more information about MaxL keywords, and the syntax for granting roles, see the online *MaxL Language Reference* in your `Docs \Techref\MaxL` directory.

Table 2: Privileges Contained in Essbase Security Roles

MaxL System Roles	Essbase System Privileges								
	read	write	calculate	design database	create database	start application	design application	create\drop application	create\drop user
supervisor	■	■	■	■	■	■	■	■	■
designer (application)	■	■	■	■	■	■	■		
designer (database)	■	■	■	■					
execute	■	■	■						
write	■	■							
read	■								
no_access									

Granting User-Level Privileges and Roles

You can create MaxL statements beginning with the **grant** keyword to assign privileges, roles, or filters to users and groups. This section discusses granting roles, privileges, and calculations.

To learn about granting filters, see "Creating and Granting Filters" on page 36.

Granting Roles

A role is granted to users and groups using the following syntax:

```
grant <system role> [on system]
  | <application role> on application <app-name>
  | <database role> on database <dbs-name>
to <user-name>
  | <group-name>
```

When you grant a role to a user or group at a specified level (system, application or database) that user or group's existing roles are replaced. For example, if user Fiona currently has read access to Sample.Basic, then the following statement removes her read access and gives her no access:

```
grant no_access on database sample.basic to Fiona;
```

Granting Create/Delete Privileges

Create_user is the MaxL keyword for the system privilege to create and delete users and groups. **Create_application** is the MaxL keyword for the system privilege to create and delete applications. These are the only system privileges that you may grant independently of a role (meaning that you can grant the privilege itself, and not necessarily a role containing the privilege).

These privileges correspond to the following "user types" in found in Application Manager:
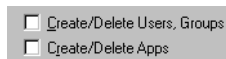
Figure 8: System-level system privileges as shown in New User Dialog Box

When you grant a user or group one of the built-in privileges, **create_user** or **create_application**, existing roles or privileges are appended to, but not replaced.

Granting Calculations

You can grant calculation privileges to users and groups in the following ways:

- Grant execute access for all calculations on an application, database, or system.

- Grant execute access for the default calculation.

- Grant execute access for specific calculations named and stored at the database level.

Granting All Calculations or the Default Calculation

Grant execute any gives the specified user or group permission to execute all calculations, including the default calculation, for the specified scope (application, database or entire system). **Grant execute default** gives permission to execute the default calculation, which is typically CALC ALL; (but you can change it using alter database set default calculation <calc-name>).

Example

➤ To grant Fiona permission to execute all calculations on the Sample application, you could use the following MaxL script:

```
login admin password;
grant execute any on application Sample to Fiona;
logout;
exit;
```

Granting Stored Database Calculations

A user or group may be granted any number of stored calculations per database. Therefore, granting a calculation adds it to the user or group's list of executable calculations.

In MaxL, all stored calculations are named with the triple naming convention, to show database context. They also must reside in the $ARBORPATH\App\Appname\Dbname directory. Therefore, when granting execute access for a stored calculation (CALC-NAME), you need not specify the scope, as you would when granting execute access to all calculations.

Example

The following calculation has been created on the Sample Basic database. The name of the calculation is a triple, which shows that it belongs to the database.

Note: For information about naming conventions, see "Names" on page 14.

Calculation name	Calculation content
`sample.basic.'var.csc'`	`"Variance"=@VAR(Actual, Budget);`
	`"Variance %"=@VARPER(Actual, Budget);`

➤ To grant `var.csc` to Fiona, you could use the following MaxL script:

```
login admin password;
grant execute sample.basic.'var.csc' to Fiona;
logout;
exit;
```

For more information about the syntax for granting calculations, see the online *MaxL Language Reference* in your DOCS directory.

Setting Minimum Permissions for Applications and Databases

To set a global or minimum level of permission that all users can have (or not have) on an application or database, use **alter application** or **alter database** statements. For example,

```
alter database Sample.Basic set minimum permission read;
```

```
alter application Demo set minimum permission no_access;
```

Minimum permission settings are overridden by filters, and also by higher privileges and roles granted to users or groups.

For example, an application's minimum-permission setting of **no_access** (None in Application Manager) means that no application-level access of any kind is permitted except in the following cases:

● Higher permissions have been granted to a user or group.

● Users have access to a database in the application because the minimum database access setting is higher.

● Access has been granted to a database in the application by means of a filter.

A *database's* minimum-permission setting of **no_access** means that no database-level access of any kind is permitted except in the following cases:

● Higher permissions have been granted to a user or group.

● Access has been granted to a database in the application by means of a filter.

Creating and Granting Filters

➤ To create and edit filters, use the MaxL **create filter** grammar.

For the exact syntax for creating filters, see the *MaxL Language Reference* in the `Docs \Techref\MaxL` directory.

Filtering members separately, using the OR relationship diagrammed below, affects whole regions of data for those members. Filtering member combinations, using the AND relationship diagrammed below, affects data at the member intersections.

Figure 9: How Filters Affect Data: AND/OR Relationships
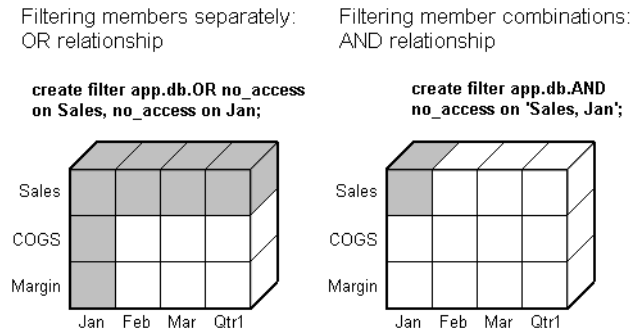
➤ To grant filters, use the MaxL **grant** grammar.

For the exact syntax of **grant**, see the *MaxL Language Reference* in the `Docs \Techref\MaxL` directory.

Granting filters enables the most detailed level control over the access users and groups can have to a database. Access that is granted by filters takes precedence over minimum permission settings and user-granted privileges and roles.

