SOLID *Embedded Engine*

Administrator Guide

Version 3.0

Document number SSAG-3.0-0399
Date: March 26, 1999

Contents

Welcome ix

1 Introducing SOLID *Embedded Engine*

About SOLID *Embedded Engine* 1-1
SOLID *Embedded Engine* Components 1-3
SOLID *SynchroNet* 1-6

2 Basic Administration Tasks

Installing SOLID *Embedded Engine* 2-1
Starting SOLID *Embedded Engine* 2-1
Creating a New Database 2-2
Connecting to SOLID *Embedded Engine* 2-3
Viewing the SOLID *Embedded Engine* Message Log 2-4
Shutting Down SOLID *Embedded Engine* 2-4

3 Database Maintenance

Making Backups 3-1
Restoring Backups 3-3
Recovering from Abnormal Shutdown 3-4
Transaction Logging 3-4
Creating Checkpoints 3-4
Closing the Database 3-5
Changing Database Location 3-5
Running Several Servers on One Computer 3-6
Entering Timed Commands 3-6

4 Using SOLID Data Management Tools

Command Line Arguments 4-1
SOLID *SpeedLoader* 4-2
SOLID *Export* 4-11
SOLID *Data Dictionary* 4-12
SOLID *Remote Control* (Teletype) 4-13
SOLID *SQL Editor* (Teletype) 4-16
Tools Sample: Reloading a Database 4-18

5 Administration with SQL Statements

About SOLID SQL Syntax 5-1
Administering the Database 5-1
Managing User Privileges and Roles 5-2
Managing Tables 5-5
Managing Indexes 5-7
Primary Keys 5-8
Foreign Keys 5-8

6 Network Connections

Communication between Client and Server 6-1
Network Names for SOLID *Embedded Engine* 6-1
Network Name for Clients 6-3
Communication Protocols 6-4
Logical Data Source Names 6-11

7 Configuration

Configuration File and Default Settings 7-1
Most Important Parameters 7-1
Managing Parameters 7-5

8 Performance Tuning

Tuning SQL Statements and Applications 8-1
Tuning Memory Allocation 8-3

Tuning I/O 8-4
Sorting 8-5
Tuning Checkpoints 8-5

9 Diagnostics and Troubleshooting

Observing Performance 9-2
Tracing Communication between Client and Server 9-8
Problem Reporting 9-12
Problem Categories 9-12

A Error Codes

Error Categories A-1
SOLID SQL Errors A-3
SOLID Database Errors A-12
SOLID Utility Errors A-17
SOLID Table Errors A-20
SOLID *Embedded Engine* Errors A-30
SOLID Communication Errors A-32
SOLID Communication Warnings A-35
SOLID Procedure Errors A-36
SOLID Sorter Errors A-40

B Configuration Parameters

General Section B-2
IndexFile Section B-3
Logging Section B-4
Communication Section B-5
Data Sources B-6
Server Section B-6
SQL Section B-7
Sorter Section B-8

C Data Types

Supported Data Types in SOLID *Embedded Engine* C-1

D SOLID SQL Syntax

ADMIN COMMAND D-2
ALTER TABLE D-4
ALTER USER D-5
CALL D-5
COMMIT D-5
CREATE EVENT D-6
CREATE INDEX D-8
CREATE PROCEDURE D-8
CREATE ROLE D-16
CREATE SEQUENCE D-16
CREATE TABLE D-17
CREATE USER D-18
CREATE VIEW D-19
DELETE D-19
DELETE (positioned) D-19
DROP EVENT D-20
DROP INDEX D-20
DROP PROCEDURE D-20
DROP ROLE D-20
DROP SEQUENCE D-21
DROP TABLE D-21
DROP USER D-21
DROP VIEW D-21
EXPLAIN PLAN FOR D-22
GRANT D-22
INSERT D-23
INSERT (Using Query) D-24
REVOKE (Role from User) D-24
REVOKE (Privilege from Role or User) D-25
ROLLBACK D-25
SELECT D-26
SET D-27
SET SCHEMA D-28
UPDATE (Positioned) D-29

UPDATE (Searched) D-30
Table-reference D-30
Query-specification D-31
Search-condition D-31
Check-condition D-32
Expression D-33
String Function D-34
Numeric Function D-35
Date Time Function D-36
System Function D-37
Data-type D-37
Date and Time Literals D-38
Pseudo Columns D-38

E System Views and System Tables

F SOLID SQL API Reserved Words

G SOLID *Embedded Engine* Command Line Options

Glossary

Index

Welcome

SOLID Embedded Engine™ provides the local data storage needs required for today's complex distributed systems.

SOLID *Embedded Engine* provides support for real-time operating systems such as VxWorks and ChorusOS, and for preferred platforms such as Window 98/NT, Linus, Solaris, HP-UX and other UNIX platforms. It provides the features you would expect to find in any industrial-strength database server—multithread architecture, stored procedures, row level transaction management—but it delivers them with the special needs of today's applications.

About This Guide

This SOLID **Administrator Guide** is designed to make the administration of SOLID *Embedded Engine* smoother. This guide provides quick instructions on basic administration and maintenance, tools and utilities, and also provides reference information.

Organization

This manual contains the following chapters:

- *Chapter 1, Introduction to SOLID Embedded Engine* familiarizes you with the background and components of your SOLID data management system.
- *Chapter 2, Basic Administrative Tasks* covers the typical administration tasks such as starting, connecting to, and shutting down servers.
- *Chapter 3, Database Maintenance* explains how to make backups, create checkpoints, and use timed commands.
- *Chapter 4, Using SOLID Data Management Tools* describes the available utilities for handling database operations.
- *Chapter 5, Administration with SQL Statements* gives readers the information they need to manage users, tables and indexes.

- *Chapter 6, Network Connections* describes how to connect to SOLID *Embedded Engine* using different communication protocols.
- *Chapter 7, Configuration* describes how to set SOLID *Embedded Engine* parameters for customization to meet your own environment, performance, and operation needs.
- *Chapter 8, Performance Tuning* describes how to optimize SOLID *Embedded Engine* to improve performance.
- Chapter 9, *Diagnostics and Troubleshooting* describes tools to use for observing performance and tracing problems.

Appendixes

The *Appendixes* give you detailed information about error messages, configuration parameters, and SOLID SQL functionality.

Glossary

The *Glossary of Terms* explains some of the terminology used in SOLID documentation.

Audience

This manual assumes general DBMS knowledge, and a familiarity with SQL.

Conventions

Product Name

In version 3.0, SOLID *Server* or SOLID Web Engine is now known as SOLID *Embedded Engine.* This guide may still make reference to SOLID *Server.* Throughout this guide, "SOLID *Server"* and "SOLID *Embedded Engine"* are used synonymously.

Typographic

This manual uses the following typographic conventions.

Format	Used for
WIN.INI	Uppercase letters indicate filenames, SQL statements, macro names, and terms used at the operating-system command level.
`RETCODE SQLFetch(hdbc)`	This font is used for sample command lines and program code.

argument	Italicized words indicate information that the user or the application must provide, or word emphasis.
SQLTransact	Bold type indicates that syntax must be typed exactly as shown, including function names.
[ ]	Brackets indicate optional items; if in bold text, brackets must be included in the syntax.
|	A vertical bar separates two mutually exclusive choices in a syntax line.
{}	Braces delimit a set of mutually exclusive choices in a syntax line; if in bold text, braces must be included in the syntax.
...	An ellipsis indicates that arguments can be repeated several times.
. . .	A column of three dots indicates continuation of previous lines of code.

Other Solid Documentation

SOLID *Embedded Engine* documentation is distributed in an electronic format (PDF, HTML, or Windows Help files).

Solid Online Services on our Web server offer the latest product and technical information free of charge. The service is located at:

`http://www.solidtech.com/`

Electronic Documentation

- **Read Me** contains installation instructions and additional information about the specific product version. This `readme.txt` file is typically copied onto your system when you install the software.
- **Release Notes** contains additional information about the specific product version. This `relnotes.txt` file is typically copied onto your system when you install the software.
- **SOLID *SynchroNet* Guide** introduces you to synchronization concepts and architecture and describes how to set up, use and administer SOLID *SynchroNet*.

- **SOLID Programmer Guide** describes the interfaces (APIs and drivers) available for accessing SOLID *Embedded Engine* and how to use them with an embedded engine.

1

Introducing SOLID *Embedded Engine*

This chapter introduces you to SOLID *Embedded Engine*™, providing local data storage capabilities in today's complex distributed system environments. It describes its benefits, features, and main components.

About SOLID *Embedded Engine*

SOLID *Embedded Engine*, developed for this new era of distributed computing systems, provides what developers need, data storage features that meet the demands and requirements of their application environments.

Application developers can rely on SOLID *Embedded Engine's* wide range of data types, volumes, and processing features, which include, multithreaded parallel processing, symmetric multiprocessing (SMP), automatic roll-forward recovery, and stored procedures. Furthermore, SOLID *Embedded Engine's* portability and ease of deployment are ideal in today's internetworked environments. SOLID *Embedded Engine* supports operating systems in such infrastructure platforms as Window 98/NT, Linux, ChorusOS, VxWorks, Solaris, HP-UX and other UNIX platforms. It is fully Year 2000 Compliant.