If you have the role of Supervisor, you can define and assign any filters to any users or groups. Filters do not affect you.

If you are a user with Create/Delete Applications privilege, you can assign and define filters for applications you created.

If you have the role of Application Designer or Database Designer, you can define and assign filters within your applications or databases.

MaxL Security Task Table

The following task table lists common Hyperion Essbase security tasks which were not discussed at length in this chapter. The corresponding MaxL statements for completing those tasks are shown on the right. For the exact syntax and usage of MaxL statements, consult the MaxL documentation in the online *Technical Reference* in the DOCS directory. If you did not install the Technical Reference, see the standalone *MaxL Language Reference* in the DOCS\Techref\MaxL directory.

For general information about Hyperion Essbase security, see the *Hyperion Essbase Database Administrator's Guide*.

Tasks	MaxL Statements
Create or copy users or groups	create user, create group
Edit or rename users or groups	alter user, alter group
Delete users or groups	drop user, drop group
View a list of users or groups	display user, display group
View group membership information	display user in group
Enable or disable a user name	alter user
Change a user's password or settings	alter user
Log out a user or all users	alter system
Change global password and login management settings	alter system
Create, copy, or edit filters	create filter
Delete filters	drop filter

MaxL Quick Reference

This chapter contains information to help you get started using MaxL. It contains the following sections:

- "Starting the MaxL Command Shell" on page 39
- "MaxL Syntax Diagrams" on page 40

Starting the MaxL Command Shell

To issue MaxL statements to Essbase using the MaxL Command Shell, first start the Hyperion Essbase 6.1 server. Then invoke the MaxL Command Shell at the operating system prompt, and log in to Essbase. Here is an example of starting an interactive session, where user Fiona is identified by password sunflower:

essmsh

```
Hyperion Essbase MaxL Command Shell Version 6.1.0.
(c) Copyright 2000 Hyperion Solutions Corporation.
All rights reserved.
```

MAXL> **login fiona sunflower on localhost;**

```
49 - User logged in: [fiona].
```

MAXL>

You are now ready to issue MaxL statements interactively. To learn more advanced features of the MaxL Command Shell, see the *MaxL Language Reference*, located in the online *Technical Reference* in the DOCS directory.

MaxL Syntax Diagrams

The syntax for the MaxL language is illustrated in this section using BNF diagrams. Remember to terminate each statement with a semicolon when using the MaxL Command Shell.

For more detailed information and examples of MaxL grammar, or to learn how to read BNF, see the online *Technical Reference* in the DOCS directory.

alter application

Change application-wide settings. Permission required: Application designer.
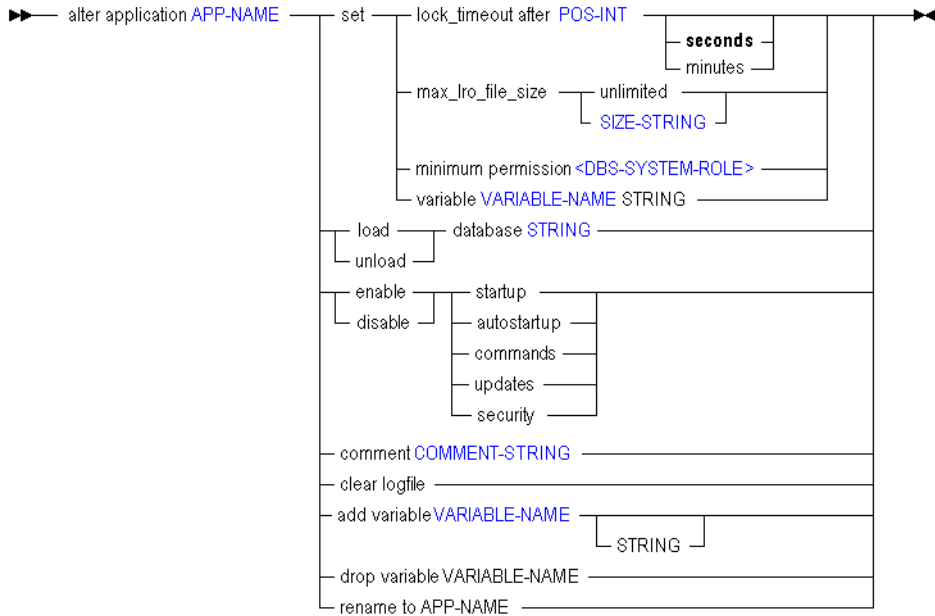
Terminal or <non-terminal>	Example
APP-NAME	Sample
POS-INT (seconds)	60
SIZE-STRING	12MB
<DBS-SYSTEM-ROLE>	no_access, read, write, or designer
VARIABLE-NAME	sample.basic.new_variable
COMMENT-STRING	'Nice application!'