SOLID *Embedded Engine* delivers performance within SQL-92, scalability, and high availability; yet it is lightweight, flexible, easy-to-use, and maintenance free with automatic operations.

SOLID *Embedded Engine* Features:

SOLID *Embedded Engine* is a secure, reliable, and accommodating solution to your data storage needs. This section includes some of its unique benefits and features.

SOLID Bonsai Tree™

SOLID *Embedded Engine* features a small, but efficient index, known as The Bonsai Tree. This index tree resides in the main memory and maintains multiversion information. The Bonsai Tree performs concurrency control, detecting if any operations conflict with each

other. This minimizes the effort needed for validating transactions. Active new data is separated from older, more stable data, which is transferred to a storage server as a highly-optimized batch insert, thus minimizing the hard disk load. The Bonsai Tree offers:

- Both optimistic and pessimistic concurrency control
- Fully serializable transactions free from phantom updates
- Multi-versioning that allows a consistent view of the database without extra locking
- Row-level locking is available if needed for pessimistic or mixed concurrency control methods. It can be turned on table by table, and a single transactions can use both pessimistic and optimistic concurrency control methods simultaneously.
- Declarative referential integrity ensuring the validity of references between tables.

Wide range of data type support

SOLID *Embedded Engine* supports binary compatible databases across all platforms. This support includes:

- Binary Large Objects (BLOBs), such as a picture, video clip, sound excerpt, or a formatted text object.
- Data stored in a variable-length format.
- Practically unlimited amount of tables, columns, keys, etc.
- Unicode support for double-byte character sets.

Stored procedures, event alerts, and sequencer objects

SOLID *Embedded Engine* provides these active database objects for reduced overhead:

- Stored procedure are used to execute part of the application logic in the server and for optimizing queries. A stored procedure can contain several SQL statements or a whole transaction for execution with a single call statement.
- Event alerts are used with stored procedures to signal an event in the database, thereby freeing the stored procedure from conducting its own database polling.
- Sequencer objects generate number sequences for objects stored in databases. Sequences have an advantage over separate tables. They are specifically fine-tuned for fast execution and result in less overhead than normal update statements.

Easy Administration

With SOLID *Embedded Engine* all administrative operations, including backups are performed automatically or at the administrator's request. Built-in timers are available for vari-

ous administrative tasks. For example, administrator's can specify automated daily or weekly backups.

SOLID *Embedded Engine* also features online concurrent backup, and automatic and roll-forward recovery. Automatic recovery returns the database to the state it was in at the moment it encountered the error. To guarantee database integrity, all committed transactions are read from the transaction log.

SOLID *Embedded Engine* provides administrative tools for interactive SQL, remote administration, as well as transformation tools for loading character data from character format data files, exporting character data to character format files, and writing data dictionary definitions of a database. For brief description of these tools, read *System Tools and Utilities* in this chapter.

SOLID *Embedded Engine* Components

SOLID *Embedded Engine*, the local data storage system for complex distributed network environments, contains the components described in the following sections.

Programming interfaces (*SQL API*, ODBC, and JDBC)

SOLID *SQL API* is a Call Level Interface (CLI) that follows the ANSI SQL CLI and ODBC CLI specifications. SOLID provides ODBC and JDBC APIs for programming access to SOLID data. For more details on programming interfaces, read the **SOLID Programmer Guide**.

Network Services

SOLID Embedded Engine runs on all major network types and supports all of the main communication protocols. Developers can create distributed applications for use in heterogeneous computing environments. For more details on network communication, read Chapter 6, "Network Connections" in this guide.

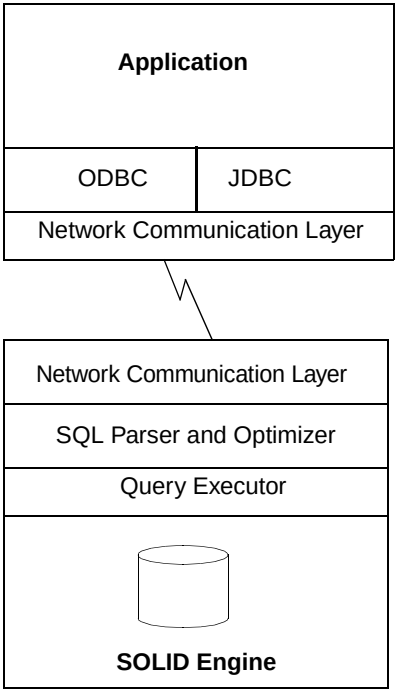
SQL Parser and Optimizer

The SQL syntax used is based on the ANSI X3H3-1989 Level 2 standard and ANSI X3H3-1992 (SQL2) extensions. SOLID *Embedded Engine* contains an advanced cost-based optimizer, which ensures that even complex queries can be run efficiently. The optimizer automatically maintains information about table sizes, the number of rows in tables, the available indices, and the statistical distribution of the index values.

Engine

The SOLID engine is the core of the SOLID *Embedded Engine* product. It processes the data requests submitted via SOLID SQL. The engine stores data and retrieves it from the database.

Figure 1–1 SOLID Embedded Engine Components



System Tools and Utilities

SOLID Embedded Engine also includes the following tools for data management and administration:

SOLID *Remote Control*

SOLID *Remote Control* is a program for administration of SOLID servers. With SOLID *Remote Control*, you can:

- administer all database servers in a network from a single workstation
- generate backups either on-line or as a timed command
- obtain server status information

SOLID *SQL Editor*

SOLID *SQL Editor* is a tool for executing SQL queries and commands. It has an easy-to-use graphical user interface. With SOLID SQL Editor, you can:

- use either the interactive or batch mode operation
- have multiple active connections to various servers
- save or print query results

Tools for handling ASCII data

SOLID Embedded Engine provides the following tools for handling ASCII data:

- SOLID *Speedloader* loads data from external ASCII files into a SOLID database. It is capable of inserting character data from character format. SOLID *SpeedLoader* bypasses the SQL parser and uses direct writes to the database file with loading, which allows for fast loading speed.
- SOLID *Export* writes from a SOLID database to character format files. It is capable of writing control files used by SOLID *SpeedLoader* to perform data unload/load operations.
- SOLID *Data Dictionary* (SOLDD) writes the data dictionary of a database. This tool produces an SQL script that contains data definition statements describing the structure of the database.

Read Chapter 4, “Using SOLID Data Management Tools” for details.

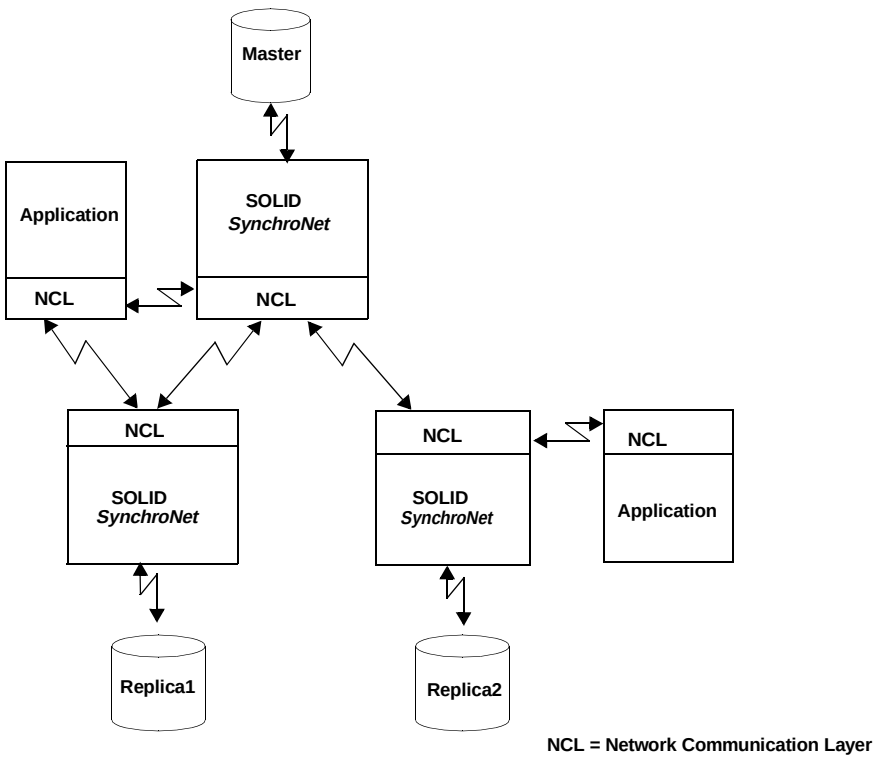
SOLID *SynchroNet*

SOLID *SynchroNet* builds on the local data storage capabilities of SOLID *Embedded Engine*. It provides system-wide data sharing, which is particularly suited for applications in today's internetworked systems. With SOLID *SynchroNet's* asynchronous, bi-directional data synchronization, you can store data where it makes sense and deliver data where and when you need it there.