alter database

Change database-wide settings. Permission required: Database designer.
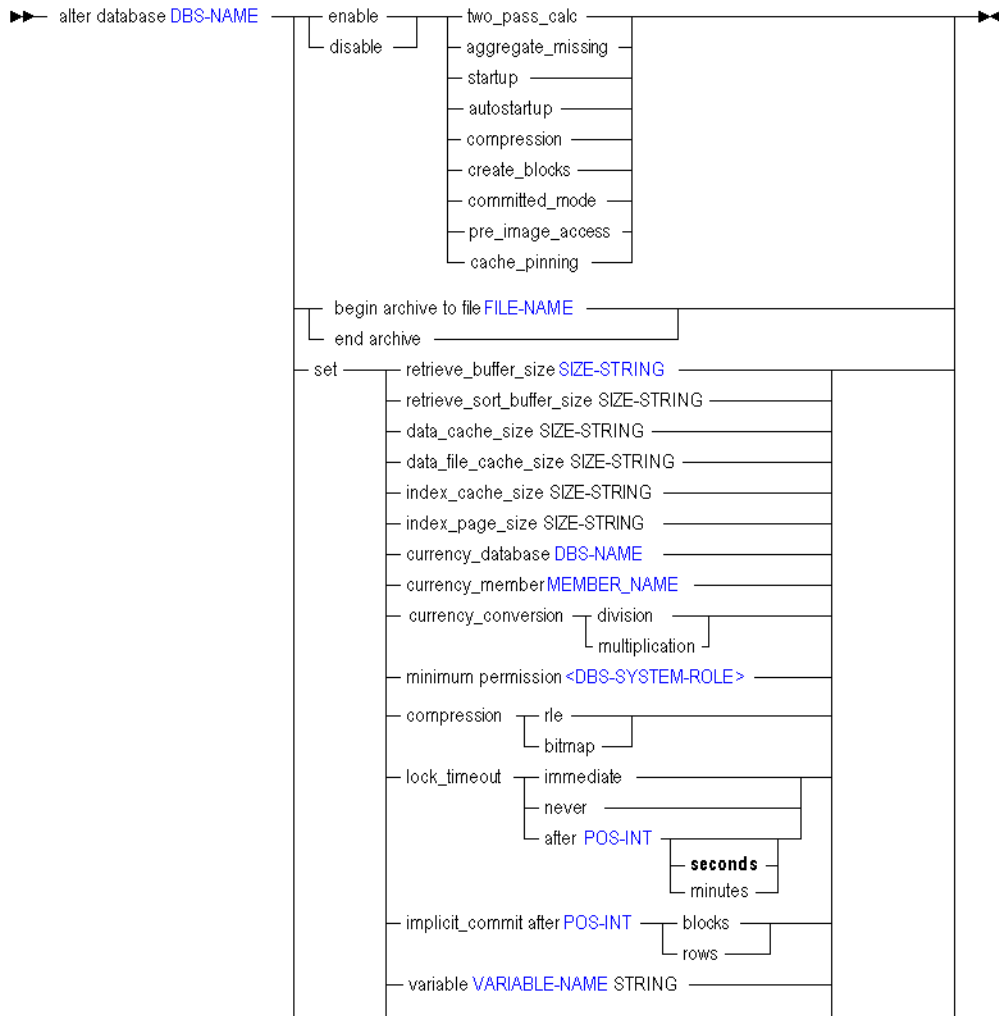
The language diagram continues on next page.

alter database *(Continued)*
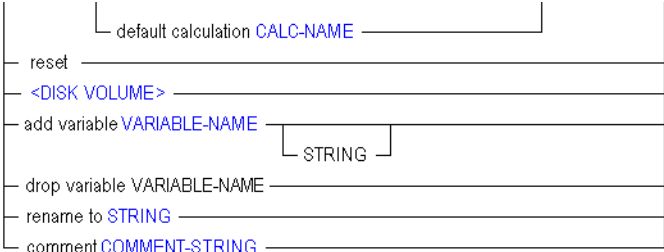
Terminal or <non-terminal>	Example
DBS-NAME	sample.basic
FILE-NAME	'../../archive/basic/archfile'
SIZE-STRING	12MB
MEMBER-NAME	'Act xchg'
<DBS-SYSTEM-ROLE>	no_access, read, write, or designer
POS-INT	60
VARIABLE-NAME	sample.basic.daily_quote
CALC-NAME	sample.basic.calcname
<DISK-VOLUME>	add | drop disk volume sample.basic.c; set disk volume sample.basic.c file_type data;
COMMENT-STRING	'Nice database!'

alter group

Rename a group or change the comment that describes the group. Permission required: create_user.

►►— alter group GROUP-NAME ┬─ rename to group GROUP-NAME ─┬─►◄
                            └─ comment COMMENT-STRING ──────┘

Terminal	Example
GROUP-NAME	supervisors
COMMENT-STRING	'Nice group!'

alter system

Change the state of the Hyperion Essbase server. Start and stop applications, delete application log files, manipulate system-wide variables, manage password and login activity, disconnect users, and shut down the server.
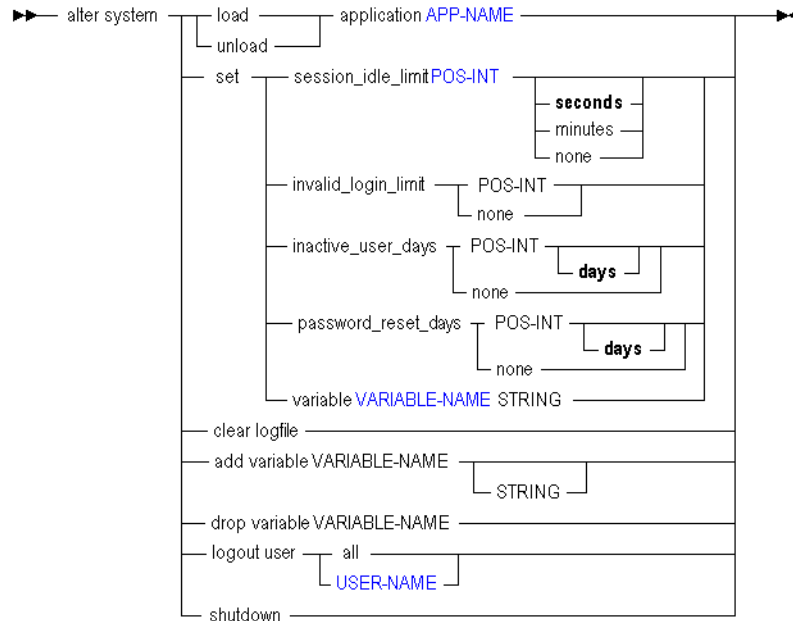
Terminal	Example
APP-NAME	Sample
POS-INT (days)	5
VARIABLE-NAME	sample.basic.new_variable
USER-NAME	Fiona