SOLID *SynchroNet's* new approach to replication addresses the data reliablity shortcomings of traditional replication models. Its architecture builds data synchronization functionality inside a business application. Using SOLID *SynchroNet* SQL extensions and Intelligent Transactions^{TM}, application developers, with minimal effort, can provide the logic to ensure data reliability within the context of their applications.

For details on SOLID *SynchroNet*, read the *SOLID SynchroNet* Guide.

Figure 1–2 System-wide sharing with SOLID SynchroNet



2

Basic Administration Tasks

This chapter covers the basic SOLID *Embedded Engine* administrative tasks. It tells you how to:

- Install SOLID *Embedded Engine*
- Start SOLID *Embedded Engine*
- Create a new database
- Connect to the server using SOLID *Remote Control* or SOLID *SQL Editor*
- Shut down SOLID *Embedded Engine* using SOLID *Remote Control* or from the server computer console

Installing SOLID *Embedded Engine*

If you have not yet installed SOLID *Embedded Engine*, refer to the **ReadMe** notice delivered with the software for a detailed description of the installation.

Starting SOLID *Embedded Engine*

When SOLID *Embedded Engine* is started, it checks if a database already exists in the SOLID directory, that is, the directory where you installed SOLID executables. If a database file is found, SOLID *Embedded Engine* will automatically open that database. If not, which is the case when you start the server for the first time, a new database will be created.

Operating System	To Start the Server...
UNIX	Enter the command `solid` at the command prompt. When you start the server for the first time, enter the command `solid -f` at the command prompt to force the server to run in the foreground.
Novell Netware	Enter the command `load solid.nlm` at the command prompt.

Operating System	To Start the Server...
Open VMS	Enter the command `run solid` at the command prompt.
Windows	Click the icon labeled `SOLID Embedded Engine` in the SOLID Embedded Engine program group.

Creating a New Database

If a database does not exist, SOLID *Embedded Engine* will automatically start creating a new database. In the Windows environment, creating the database begins with a dialog prompting for the database administrator's username and password. In other environments, if you do not have an existing database, the following message appears:

```
Database does not exist. Do you want to create a new database (y/n)?
```

Answer y(es), and SOLID *Embedded Engine* will prompt for the database administrator's username and password. When they have been accepted, a new database will be created.

The username and password are case insensitive. The username must have at least two characters; the password at least three. You can use lower case letters from a to z, upper case letters from A to Z and the underscore character '_', and numbers from 0 to 9.

NOTE: You must remember your username and password to be able to connect to SOLID *Embedded Engine*. There are no default usernames; the username you enter when creating the database is the only username available for connecting to the new database.

After accepting the database administrator's username and password, SOLID *Embedded Engine* will create a new database. By default the database will be created as one file (solid.db) into the SOLID directory, where the current working directory is located. The time that the database creation process takes depends on the hardware platform you are using.

After the database has been created, SOLID *Embedded Engine* starts listening to the network for client connection requests. In the Windows environment, you will see a SOLID *Embedded Engine* icon, but in most environments SOLID *Embedded Engine* will run invisibly in the background as a daemon process.

Windows only If in the Windows environment you double-click the icon of a running server, nothing will happen. SOLID *Embedded Engine* is a background process that only reacts to messages from clients through the communication interface.

Connecting to SOLID *Embedded Engine*

After starting SOLID *Embedded Engine*, you can test the configuration by connecting to the server from your workstation by using either SOLID *Remote Control* or SOLID *SQL Editor*.

NOTE: You need to have the privileges of SYS_ADMIN_ROLE or SYS_CONSOLE_ROLE to be able to connect to a server using SOLID *Remote Control*.

Connecting with SOLID *Remote Control*

1. View the `solmsg.out` file for valid network names that you can use to connect to SOLID *Embedded Engine*.

   The following messages indicate what names you can use.

   ```
   Listening of 'ShMem Solid' started.
   Listening of 'TCP/IP 1313' started.
   ```

2. Start SOLID *Remote Control* and give the network name of server as a command line parameter:

   ```
   solcon "tcp hobbes 1313"
   ```

3. Enter the database administrator's user name and password when prompted.

4. After a while you will see a message indicating that a connection to the server has been established.

Connecting with SOLID *SQL Editor*

1. View the `solmsg.out` file for valid network names that can be used to connect to SOLID *Embedded Engine*.

   The following messages indicate what names you can use:

   ```
   Listening of 'ShMem solid' started.
   Listening of 'TCP/IP 1313' started.
   ```

2. Start SOLID *SQL Editor* and give the network name of server as a command line parameter:

   ```
   solsql "tcp hobbes 1313"
   ```

3. Enter the database administrator's user name and password when prompted.

4. After a while you will see a message indicating that a connection to the server has been established.

Viewing the SOLID *Embedded Engine* Message Log

SOLID *Embedded Engine* writes all error and info messages to a text file. This message log file is named `solmsg.out` and it is located in the SOLID directory. You can view this file using any text editor or file viewer. The error messages and their explanations are listed in *Appendix A, "Error Codes"* of this document.

Shutting Down SOLID *Embedded Engine*

You can shut down SOLID *Embedded Engine* in these ways:

- Programmatically from an application using the following SQL commands: ADMIN COMMAND 'throwout all' and ADMIN COMMAND 'shutdown'
- Using the SOLID *Remote Control* program
- Clicking the server icon and selecting `Close` from the menu appearing in the Windows environment

All the shutdown mechanisms will start the same routine, which writes all buffered data to the database file, frees cache memory and finally terminates the server program. Shutting down a server may take a while since the server must write all buffered data from main memory to the disk.

3
Database Maintenance

This chapter gives you information on data security and database maintenance. It is divided into the following topics:

- making backups
- restoring backups
- recovering from abnormal shutdown
- logging
- creating checkpoints
- closing and opening the database
- changing database location
- running several servers on one computer
- entering timed commands

Making Backups

Backups are made to secure the information stored in your database files. If you have lost your database files because of a system failure, you can continue working with the backup database.

You can initiate a backup in the following ways:

- Automate the backup using a timed command that initiates the backup according to a pre-defined schedule. Read *"Entering Timed Commands"* in this chapter for details.
- Use the SQL command ADMIN COMMAND 'backup' from the application.
- Start the backup from SOLID *Remote Control*.

NOTE: Be sure to have enough disk space in the backup directory. You will need space for your database and log files.

SOLID *Embedded Engine* uses a multiversioning technique allowing backups to be made on-line. You need not close the database file or shut down the server. However, it is advisable to automate your backups to be run at non-busy hours. After completing the backup, copy your backup files on tape using your backup software for protection against disk crashes.

Please also note the following:

- You can query programmatically the status of the most recently started backup by using SQL command ADMIN COMMAND 'status backup'. To query the list of all completed backups and their success status, use SQL command ADMIN COMMAND 'backuplist'.
- The backup directory you enter must be a valid path name in the server operating system. For example, if the server runs on a UNIX operating system, path separators must be slashes, not backslashes.
- The time needed for making a backup is the time that passed between the messages `Backup started` and `Backup completed successfully`, which arrive to your SOLID *Remote Control* `MESSAGES` page.

Before starting the backup process, a checkpoint is created automatically. This guarantees that the state of a backup database is from the moment the backup process was started. The following files will be copied to the backup directory:

- database file(s)
- configuration file (`solid.ini`)
- log file(s) modified or created after the previous backup (parameter `BackupCopyLog` is set `yes` by default)

The unnecessary log files are deleted from original directory after successful backup (parameter `BackupDeleteLog` is set `yes` by default).

To Correct a Failed Backup

When SOLID *Embedded Engine* is performing a backup, the ADMIN COMMAND 'status backup' command returns the value 'ACTIVE'. Once the backup is completed, the command returns either 'OK' or 'FAILED'. You can also query this information from SOLID *Embedded Engine* using SOLID *Remote Control*.

If the backup failed, you can find the error message that describes the reason for the failure from the solmsg.out file. Correct the cause of the error and try again. The most common causes for failed backups are:

- the backup media is out of disk space
- the backup directory does not exist
- a database directory is defined as the backup directory

Restoring Backups

There are two alternative ways to restore a backup. You can either:

- Return to the state when backup was created, or
- Revive a backup database to the current state by using log files to add data inserted or updated after the backup was made.

To Return to the State when the Backup was Made

1. Shut down SOLID *Embedded Engine*, if it is running.
2. Delete all log files from the log file directory. The default log file names are `sol00001.log`, `sol00002.log`, etc.
3. Copy the database file(s) from the backup directory to the database file directory.
4. Start SOLID *Embedded Engine*.

This method will not perform any recovery because no log files exist.

To Revive a Backup Database to the Current State

1. Shut down SOLID *Embedded Engine*, if it is running.
2. Copy the database file(s) from the backup directory to the database file directory.
3. Copy the log files from the backup directory to the log file directory. If there are log-files with the same file names, **do not** replace those log files in the log file directory with log files from the backup directory.
4. Start SOLID *Embedded Engine*.