alter user

Add or remove a user to or from a group. Rename a user. Change the comment that describes a user. Enable or disable a user account. Change a user's password, or specify whether it should expire. Permission required: create_user.
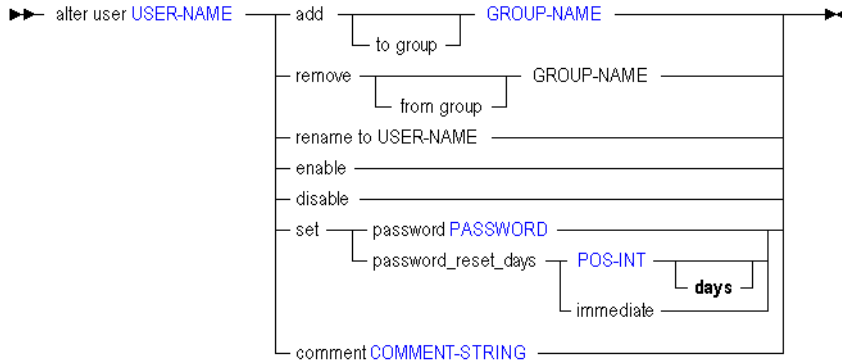
Terminal	Example
USER-NAME	Fiona
GROUP-NAME	supervisors
PASSWORD	'secret word'
COMMENT-STRING	'Username disabled while BK is on vac.'

create application

Create or re-create an application, either from scratch or as a copy of another application on the same system. Permission required: create_app. APP-NAME must consist of 8 or fewer characters. Avoid spaces and special characters when naming applications and databases.

```
►►── create ──┬──────────────┬── application APP-NAME ──┬──────────────┬──┬─────────────────────────┬── ►◄
              └── or replace ─┘                          └── as APP-NAME ─┘  └── comment COMMENT-STRING ─┘
```

Terminal	Example
APP-NAME	Sample
COMMENT-STRING	'Nice application!'

create calculation

Create, replace, or copy a stored calculation.

```
►►── create ──┬──────────────┬── calculation CALC-NAME ──┬── CALC-STRING ──── ►◄
              └── or replace ─┘                           └── as CALC-NAME ─┘
```

Terminal	Example
CALC-NAME	sample.basic.alloc
CALC-STRING	'CALC DIM(Year, Measures, Product);'

create database

Create or re-create a regular or currency database. Optionally create the database as a copy of another database on the same system. DBS-NAME must consist of 8 or fewer characters. Avoid spaces and special characters when naming applications and databases.

Terminal	Example
DBS-NAME	sample.basic
COMMENT-STRING	'Nice database!'

create filter

Create or re-create a database security filter, either from scratch or as a copy of another filter on the same system. Filters control security for database objects. Use **grant** to assign filters to users and groups. Privilege required: Database designer.

Terminal	Example
FILTER-NAME	sample.basic.filt1
MEMBER-EXPRESSION	'@ANCESTORS(Qtr2)'

create function

Create or re-create your own registered Hyperion Essbase function, using a Java method. Privilege required: Database designer.

Process to follow:

1. Develop the functions in Java classes.

2. Use MaxL's **create function** to register them in the Hyperion Essbase calculator framework.

3. Use the functions in the same way that you use the standard Hyperion Essbase calculation functions.

To use this feature, you must have selected to install the Java Virtual Machine with Hyperion Essbase. For more information about creating and using custom-defined functions, see the online *Technical Reference* in the DOCS directory, and the *Hyperion Essbase Database Administrator's Guide*.
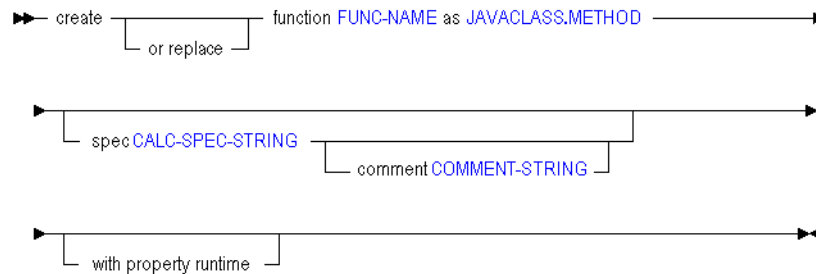
Terminal	Example
FUNC-NAME	Sample.'@COVARIANCE' (application-level function name) or '@COVARIANCE' (global function name)
JAVA	'com.hyperion.essbase.calculator.Statistics.covariance'
CALC-SPEC-STRING	'@COVARIANCE (expList1,expList2)'
COMMENT-STRING	'computes covariance of two sequences given as expression lists'

create group

Create or re-create a group, either from scratch or as a copy of another group.

Terminal	Example
GROUP-NAME	supervisors
COMMENT-STRING	'Nice group!'

create location alias

Create on the database a location alias identifying a host name, database, user name, and password. Location aliases provide a shorthand way of referencing login information for other Hyperion Essbase databases. Permission required: Database designer.

Terminal	Example
NEW-ALIAS-NAME	Eastern
DBS-NAME	this.basic
DBS-NAME	that.basic
HOST-NAME	Aspen
USER-NAME	Fiona
PASSWORD	sunflower

create macro

Create or re-create your own Hyperion Essbase macro using a Java method. You create a macro as your chosen combination of existing calculation functions or macros. Privilege required: Database designer.

To use this feature, you must have selected to install the Java Virtual Machine with Hyperion Essbase. For more information about creating and using custom-defined macros, see the online *Technical Reference* in the DOCS directory, and the *Hyperion Essbase Database Administrator's Guide*.
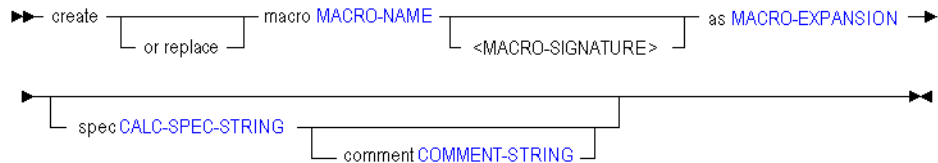
Terminal	Example
MACRO-NAME	Sample.'@COVARIANCE ' (application-level macro name) or '@COVARIANCE' (single, single) (global macro name with optional signature)
MACRO-EXPANSION	'@COUNT(SKIPMISSING,@RANGE(@@S))'
CALC-SPEC-STRING	'@COVARIANCE (expList1, expList2)'
COMMENT-STRING	'computes covariance of two sequences given as expression lists'

create partition

Create a partition definition between two databases. Permission required: Database designer at both sites. The first DBS-NAME is the local database, and the second DBS-NAME is the remote database. Creating a partition *to* the remote site means the current database is the source. Creating a partition *from* the remote site means the current database is the target..
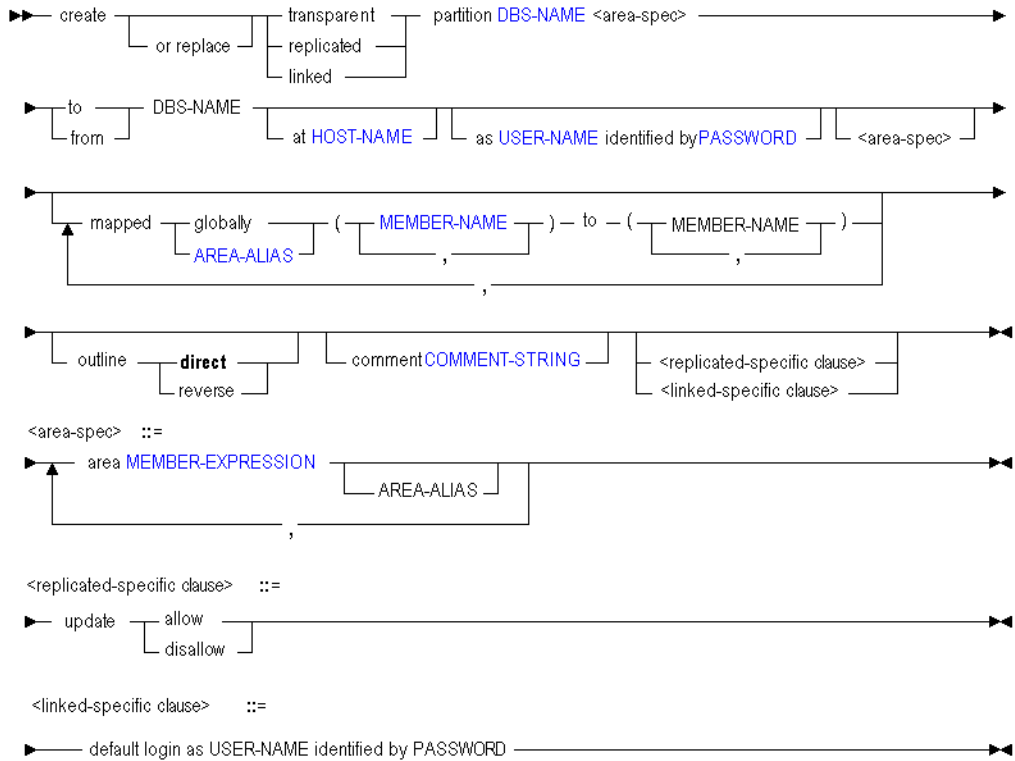
Terminal	Example
DBS-NAME	sampeast.east
HOST-NAME	aspen
USER-NAME	Fiona
PASSWORD	sunflower
AREA-ALIAS	samppart.company.part_alias

Terminal	Example
MEMBER-NAME	'New York'
COMMENT-STRING	'Mapping main to east subcube.'
MEMBER-EXPRESSION	'@IDESC(East), @IDESC(Qtr1)'

create user

Create or re-create a user, either from scratch or as a copy of another user. Optionally create the user as a member of a group.
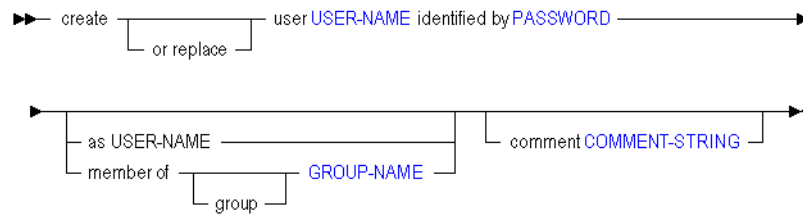
Terminal	Example
USER-NAME	Fiona
PASSWORD	sunflower
GROUP-NAME	newhires
COMMENT-STRING	'admin assistant.'

display application

View information about current application-wide settings.
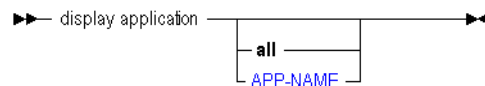
Terminal	Example
APP-NAME	Sample

display calculation

View a list of stored calculations on the system.

```
►►── display calculation ──────────────────►◄
                          ├─ all ─────┤
                          └─ CALC-NAME ─┘
```
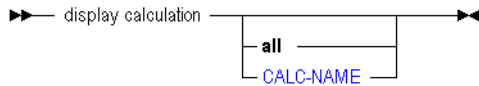
Terminal	Example
CALC-NAME	sample.basic.calcname

display database

View information about current database-wide settings.

```
►►── display database ──────────────────►◄
                        ├─ all ────┤
                        └─ DBS-NAME ─┘
```

Terminal	Example
DBS-NAME	sample.basic

display disk volume

View a list of currently defined disk volume definitions.

```
►►── display disk volume ──────────────────►◄
                           ├─ all ──────┤
                           └─ VOLUME-NAME ─┘
```
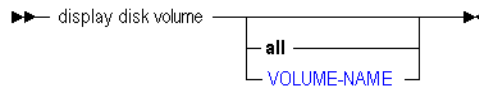
Terminal	Example
VOLUME-NAME	sample.basic.vol3/hyperion/essbase

display filter

View a specific filter or a list of all filters on the system.

```
▶▶── display filter ──┬─────────────┬──▶◀
                      ├─ all ───────┤
                      └─ FILTER-NAME ┘
```

Terminal	Example
FILTER-NAME	sample.basic.filt2

display filter row

View the rows which define database access within a specific filter.

```
▶▶── display filter row ──┬─────────────┬──▶◀
                          ├─ all ───────┤
                          └─ FILTER-NAME ┘
```

Terminal	Example
FILTER-NAME	sample.basic.filt3

display function

View a list of custom-defined functions available to an application. You must specify an application name. Minimum permission required: read.