SOLID *Embedded Engine* will automatically use the log files to perform a roll-forward recovery.

Recovering from Abnormal Shutdown

If the server was closed abnormally, that is, if it was not shut down using the procedures described earlier, SOLID *Embedded Engine* will automatically use the log files to perform a roll-forward recovery during the next start up. No administrative procedures are needed to start the recovery.

The message `Starting roll-forward recovery` appears. After the recovery has been completed, a message will indicate how many transactions were recovered. If no transactions were made since the last checkpoint, this is indicated by the following message

```
Recovery successfully completed
```

Transaction Logging

Transaction logging guarantees that no committed operations are lost in case of a system failure. When an operation is executed in the server, the same operation is also saved to a log file. The log file is used for recovery in case the server is shut down abnormally.

A backup operation will copy the log and database files to the backup directory and delete the log files from the database directory. You may change the default behavior by changing the parameters `BackupCopyLog` and `BackupDeleteLog` in the `General` section of parameters in `solid.ini`.

TIP: For both security and performance reasons, it is a good idea to keep log files and database files on different physical disk devices. If one disk drive is damaged, you will lose either your database files or log files but not both.

Creating Checkpoints

Checkpoints are used to store a consistent state of the database onto the database file. Checkpoints are needed for speeding up the roll-forward recovery after a system failure. In the roll-forward recovery, the database will start recovering transactions from the last checkpoint. The longer it has been since the last checkpoint was created, the more operations are recovered from the log file(s).

To speed up recoveries, checkpoints should be created frequently; however, the server performance is reduced during the creation of a checkpoint. Furthermore, the speed of checkpoint creation depends on the amount of database cache used; the more database cache is used, the longer the checkpoint creation will take. Consider these issues when deciding the frequency of checkpoints. See *Appendix B, "Configuration Parameters"* for a description of the use of `CacheSize` parameter.

SOLID *Embedded Engine* has an automatic checkpoint creation daemon, which creates a checkpoint after a certain number of writes to the log files. The default checkpoint interval is

every 5000 log writes. You may change the value of the parameter `CheckpointInterval` in the `General` section of parameters. To learn how to change a parameter value, see *Chapter 7, "Configuration"* in this guide.

Before and after a database operation, you may want to create a checkpoint manually. You can do this programmatically from your application with SQL command ADMIN COMMAND 'makecp'. You can also force a checkpoint using a timed command. Read the section "*Entering Timed Commands"* in this chapter for details.

NOTE. There can be only one checkpoint in the database at a time. When a new checkpoint is created, the older checkpoint is automatically erased.

Closing the Database

To close the database, use SQL command ADMIN COMMAND 'close'.

In some cases you may want to prevent users from connecting to the engine. For example, when you are shutting down an engine, you may want to prevent new users from connecting to the engine. After closing the database, only connections from SOLID *Remote Control* will be accepted. Closing the database does not affect existing user connections.

When the database is closed no new connections are accepted (clients will get SOLID Error Message 14506).

Changing Database Location

Changing a database location in SOLID *Embedded Engine* is as easy as copying a file from one directory to another.

NOTE. To copy a database file, you need to shut down the engine to release the operating system file locks on the database file and log files.

To Change Database Location

1. Verify that SOLID *Embedded Engine* is not running.
2. Copy the database and log files to the target directory.
3. Copy the `solid.ini` file to the target directory. Check that the database file directory, log file directory and backup directory are correctly defined in the configuration file `solid.ini`.
4. Start SOLID *Embedded Engine* using the target directory as the current working directory using the command line option `-c` *`directory-name`*.

Running Several Servers on One Computer

In some cases, you may want to run two or more databases on one computer. For example, you may need a configuration with a production database and a test database running on the same computer.

SOLID *Embedded Engine* is able to use one database per database server, but you can start several engines each using its own database file. To make these engines use different databases, either start the engine processes from the directories your databases are located in or give the locations of configuration files by using the command line option `-c` *`directory-name`* to change the working directory. Remember to use different network names for each engine.

Entering Timed Commands

SOLID *Embedded Engine* has a built-in timer, which allows you to automate your administrative tasks. You can use timed commands to execute system commands, to create backups, checkpoints and database status reports, to open and close databases, to disconnect users or to shut down engines.

To Enter a Timed Command

Timed commands are entered manually by editing the `At` parameter of the `[Srv]` section in the `solid.ini` file. The syntax is:

```
At-string := timed-command[, timed-command]
timed-command := [day] HH:MM command argument
day := sun | mon | tue | wed | thu | fri | sat
```

If the day is not given, the command is executed daily.

Example:

```
[Srv]
At=20:30 makecp,21:00 backup,sun 23:00 shutdown
```

NOTE: The format used is HH:MM (24-hour format).

Arguments and the Defaults for the Different Timed Commands

Command	Argument	Default
`backup`	backup directory	the default backup directory that is set in the configuration file
`throwout`	user name, all	no default, argument compulsory
`makecp`	no arguments	no default
`shutdown`	no arguments	no default
`report`	report file name	no default, argument compulsory
`system`	system command	no default
`open`	no arguments	no default
`close`	no arguments	no default

4

Using SOLID Data Management Tools

This chapter describes SOLID data management tools, a set of utilities for performing various database tasks. Not all SOLID Tools are necessarily part of the standard product delivery, and their availability on some platforms may be limited. For information about SOLID data management tools, contact your SOLID sales representative or SOLID Online Services at the Solid Web site:

`http://www.solidtech.com/`

Command Line Arguments

This paragraph lists and describes the available command line arguments that can be used with all teletype SOLID Database Tools. The tool-specific options are listed with the usage of each tool.

NOTE. When there is a contradiction in the command line, the tool gives you a list of the possible options as a result. Please check the command line you entered.

Command Line Arguments

Argument	Description
server name	This network name of the SOLID server that you are connected to. Logical Data Source Names can also be used with tools; refer to the **chapter *Network Connections*** for further information. The given network name must be enclosed in quotes.
user name	This is required to identify the user and to determine which rights he has. Without appropriate rights execution is denied.
password	This password given to the user for accessing the database.

table name	The name of the table accessed. * can be used with SOLID *Export* to export all tables with one command line.
control file	The name of the control file that defines the import file used with SOLID *SpeedLoader.* A file of this type is produced by executing SOLID *Export*.

SOLID *SpeedLoader*

SOLID *SpeedLoader* is a tool for loading data from external ASCII files into a SOLID database. SOLID *SpeedLoader* can load data in a variety of formats and produce detailed information of the loading process into a log file. The format of the import file, that is, the file containing the external ASCII data, is specified in a control file.

The data is loaded into the database through the SOLID *Embedded Engine* program. This enables online operation of the database during the loading. The data to be loaded does not have to reside in the server computer.

Control File

The control file provides information on the structure of the import file. It gives the following information:

- name of the import file
- format of the import file
- table and columns to be loaded

NOTE. Each import file requires a separate control file. SOLID *SpeedLoader* loads data into one table at a time.

The control file format is somewhat similar to control file structures found in other database management systems, such as Oracle and DB/2. Please note the following:

- The table must exist in the database in order to perform data loading.
- Schema support is not currently available in SOLID *Speedloader.*

Import File

The import file must be of ASCII type. The import file may contain the data either in a fixed or a delimited format:

- In fixed-length format data records have a fixed length, and the data fields inside the records have a fixed position and length.

- In delimited format data records can be of variable length. Each data field and data record is separated from the next with a delimiting character such as a comma (this is what SOLID *Export* produces). Fields containing no data are automatically set to NULL.

Data fields within a record may be in any order specified by the control file. Please note the following:

- Data in the import file must be of a suitable type. For example, numbers that are presented in a Float format cannot be loaded into a field of Integer or Smallint type.
- Data of Varbinary and Long Varbinary type are hexadecimal encoded in the import file.

Message Log File

During loading, SOLID *SpeedLoader* produces a log file containing the following information:

- the date and time of the loading
- loading statistics such as the number of rows successfully loaded, the number of failed rows, and the load time if it has been specified with the option
- Any possible error messages

If the log file cannot be created, the loading process is terminated. By default the name of the log file is generated from the name of the import file by substituting the file extension of the import file with the file extension `.log`. For example, `my_table.ctr` creates the log file `my_table.log`. To specify another kind of file name, use the option -l.

Configuration File

A configuration file is not required for SOLID *SpeedLoader.* The configuration values for the server parameters are included in the SOLID *Embedded Engine* configuration file `solid.ini`.