If MaxL shows no application name next to a function in the display output, then that function is global (system-wide).

```
▶▶── display function APP-NAME ──────────▶◀
```

Terminal	Example
APP-NAME	sample.basic

display group

View a specific group or a list of all groups on the system. To view group membership information, use **display user**.

Terminal	Example
GROUP-NAME	supervisors

display location alias

View a specific location alias or a list of all location aliases defined on the system.

Terminal	Example
LOCATION-ALIAS-NAME	sample.basic.Eastern

display macro

View a list of custom-defined macros available to an application. You must specify an application name. Minimum permission required: read.

If MaxL shows no application name next to a macro in the display output, then that macro is global (system-wide).

Terminal	Example
APP-NAME	sample.basic

display partition

View information about a specific partitioned database or all partitioned databases on the system.
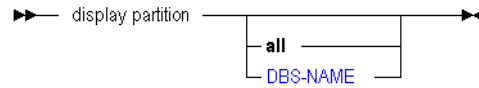
Terminal	Example
DBS-NAME	samppart.east

display privilege

View a list of privileges, calculations, or filters held by users or groups.
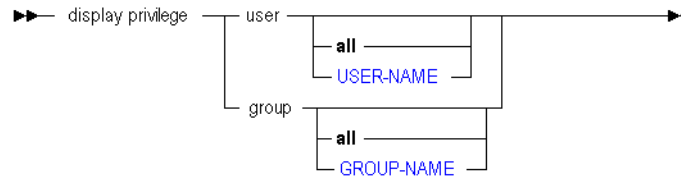
Terminal	Example
USER-NAME	Fiona
GROUP-NAME	field_svcs

display system

View information about current system-wide settings.

display user

View a specific user or a list of all users defined on the system. View group membership information.

```
►►─ display user ─┬──────────────────────────┬──── ►◄
                  ├─── all ──────────┤
                  │  └─ USER-NAME ──┤
                  └─ in group ─┬──────────────┤
                               ├─── all ──────┤
                               └─ GROUP-NAME ─┘
```
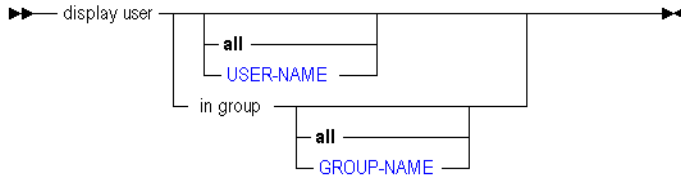
Terminal	Example
USER-NAME	Fiona
GROUP-NAME	field_svcs

display variable

View a list of substitution variables, or any one variable, defined on the system.

```
►►─ display variable ─┬─────────────────┬── ►◄
                      ├─── all ─────────┤
                      └─ VARIABLE-NAME ─┘
```

Terminal	Example
VARIABLE-NAME	sample.basic.current_month

drop application

Delete an empty application from the system. To remove an application with databases, use **cascade**. To remove an application that has locked objects in a constituent database, you can use **force**. Minimum permission required: designer.

▶▶── drop application APP-NAME ┬───────────────┬──▶◀
                              └─ cascade ─┘ └─ force ─┘

Terminal	Example
APP-NAME	sample

drop calculation

Delete a stored calculation from a database. Minimum permission required: designer.

▶▶── drop calculation CALC-NAME ─────────────▶◀

Terminal	Example
CALC-NAME	sample.basic.default

drop database

Delete a database from the system. Minimum permission required: Database designer. If the database has outstanding locks, clear them first, or use **force** to drop with locks.

▶▶── drop database DBS-NAME ┬──────────┬──▶◀
                           └─ force ─┘

Terminal	Example
DBS-NAME	sample.basic

drop filter

Delete a security filter from the database. Minimum permission required: Database designer.

►►── drop filter FILTER-NAME ─────────────► ►◄

Terminal	Example
FILTER-NAME	sample.basic.filt4

drop function

Delete a user-defined function from the application. Minimum permission required: Database designer.

For local functions, Essbase requires the application to be restarted. For global functions, Essbase requires all loaded applications to be restarted. Use **immediate** to let MaxL restart the application(s).

►►── drop function FUNC-NAME ────────────► ►◄
             └─ immediate ─┘

Terminal	Example
FUNC-NAME	sample.'@COVARIANCE'

drop group

Delete a user group from the system. Users belonging to the group are not deleted. Minimum permission required: create_user.

►►── drop group GROUP-NAME ─────────────► ►◄

Terminal	Example
GROUP-NAME	supervisors

drop location alias

Delete from the database a location alias identifying a host name, application, database, user name, and password. Minimum permission required: database designer.

Terminal	Example
LOCATION-ALIAS-NAME	sample.basic.Eastern

drop macro

Delete a user-defined macro from the application. Minimum permission required: Database designer.

For local macros, Essbase requires the application to be restarted. For global macros, Essbase requires all loaded applications to be restarted. Use **immediate** to let MaxL restart the application(s).

Terminal	Example
MACRO-NAME	Sample.'@COVARIANCE'

drop partition

Delete from the system a partition definition between two databases. Database designer privilege for each database is required.

Terminal	Example
DBS-NAME	sample.basic.Eastern

Terminal	Example
HOST-NAME	HQserver

drop user

Delete a user account from the system. Minimum permission required: create_user.

►►— drop user USER-NAME ─────────────────►◄

Terminal	Example
USER-NAME	Fiona

execute calculation

Execute a stored calculation, the stored default calculation (determined by **alter database**), or an anonymous (non-stored) calculation string.

►► — execute calculation ─┬─ CALC-NAME ─────────────────►◄
                  ├─ CALC-STRING ─┬─ on DBS-NAME ─┘
                  └─ default ─────┘

Terminal	Example
CALC-NAME	sample.basic.calcname
CALC-STRING	'CALC DIM(Year, Measures, Product);'
DBS-NAME	sample.basic

export

Export all data, level-0 data, or input-level data, which does not include calculated values. Export files are stored in the `$ARBORPATH/app` directory on the server.

Note: Export is part of the MaxL Command Shell grammar, not the MaxL language itself. You can use an export statement in MaxL scripts and the MaxL Command Shell, but you cannot embed it in Perl.
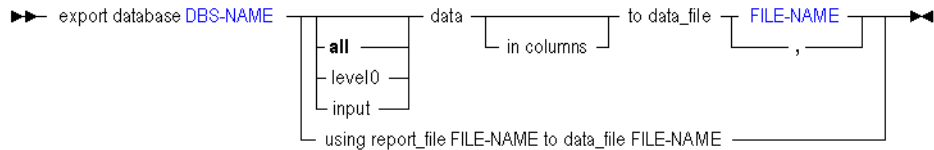
Terminal	Example
DBS-NAME	sample.basic
FILE-NAME	datafile.txt

grant

Grant a permission, a filter or a stored calculation to a user or a group.

Granting permissions:

At each level (system, application or database) existing roles are replaced. However, the built-in privileges create_user and create_application are not replaced.

Granting filters:

There may be only one filter per user per database. Therefore, granting a filter replaces any filters the user may already have on that database.

Granting calculations:

A user may have any number of calculations per database. Therefore, granting a calculation adds it to the user's list of calculations.
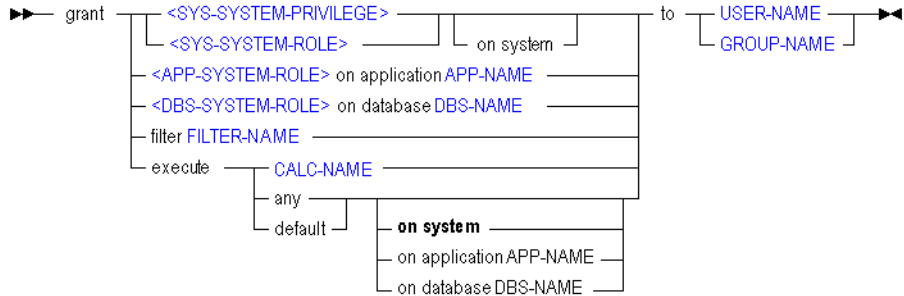
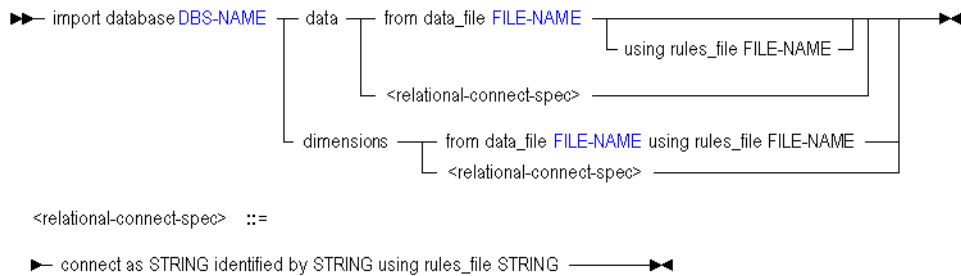| Terminal or <non-terminal>	Example
<SYS-SYSTEM-PRIVILEGE>	create_application or create_user
<SYS-SYSTEM-ROLE>	no_access or supervisor
<APP-SYSTEM-ROLE>	no_access or designer
APP-NAME	sample
<DBS-SYSTEM-ROLE>	no_access, read, write, or designer
DBS-NAME	sample.basic
FILTER-NAME	sample.basic.filt1
CALC-NAME	sample.basic.calcname
USER-NAME	Fiona
GROUP-NAME	newhires

import

Import data or dimensions from text data files, with or without a rules file. To import from a relational source, you must connect as the relational user name, and use a rules file.

Note: In some cases, import will generate an error log file, `essmsh.err`, in the current directory.

Note: Import is particular to the MaxL Command Shell. Because it is not part of the MaxL language itself, it cannot be used with Perl functions.

refresh replicated partition

Refresh the current replicated-partition database target from the remote (second DBS-NAME) source partition. Database designer privilege for each database is required.
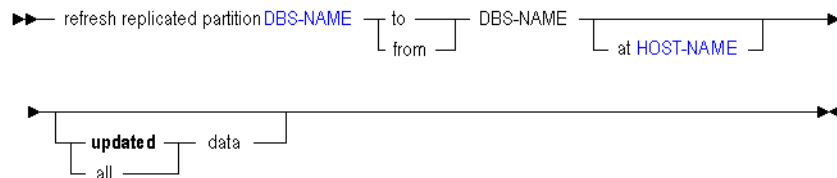
Terminal	Example
DBS-NAME	sample.basic.Eastern
HOST-NAME	HQserver

Error and Status Messages

This appendix lists MaxL error and status messages. These may be useful if you write Perl scripts or shell scripts which perform error handling.