Client copies of this file can be made to provide connection information required for SOLID *Speedloader.* If no server name is specified in the command line, SOLID *SpeedLoader* will choose the server name it will connect to from the server configuration file. For example to connect to a server using the NetBIOS protocol and with the server name SOLID, the following lines should be included in the configuration file:

```
[Com]
Connect=netbios SOLID
```

Invoking SOLID *SpeedLoader*

SOLID *SpeedLoader* is invoked with the command `solload` followed by various arguments. If you invoke SOLID *SpeedLoader* with no arguments, you will see a summary of the arguments with a brief description, i.e. their usage. The command line syntax is:

```
solload [options] [server-name] <user-name> <password> <control-file>
```

The possible options are in the following table:

Option	Description
-b<records>	Number of records to commit in one batch
-c<dir>	Change working directory
-l<filename>	Write log entries to this file
-L<filename>	Append log entries to this file
-n<records>	Insert array size (network version)
-t	Print load time
-x emptytable	Load data only if there are no rows in the table
-x errors:<count>	Maximum error count
-x nointegrity	No integrity checks during load (standalone version)
-x skip:<records>	Number of records to skip
-?	Help = Usage

Control File Syntax

The control file syntax has the following characteristics:

- keywords must be given in capital letters
- comments can be included using the standard SQL double-dash (--) comment notation
- statements can continue from line to line with new lines beginning with any word

SOLID *SpeedLoader* reserved words must be enclosed in quotes if they are used as data dictionary objects, that is, table or column names. The following list contains all reserved words for the SOLID *SpeedLoader* control file:

AND	ANSI
APPEND	BINARY

BLANKS	BY
CHAR	CHARACTERSET
DATA	DATE
DECIMAL	DOUBLE
ENCLOSED	ERRORS
FIELDS	FLOAT
IBMPC	INFILE
INSERT	INTEGER
INTO	LOAD
LONG	MSWINDOWS
NOCNV	NOCONVERT
NULLIF	NULLSTR
NUMERIC	OPTIONALLY
OPTIONS	PCOEM
POSITION	PRECISION
PRESERVE	REAL
REPLACE	SCAND7BIT
SKIP	SMALLINT
TABLE	TERMINATED
TIME	TIMESTAMP
TINYINT	VARBIN
VARCHAR	WHITESPACE

The control file begins with the statement LOAD DATA followed by several statements that describe the data to be loaded. Only comments or the OPTIONS statement may optionally precede the LOAD DATA statement.

The following table describes the full syntax of the control file.

Syntax Element	
control-file	::= [option-part] load-data-part into-table-part

option-part	::= OPTIONS (options)
options	::= option [, option]
option	::= [SKIP = 'int_literal'] | [ERRORS = 'int_literal']
load-data-part	::= LOAD [DATA] [characterset-specification] [DATE date_mask] [TIME time_mask] [TIMESTAMP timestamp_mask] [INFILE filename] [PRESERVE BLANKS]
characterset-specification	::= CHARACTERSET { NOCONVERT | NOCNV | ANSI | MSWINDOWS | PCOEM | IBMPC | SCAND7BIT }
into-table-part	::= INTO TABLE tablename [APPEND | INSERT | REPLACE] [FIELDS TERMINATED BY { WHITESPACE | hex_literal |'char'}] [FIELDS [OPTIONALLY] ENCLOSED BY {"char'"| hex_literal} [AND "char" | hex_literal]] (column_list)
hex_literal	::= X'hex_byte_string'
column_list	::= column [, column]
column	::= column_name datatype_spec [POSITION ('int_literal' {: | -} 'int_literal')] [DATE date_mask] [TIME time_mask] [TIMESTAMP timestamp_mask] [NULLIF BLANKS | NULLIF NULLSTR| NULLIF 'string' | NULLIF (('int_literal' {: | -} 'int_literal') = 'string')]
datatype_spec	::= {BINARY | CHAR [length] | DATE | DECIMAL [(precision [, scale])] | DOUBLE PRECISION | FLOAT [(precision)] | INTEGER | LONG VARBINARY | LONG VARCHAR | NUMERIC [(precision [, scale])] | REAL | SMALLINT | TIME | TIMESTAMP [(timestamp precision)] | TINYINT | VARBINARY | VARCHAR [(length)] }

The following paragraphs explain syntax elements and their use is in detail.

CHARACTERSET

The CHARACTERSET keyword is used to define the character set used in the input file. If the CHARACTERSET keyword is not used or if it is used with the parameter NOCON-

VERT or NOCNV, no conversions are made. Use the parameter ANSI for the ANSI character set, MSWINDOWS for the MS Windows character set, PCOEM for the ordinary PC character set, IBMPC for the IBM PC character set, and SCAND7BIT for the 7-bit character set containing Scandinavian characters.

DATE, TIME, and TIMESTAMP

These keywords can be used in two places with different functionality:

- When one of these keywords is used as a part of the load-data-part element, it defines the format used in the import file for inserting data into any column of that type.
- When a keyword appears as a part of a column definition it specifies the format used when inserting data into that column.

NOTE 1. Masks used as part of the load-data-part element must be in the following order: DATE, TIME, and TIMESTAMP. Each is optional.

NOTE 2. Data must be of the same type in the import-file, the mask, and the column in the table into which the data is loaded.

The following table shows the available data masks:

Data Type	Available Data Masks
DATE	YYYY/YY-MM/M-DD/D
TIME	HH/H:NN/N:SS/S
TIMESTAMP	YYYY/YY-MM/M-DD/D HH/H:NN/N:SS/S

In the above table, year masks are YYYY and YY, month masks MM and M, day masks DD and D, hour masks HH and H, minute masks NN and N, and second masks SS and S. Masks within a date mask may be in any order, e.g., a date mask could be ‘MM-DD-YYYY’. If the date data of the import file is formatted as 1995-01-31 13:45:00, use the mask YYYY-MM-DD HH:NN:SS.

PRESERVE BLANKS

The PRESERVE BLANKS keyword is used to preserve all blanks in text fields.

into-table-part

The into-table-part element is used to define the name of the table and columns that the data is inserted into.

FIELDS TERMINATED BY

The FIELDS TERMINATED BY keyword is used to define the character used to distinguish where fields end in the input file.

The ENCLOSED BY keyword is used to define the character that precedes and follows data in the input file.

POSITION

The POSITION keyword is used to define a field's position in the logical record. Both start and end positions must be defined.

NULLIF

The NULLIF keyword is used to give a column a NULL value if the appropriate field has a specified value. An additional keyword specifies the value the field must have. The keyword BLANKS sets a NULL value if the field is empty; the keyword NULL sets a NULL value if the field is a string 'NULL'; the definition 'string' sets a NULL value if the field matches the string 'string'; the definition '((start : end) = 'string')' sets a NULL value if a specified part of the field matches the string 'string'.

Loading Fixed-format Records

Examples of the control file when loading data from a fixed-format import file:

```
-- EXAMPLE 1
LOAD DATA
INFILE 'EXAMP1.DAT'
INTO TABLE SUPPLIERS (
NAME  POSITION(01:19) CHAR,
ADDRESS     POSITION(20:40) VARCHAR,
ID    POSITION(41:48) INTEGER )
-- EXAMPLE 2
OPTIONS (SKIP = 10, ERRORS = 5)
-- Skip the first ten records. Stop if
-- errorcount reaches five.
LOAD DATA
INFILE 'sample.dat'
-- import file is named sample.dat
INTO TABLE TEST1 (
ID         INTEGER POSITION(1-5),
ANOTHER_ID INTEGER POSITION(8-15),
DATE1      POSITION(20:29) DATE 'YYYY-MM-DD',
```

```
DATE2      POSITION(40:49) DATE 'YYYY-MM-DD' NULLIF NULL)
```

Loading Variable-length Records

Examples of the control file when loading data from a variable-length import file:

```
-- EXAMPLE 1
LOAD DATA
INFILE 'EXAMP2.DAT'
INTO TABLE SUPPLIERS
FIELDS TERMINATED BY ','
(NAME VARCHAR, ADDRESS VARCHAR, ID INTEGER)
-- EXAMPLE 2
OPTIONS (SKIP=10, ERRORS=5)
-- Skip the first ten records. Stop if
-- errorcount reaches five.
LOAD
DATE 'YYYY-MM-DD HH:NN:SS'
-- The date format in the import file
INFILE 'sample.dat'
-- The import file
INTO TABLE TEST1
-- data is inserted into table named TEST1
FIELDS TERMINATED BY X'2C'
-- Field terminator is HEX ',' == 2C
-- This line could also be:
-- FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '[' AND ')'
-- Fields may also be enclosed
-- with '[' and ')'
(
ID INTEGER,
ANOTHER_ID DECIMAL(2),
DATE1 DATE(20) DATE 'YYYY-MM-DD HH:NN:SS',
DATE2 NULLIF NULL
)
-- ID is inserted as integer
-- ANOTHER_ID is a decimal number with 2
-- digits.
-- DATE1 is inserted using the datestring
-- given above
-- The default datestring is used for DATE2.
```

```
-- If the column for DATE2 is 'NULL' a NULL is
-- inserted.
```

Running a Sample Load Using Solload

To Run a Sample Load Using Solload

1. Start SOLID *Embedded Engine*.
2. Create the table using the `sample.sql` script and your SOLID *SQL Editor*.
3. Start loading using the following command line:

   ```
   solload "shmem solid" dba dba delim.ctr
   ```

The user name and password are assumed to be 'dba'. To use the fixed length control file, use the following command line:

```
solload "shmem solid" dba dba fixed.ctr
```

The output of a successful loading using `delim.ctr` will be:

```
SOLID Speed Loader v.3.00.00xx
(C) Copyright Solid Information Technology Ltd 1992-1999
Load completed successfully, 19 rows loaded.
```

The output of a successful loading using `fixed.ctr` will be:

```
SOLID Speed Loader v.3.00.00xx
(C) Copyright Solid Information Technology Ltd 1992-1999
Load completed successfully, 19 rows loaded.
```

Hints to Speed up Loading

The following hints can be used to ensure that loading is done with maximum performance:

- It is faster not to load data over the network, that is, connect locally if possible.
- Increasing the number of records committed in one batch speeds up the load. By default, commit is done after each record.
- Disable logging.

To disable logging the LogEnabled parameter needs to be used. The following lines in the `solid.ini` file will disable logging:

```
[Logging]
```

```
LogEnabled=no
```

After the loading has been completed, remember to enable logging again. The following line in the `solid.ini` file will enable logging:

```
[Logging]
LogEnabled=yes
```

NOTE. Running the server with logging disabled is strongly discouraged. If logs are not written, no recovery can be made if an error occurs due to power failure, disk error etc.

SOLID *Export*

SOLID *Export* is a product for unloading data from a SOLID database to ASCII files. SOLID *Export* produces both the import file, that is, the file containing the exported ASCII data, and the control file that specifies the format of the import file. SOLID *SpeedLoader* can directly use these files to load data into a SOLID database.

NOTE. The user name used for performing the export operation must have select rights on the table exported. Otherwise no data is exported.

Invoking SOLID *Export*

SOLID *Export* is invoked with the command `solexp`. If you invoke solexp with no arguments, you'll see a summary of the arguments with a brief description. The command line syntax is:

```
solexp [options] [servername] <username> <password> <tablename|*>
```

The possible options are

Option	Description
-c<dir>	Change working directory
-e<sql-string>	Execute SQL string for export
-f<filename>	Execute SQL string from file for export
-h, -?	Help = Usage
-l<filename>	Write log entries to this file
-L<filename>	Append log entries to this file
-o<filename>	Write exported data to this file

-s<schemaname>	Use only this schema for export

NOTE 1. The symbol * can be used to export all tables with one command. However, it cannot be used as a wildcard.

NOTE 2. The -tTABLENAME (Export table) option is still supported in order to keep old scripts valid.

SOLID *Data Dictionary*

SOLID *Data Dictionary* is a product for retrieving data definition statements from a SOLID database. SOLID *Data Dictionary* produces an SQL script that contains data definition statements describing the structure of the database. The generated script contains definitions for tables, views, procedures, sequences, and events.

NOTE 1. User and role definitions are not listed for security reasons.

NOTE 2. The user name used for performing the export operation must have select right on the tables. Otherwise the connection is refused.

Invoking SOLID *Data Dictionary*

SOLID *Data Dictionary* is invoked with the command `soldd`. If you invoke `soldd` with no arguments, you'll see a summary of the arguments with a brief description. The command line syntax is:

```
soldd [options] [servername] <username> <password> [tablename]
```

The possible options are:

Option	Description
-c<dir>	Change working directory
-h, -?	Help = Usage
-o<filename>	Write data definitions to this file
-O<filename>	Append data definitions to this file
-s<schemaname>	List definitions from this schema only
-x indexonly	List index definitions only
-x tableonly	List table definitions only

Example:

```
soldd -odatabase.sql "tcp database_server 1313" dbadmin f1q32j4
```

NOTE 1. If no table name is given, all definitions are listed to which the user has rights.

NOTE 2. The -ttablename option is still supported in order to keep old scripts valid.

SOLID *Remote Control* (Teletype)

With SOLID R*emote Control* (Teletype), commands can be given at the command line, command prompt, or by executing a script file that contains the commands.

NOTE. The user performing the administration operation must have administrator's rights, or the connection will be refused.

Invoking SOLID *Remote Control* (Teletype)

SOLID *Remote Control* (Teletype) is invoked with the command `solcon`. On Novell Netware, you start SOLID *Remote Control* (Teletype) with the command `load solcon` at the command prompt. SOLID *Remote Control* (Teletype) connects to the first server specified in the `Connect` parameter in the `solid.ini` file. If you start SOLID *Remote Control* (Teletype) with no arguments, you'll be prompted for the database administrator's user name and password. The command line syntax is:

```
solcon [options] [servername] [username password]
```

The possible options are:

Option	Description
-c<dir>	Change working directory
-e<string>	Execute command string
-f<filename>	Execute command string from file
-h, -?	Help = Usage

You can give the connection information at the command line to override the connect definition in `solid.ini`.

Example:

```
solcon "spx solid"
```

Also the administrator's user name and password can be given at the command line.

Example:

```
solcon "tcp localhost 1313" admin iohi4y
```

Using SOLID *Remote Control* (Teletype)

After the connection to the server is established, the command prompt appears.

Available commands are described in the following table:

Command	Abbreviation	Explanation
backup [*backup_directory*]	bak	Makes a backup of the database. The default backup directory is the one defined in configuration parameter General.Backup.Directory. The backup directory may also be given as an argument. For example, **backup abc** creates backup on directory 'abc'. All directory definitions are relative to the SOLID *Embedded Engine* working directory.
backuplist	bls	Displays a status list of last backups.
close	clo	Closes server from new connections; no new connections are allowed.
errorcode *SOLID_error_code*	ec	Displays a description of an error code. Give the code number as an argument. For example, 'e**rrorcode 10033**
exit	ex	Exits SOLID *Remote Control*.
help	?	Displays available commands.
makecp	mcp	Makes a checkpoint.
messages	mes	Displays server messages.
monitor {on | off} [user *username* | *user id*]	mon	Sets server monitoring on and off. Monitoring logs user activity and SQL calls to SOLTRACE.OUT file
open	ope	Opens server for new connections; new connections are allowed.
report *filename*	rep	Generates a report of server info to a file given as an argument.
shutdown	sd	Stops SOLID *Embedded Engine*.

status	sta	Displays server statistics.
status backup	sta bak	Displays status of the last started backup. The status can be one of the following: ■ If the last backup was successful or any backups have not been requested, the output is 0 SUCCESS. ■ If the backup is in process; for example, started but not ready yet, the output is 14003 ACTIVE. ■ If the last backup failed, the output is: *errorcode* ERROR where the *errcode* shows the reason for the failure.
throwout {*username* | *userid* | **all**	to	Exits users from SOLID *Embedded Engine*. To exit a specified user, give the user id as an argument. To throw out all users, use the keyword ALL as an argument.
userlist [**-l**] [*name* | *id*]	ul	Displays a list of users. option -l displays more detailed output.
version	ver	Displays server version info.
pid	pid	Returns server process id.
parameters [*name*]	par	Displays server parameter values. For example: ■ **parameter** used alone displays all parameters. ■ **parameter genera**l displays all parameters from section “general.” ■ **parameter general.readonly** displays a single parameter “readonly” from section “general.”
perfmon [**-c**]	pmon	Returns performance statistics from the server. The -c option returns all values as counter. By default, some values are averages/second.
trace {on | **off**} **sql** | **rpc** | **sync**	tra	Sets server trace on or off. This command is similar to the **monitor** command, but traces different entities and a different levels. By default, the output is witten to the SOLTRACE.OUT file.

You can execute all commands either using this interface or giving them at the command line with the -e option or in a text file with the -f option. Commands can be given using either the complete command name or its abbreviation.

You can also execute all SOLID *Remote Control* commands programmatically from an application using options of the SQL command “ADMIN COMMAND”. For example, you can start a backup with the SQL command ADMIN COMMAND ‘backup’.

SOLID *SQL Editor* (Teletype)

With SOLID *SQL Editor* (Teletype), statements can be given at the command line, command prompt, or by executing a script file that contains the SQL statements.

NOTE. The user performing SQL statements must have appropriate user rights on the corresponding tables, or the connection will be refused.

Starting SOLID *SQL Editor* (Teletype)

SOLID *SQL Editor* (Teletype) is started by entering the command `solsql.` On Novell Netware, you start SOLID *SQL Editor* (Teletype) with the command `load solsql` at the command prompt. SOLID *SQL Editor* (Teletype) connects by default to the first server specified in the `Connect` parameter in `solid.ini` file and prompts for a user name and password. The command line syntax is:

```
solsql [options] [servername] [username] [password] [filename]
```

The possible options are:

Option	Description
-a	Auto commit every statement
-c	Change working directory
-e<sql-string >	Execute SQL string
-f<filename>	Execute SQL string from file
-h, -?	Help = Usage
-o<filename>	Write result set to this file
-O<filename>	Append result set to this file
-s<schemaname>	Use only this schema
-t	Print execution time per command
-u	Expect input is in UTF-8format

-x onlyresults	Print only rows

NOTE: If user name and password are given as command line arguments also the server name must be given as a command line argument. Also if the name of the SQL script file is given as a command line argument (not with the option -f), the server name, user name and password must also be given as command line arguments.