For example, the following portion of a UNIX shell script ensures that there are no applications on the system, by testing whether **display application** returns zero applications. A MaxL statement is echoed into a stream-oriented MaxL session (invoked using essmsh -i), and then **awk** searches for the occurrence of a particular status message.

```
if [ $(echo "display application;" | essmsh -l admin password1 -i | \
       awk '/53 - Records returned:/ {print $5}' ) != "[0]." ]
then
    print "This test requires that there be no applications on the system"
    print "Quitting"
    exit 2
fi
```

Messages

```
1 - Internal error encountered.

2 - Invalid user name or password.

3 - Not logged in to Essbase.

4 - Password for this account has expired.

5 - User account locked.

6 - (line no) dynamic message

7 - Unable to resolve the host name.

8 - Bad error number.

9 - Essbase error encountered: [API func] [error no].
```

```
10 - Recursive login to remote site failed.

11 - Keyword, GLOBALLY, cannot be used as area alias.

12 - Duplicate area alias name.

13 - Bad area alias reference on the MAPPED clause.

14 - Unimplemented function.

15 - Partition definition skipped.

16 - Invalid destination for a data export.

17 - User does not exist: ['name'].

18 - User exists: ['name'].

19 - User altered: ['name'].

20 - User created: ['name'].

21 - User replaced: ['name'].

22 - User dropped: ['name'].

23 - Group does not exist: ['name'].

24 - Group exists: ['name'].

25 - Group created: ['name'].

26 - Group replaced: ['name'].

27 - Group dropped: ['name'].

28 - Application does not exist: ['name'].

29 - Application exists: ['name'].

30 - Application created: ['name'].

31 - Application replaced: ['name'].

32 - Application dropped: ['name'].

33 - Application is not empty: ['name'].

34 - Database does not exist: ['name'.'name'].

35 - Database exists: ['name'.'name'].

36 - Database created: ['name'.'name'].

37 - Database replaced: ['name'.'name'].

38 - Database dropped: ['name'.'name'].

39 - Partition refreshed.
```

40 - Partition does not exist.

41 - Partition exists.

42 - Partition created.

43 - Partition replaced.

44 - Partition dropped.

45 - Exported partition count: [n].

46 - Unable to open file: [/bad/directory/name.out].

47 - Unable to close file: [/bad/directory/name.out].

48 - Unable to allocate memory.

49 - User logged in: [name].

50 - MaxL compilation failed.

51 - MaxL execution failed.

52 - System altered.

53 - Records returned: [n].

54 - Application altered: ['name'].

55 - Location alias created: ['name'.'name'.'name'].

56 - Location alias dropped: ['name'.'name'.'name'].

57 - Location alias does not exist: ['name'.'name'.'name'].

58 - Group altered: ['name'].

59 - Grant succeeded for user or user group: ['name'].

60 - Invalid host.

61 - Application or database scope must be supplied with DESIGNER
privilege.

62 - User does not have specified privilege.

63 - Security filter does not exist: ['name'.'name'.'name'].

64 - Security filter exists: ['name'.'name'.'name'].

65 - Security filter created: ['name'.'name'.'name'].

66 - Security filter replaced: ['name'.'name'.'name'].

67 - Security filter dropped: ['name'.'name'.'name'].

68 - Unable to create security filter due to invalid row.

69 - Compression is disabled.

70 - Committed mode is disabled.

71 - Committed mode is enabled.

72 - Database altered: ['name'.'name'].

73 - retrieve_buffer_size must be between 2048 and 102400000 b.

74 - retrieve_sort_buffer_size must be between 10240 and 102400000 b.

75 - data_cache_size must be between 3145728 and 4294967295 b.

76 - data_file_cache_size must be between 8388608 and 4294967295 b.

77 - index_cache_size must be between 1048576 and 4294967295 b.

78 - index_page_size must be between 1024 and 8192 b.

79 - session_idle_limit must be between 300 and 4294967295 s.

80 - session_idle_poll must be between 30 and 4294967295 s.

81 - invalid_login_limit must be between 1 and 65535.

82 - Invalid privilege or role.

83 - Invalid prototype name: [name of object to copy].

84 - Invalid currency specification in CREATE DATABASE AS: [APIfunc].

85 - Substitution variable exists: ['name'].

86 - Substitution variable does not exist in the specified scope: [name].

87 - Semaphore operation failed.

88 - NULL

89 - Database import completed: ['name'.'name'].

90 - Database export completed: ['name'.'name'].

91 - Disk volume does not exist: [name].

92 - Disk volume exists: [name].

93 - Requested column number is too large.

94 - At least one database required for this operation.

95 - This Function or Macro cannot be deleted: [name].

96 - password_reset_days meaningless for this user.

97 - Calculation created: ['name'.'name'.'name'].

98 - Calculation replaced: ['name'.'name'.'name'].

99 - Calculation dropped: ['name'.'name'.'name'].

100 - Calculation executed.

101 - Calculation exists: ['name'.'name'.'name'].

102 - Calculation doesn't exist: ['name'.'name'.'name'].

103 - User not logged in: ['name'].

104 - Cannot log yourself out: ['name'].

105 - Sessions logged out: [n].

106 - User is not in group: ['groupname'].

107 - Cannot create a user as a group: ['name'].

108 - Cannot create a group as a user: ['name'].

109 - Invalid partition definition: [explanation].

110 - Unable to free memory.

111 - password_reset_days must be between 1 and 65535.

112 - inactive_user_days must be between 1 and 65535.

113 - lock_timeout (app) must be between 0 and 4294967295 s.

114 - max_lro_file_size must be between 1024 and 4398046510080 b.

115 - Disk volume file_size must be between 8388608 and 2147483648 b.

116 - Disk volume partition_size must be between 8388608 and 9007199254740991 b.

117 - implicit_commit must be between 0 and 4294967295 blocks/rows.

118 - lock_timeout (db) must be between 1 and 2147483 s.

119 - Function created: ['name'].

120 - Function replaced: ['name'].

121 - Function dropped: ['name'].

122 - Function exists (possibly internal): ['name'].

123 - Function does not exist: ['name'].

124 - Function definition is not valid: ['bad.javaclass.method'].

125 - Macro created: ['name'].

126 - Macro replaced: ['name'].

127 - Macro dropped: ['name'].

128 - Macro exists (possibly internal): ['name'].

129 - Macro does not exist: ['name'].

130 - Macro definition is not valid: ['macrodef'].

131 - DX compilation failed.

132 - DX execution failed.

133 - Maximum number of export files (8) exceeded.

134 - Invalid name: ['name'].