Using SOLID *SQL Editor* (Teletype)

Executing SQL Statements

After the connection to the server has been established a command prompt appears. SOLID *SQL Editor* (Teletype) executes SQL statements terminated by a semicolon.

Example:

```
create table testtable (value integer, name varchar);

commit work;

insert into testtable (value, name) values (31, 'Duffy Duck');
select value, name from testtable;

commit work;

drop table testtable;

commit work;
```

Exiting SOLID SQL Editor

To exit from SOLID *SQL Editor* (Teletype) enter the command:

```
exit;
```

Executing an SQL Script

To execute an SQL script from a file, the name of the script file must be given as a command line parameter:

```
solsql server-name user-name password file-name
```

All statements in the script must be terminated by a semicolon. SOLID *SQL Editor* (Teletype) exits after all statements in the script file have been executed.

Example:

```
solsql "tcp localhost 1313" admin iohe4y tables.sql
```

NOTE: Remember to commit work at the end of the SQL script or before exiting SOLID *SQL Editor* (Teletype). If an SQL-string is executed with the option -e, commit can only be done using the -a option.

Tools Sample: Reloading a Database

This example demonstrates how a SOLID *Embedded Engine* database can be reloaded to a new one. At the same time the use of each SOLID tool is introduced with an example. This reload is a useful procedure since it shrinks the size of the database file `solid.db` to a minimum.

To Reload the Database:

1. Extract data definitions from the old database.
2. Extract data from the old database.
3. Replace the old database with a new one.
4. Load data definitions into a new database.
5. Load data into the new database.

Walkthrough

In this example the server name is SOLID and the protocol used for connections is Shared Memory. Therefore, the network name is “ShMem SOLID”. The database has been created with the user name “dbadmin” and the password “password”.

1. Data definitions are extracted with SOLID *Data Dictionary.* Use the following command line to extract an SQL-script containing definitions for all tables, views, procedures, sequences, and events. The default for the extracted SQL-file is `soldd.sql`.

   ```
   soldd "ShMem SOLID" dbadmin password
   ```

   With this command all data definitions are listed into one file, `soldd.sql` (the default name). As mentioned earlier, user and role definitions are not listed for security reasons. If the database contains users or roles, they need to be appended into this file.

2. All data is extracted with SOLID *Export.* The export results in control files (files with the extension `.ctr`) and data files (files with the extension `.dat`). The default file name is the same as the exported table name. In 16-bit environments, file names longer than

eight letters are concatenated. Use the following command line to extract the control and data files for all tables.

```
solexp "ShMem SOLID" dbadmin password *
```

With this command data is exported from all tables. Each table's data is written to an import file named `table_name.dat`. A separate control file `table_name.ctr` is written for each table name.

3. A new database can be created to replace the old one by deleting the `solid.db` and all `sol####.log` files from the appropriate directories. When SOLID *Embedded Engine* is started for the first time after this, a new database is created.

   NOTE. It is recommended that a backup is created of the old database before it is deleted. This can be done using SOLID *Remote Control* (Teletype).

4. Use the following command line to create a backup using SOLID *Remote Control* (Teletype):

   ```
   solcon -eBACKUP "ShMem SOLID" dbadmin password
   ```

   With this command a backup is created. The option -e precedes an administration command.

5. Load data definitions into the new database. This can be done using SOLID *SQL Editor* (Teletype). Use the following command line to execute the SQL-script created by SOLID *Data Dictionary*.

   ```
   solsql -fSOLDD.SQL "ShMem SOLID" dbadmin password
   ```

   With this command, data definitions are loaded into the new, empty database. Definitions are retrieved with the option -f from the file `soldd.sql`. Connection parameters are the same as in the earlier examples.

   The previous two steps can be performed together by starting SOLID *Embedded Engine* with the following command line. The option -x creates a new database, executes commands from a file, and exits. User name and password are defined as well.

   ```
   solid -Udbadmin -Ppassword -x execute:soldd.sql
   ```

6. Load data into the new database. This is be done SOLID *Speedloader.* To load several tables into the database a batch file containing a separate command line for each table is recommended. In Unix-based operating systems and in OS/2, using the wildcard symbol * is possible. Use either of the following command lines to load data into the new database.

   ```
   solload "ShMem SOLID" dbadmin password table_name.ctr
   ```

7. With this command data for one table is loaded. The server is online.

   Batch files that can be used are:

- Shell scripts in Unix environments
- `.com` -scripts in VMS
- `.bat` -scripts in Windows 95, 98 and NT

5

Administration with SQL Statements

This chapter tells you how to manage the database as well as its users and schema using SQL statements. You can use SOLID *SQL Editor* and many ODBC compliant tools for executing these SQL statements.

To automate these tasks, you may want to save the SQL statements to a file. You can use these files for rerunning your SQL statements later or as a document of your users, tables, and indexes.

About SOLID SQL Syntax

The SQL syntax is based on the ANSI X3H2-1989 level 2 standard including important ANSI X3H2-1992 (SQL2) extensions. User and role management services missing from previous standards are based on the ANSI SQL3 draft. For a more formal definition of the syntax, refer to *Appendix D SOLID Embedded Engine SQL Syntax* of this document.

Administering the Database

SOLID *Embedded Engine* provides the SQL-extension ADMIN COMMAND '*command*[*command_args*]' to perform basic administrative tasks, such as backups, performance monitoring, and shutdown.

You can use SOLID *Remote Control* program to perform the command options provided by ADMIN COMMAND. For details, read the “SOLID Remote Control (Teletype)” in Chapter 4.

You can find a short description of available commands by executing ADMIN COMMAND 'help'. For a formal definition of the syntax of these statements, refer to Appendix D, “SOLID SQL Syntax” in this guide.

Managing User Privileges and Roles

You can use SOLID *SQL Editor* and many ODBC compliant SQL tools to modify user privileges. Users and roles are created and deleted using SQL statements or commands. A file consisting of several SQL statements is called a SQL script.

In the SOLID directory, you will find an SQL script called `users.sql`, which gives an example of creating users and roles. You can run it using SOLID *SQL Editor*. To create your own users and roles, you can make your own script describing your user environment.

NOTE: All SQL statements must be terminated with a semicolon (;).

User Privileges

When using SOLID *Embedded Engine* in a multi-user environment, you may want to apply user privileges to hide certain tables from some users. For example, you may not want an employee to see the table in which employee salaries are listed, or you may not want other users to mess with your test tables.

SOLID *Embedded Engine* allows you to apply five different kinds of user privileges. A user may be able to view, delete, insert, update or reference information in a table or view. Any combination of these privileges may also be applied. A user who has none of these privileges to a table is not able to use the table at all.

User Roles

Privileges can also be granted to an entity called a role. A role is a group of privileges that can be granted to users as one unit. SOLID *Embedded Engine* allows you to create roles and assign users to certain roles.

NOTE: Same string can not be used both as a user name and a role name.

The following user and role names are reserved:

Reserved name	Description
PUBLIC	You can use this role to grant privileges to all users. When user privileges to a certain table are granted to the role `PUBLIC`, all current and future users have the specified user privileges to this table. This role is granted automatically to all users.

SYS_ADMIN_ROLE	This is the default role for the database administrator. This role has administration privileges to all tables, indexes and users. This is also the role of the creator of the database.
_SYSTEM	This is the schema name of all system tables and views.
SYS_CONSOLE_ROLE	This role has right to use SOLID *Remote Control*, but does not have other administration privileges.

Examples of SQL Statements

Below are some examples of SQL commands for administering users, roles and user privileges.

Creating Users

```
CREATE USER <username> IDENTIFIED BY <password>;
```

Only an administrator has the privilege to execute this statement. The following example creates a new user named `CALVIN` with the password `HOBBES`.

```
CREATE USER CALVIN IDENTIFIED BY HOBBES;
```

Deleting Users

```
DROP USER <username>;
```

Only an administrator has the privilege to execute this statement. The following example deletes the user named `CALVIN`.

```
DROP USER CALVIN;
```

Changing a Password

```
ALTER USER <username> IDENTIFIED BY <new password>;
```

The user `<username>` and the administrator have the privilege to execute this command. The following example changes `CALVIN's` password to `GUBBES`.

```
ALTER USER CALVIN IDENTIFIED BY GUBBES;
```

Creating Roles

```
CREATE ROLE <rolename>;
```

The following example creates a new user role named GUEST_USERS.

```
CREATE ROLE GUEST_USERS;
```

Deleting Roles

```
DROP ROLE <role_name>;
```

The following example deletes the user role named GUEST_USERS.

```
DROP ROLE GUEST_USERS;
```

Granting Privileges to a User or a Role

```
GRANT <user_privilege> ON <table_name> TO <username or role_name>;
```

The possible user privileges on tables are SELECT, INSERT, DELETE, UPDATE, REFERENCES and ALL. ALL will give a user or a role all five privileges mentioned above. EXECUTE privilege will give a user a right to execute a stored procedure. A new user has not any privileges.

The following example grants INSERT and DELETE privileges on a table named TEST_TABLE to the GUEST_USERS role.

```
GRANT INSERT, DELETE ON TEST_TABLE TO GUEST_USERS;
```

The following example grants EXECUTE privilege on a stored procedure named SP_TEST to user CALVIN.

```
GRANT EXECUTE ON SP_TEST TO CALVIN;
```

Granting Privileges to a User by Giving the User a Role

```
GRANT <role_name> TO <username>;
```

The following example gives the user CALVIN the privileges that are defined for the GUEST_USERS role.

```
GRANT GUEST_USERS TO CALVIN;
```

Revoking Privileges from a User or a Role

```
REVOKE <user_privilege> ON <table_name> FROM <username or role_name>;
```

The following example revokes the INSERT privilege on the table named TEST_TABLE from the GUEST_USERS role.

```
REVOKE INSERT ON TEST_TABLE FROM GUEST_USERS;
```

Revoking Privileges by Revoking the Role of a User

```
REVOKE <role_name> FROM <username>;
```

The following example revokes the privileges that are defined for the GUEST_USERS role from CALVIN.

```
REVOKE GUEST_USERS FROM CALVIN;
```

Granting Administrator Privileges to a User

```
GRANT SYS_ADMIN_ROLE TO <username>;
```

The following example grants administrator privileges to CALVIN, who now has all privileges to all tables.

```
GRANT SYS_ADMIN_ROLE TO CALVIN;
```

NOTE. If the autocommit mode is set OFF, you need to commit your work. To commit your work use the SQL statement COMMIT WORK;. If the autocommit mode is set ON the transactions are committed automatically.

Managing Tables

SOLID *Embedded Engine* has a dynamic data dictionary that allows you to create, delete and alter tables on-line. SOLID *Embedded Engine* tables are managed using SQL commands.

In the SOLID directory, you can find an SQL script named sample.sql, which gives an example of managing tables. You can run the script using SOLID *SQL Editor.*

Below are some examples of SQL statements for managing tables. For a formal definition of the SOL syntax of SOLID Embedded Engine, refer to *Appendix D SOLID Embedded Engine SQL Syntax* of this document.

TIP. If you want to see the names of all tables in your database, issue the SQL statement SELECT * FROM TABLES or use predefined command TABLES from SOLID *SQL Editor.* The table names can be found in the column TABLE_NAME.

Examples of SQL Statements

Below are some examples of SQL commands for administering tables.

Creating Tables

```
CREATE TABLE <table_name> (<column> <column type>
    [,<column> <column type>]...);
```

All users have privileges to create tables.

The following example creates a new table named TEST with the column I of the column type INTEGER and the column TEXT of the column type VARCHAR.

```
CREATE TABLE TEST (I INTEGER, TEXT VARCHAR);
```

Removing Tables

```
DROP TABLE <table_name>;
```

Only the creator of the particular table or users having SYS_ADMIN_ROLE have privileges to remove tables.

The following example removes the table named TEST.

```
DROP TABLE TEST;
```

Adding Columns to a Table

```
ALTER TABLE <table_name> ADD COLUMN <column_name>
    <column type>;
```

Only the creator of the particular table or users having SYS_ADMIN_ROLE have privileges to add or delete columns in a table.

The following example adds the column C of the column type CHAR(1) to the table TEST.

```
ALTER TABLE TEST ADD COLUMN C CHAR(1);
```

Deleting Columns from a Table

```
ALTER TABLE <table_name> DROP COLUMN
    <column_name>;
```

The following example statement deletes the column C from the table TEST.

```
ALTER TABLE TEST DROP COLUMN C;
```

NOTE. If the autocommit mode is set OFF, you need to commit your work before you can modify the table you altered. To commit your work after altering a table, use the SQL statement COMMIT WORK;. If the autocommit mode is set ON transactions are committed automatically.

Managing Indexes

Indexes are used to speed up access to tables. The database engine uses indexes to access the rows in a table directly. Without indexes, the engine would have to search the whole contents of a table to find the desired row. There are two kinds of indexes: non-unique indexes and unique indexes. A unique index is an index where all key values are unique. You can create as many indexes as you like to a single table. However, adding indexes slows down updates on that table.

SOLID *Embedded Engine* allows you to create and delete indexes using the following SQL statements. For a formal definition of the syntax of these statements, refer to *Appendix D SOLID Embedded Engine SQL Syntax* of this document.

Examples of SQL Statements

Below are some examples of SQL commands for administering indexes.

Creating an Index to a Table

```
CREATE INDEX <index_name> ON <table_name>
       (<column_name> [ASC | DESC]);
```

Only the creator of the particular table or users having SYS_ADMIN_ROLE have privileges to create or delete indexes.

The following example creates an index named `X_TEST` on the table `TEST` to the column `I`.

```
CREATE INDEX X_TEST ON TEST (I);
```

Creating a Unique Index to a Table

```
CREATE UNIQUE INDEX <index_name> ON <table_name>
       (<column_name>);
```

The following example creates a unique index named `UX_TEST` on the table `TEST` to the column `I`.

```
CREATE UNIQUE INDEX UX_TEST ON TEST (I);
```

Deleting an Index

```
DROP INDEX <index_name>;
```

The following example deletes the index named `X_TEST`.

```
DROP INDEX X_TEST;
```

NOTE. If the autocommit mode is set OFF, you need to commit your work before you can modify the table on which you altered the indexes. To commit your work after modifying

indexes, use the SQL statement `COMMIT WORK;`. If the autocommit mode is set ON the transactions are committed automatically.

Primary Keys

A primary key is a column or combination of columns that uniquely identify each record in a table. Primary keys like indexes speed up access to tables. The difference between primary keys and indexes in SOLID *Embedded Engine* is that the primary key cluster data in the database according to the key values.

This behavior differs from the default clustering in SOLID *Embedded Engine*, where the data is clustered according to the insertion time only.

Foreign Keys

A foreign key is a column or group of columns within a table that refers to, or relates to, some other table through its values. The foreign key must always include enough columns in its definition to uniquely identify a row in the referenced table. The main reason for defining foreign keys is to ensure that rows in one table always have corresponding rows in another table; that is, to ensure that referential integrity of data is maintained.

6

Network Connections

Communication between Client and Server

The database server and client transfer information between each other through the computer network using a communication protocol.

When a database server process is started, it will publish at least one network name that distinguishes it in the network. We say that the server starts to *listen* to the network using the given network name. The network name consists of a communication protocol and an server name.

To establish a connection from a client to to a server they both have to be able to use the same communication protocol. The client has to know the network name of the server and often also the location of theserver in the network. The client process uses the network name to specify which server it will *connect* to.

This chapter will give you information on how to administer network names.

Network Names for SOLID *Embedded Engine*

The network name of a server consists of a *communication protocol* and a *server name.* This combination identifies the embedded engine in the network. The network names are defined in configuration file `solid.ini` in `[Com]` section with the `Listen` parameter. The `solid.ini` file should be located in the embedded engine program's working directory or in the directory set by the `SOLIDDIR` environment variable.

A server may use an unlimited number of network names. To make establishing connections easier all components of network names are case insensitive.

Network names are managed on the `NETWORK` page in SOLID *Remote Control* or directly by editing the server configuration file `solid.ini`. An example of an entry in `solid.ini`:

```
[Com]
```

```
Listen = tcpip 1313, nmpipe solid
```

The example contains two network names which are separated by a comma. The first one uses the protocol TCP/IP and the service port 1313, the other one uses the Named Pipes protocol with the name ‘solid’. In our example the ‘tcpip’ and ‘nmpipe’ are communication protocols while ‘1313’ and ‘solid’ are server names.

If the `Listen` parameter is not set in the SOLID.INI file, the environment dependent defaults as used.

NOTE 1. When a database server process is started it publishes the network names it starts to listen to. This information is also written to a file named `solmsg.out` in the located in the same directory as the `solid.ini` file.

NOTE 2. Network names must be unique within one host computer. For example, you cannot have two database servers running, both listening to the same TCP/IP port in one host, but it is possible that the same port number is in use in different hosts. Exceptions to this are the NetBIOS and IPX/SPX protocols, which require that used server names are unique throughout the whole network.

To Add a Network Name for the Server

1. Open the `solid.ini` file located in the working directory of your SOLID *Embedded Engine* process.
2. View the parameter `Listen` in the `[Com]` section.
3. Add a new network name to the list of network names. Use a comma (,) to separate nertwork names.
4. Save the changes.

   You need to restart the SOLID *Embedded Engine* process to activate the changes.

To Modify a Network Name

1. Open the `solid.ini` file located in the working directory of your SOLID *Embedded Engine* process.
2. View the parameter `Listen` in the `[Com]` section.
3. Edit the network name in the list of network names.
4. Save the changes.

You need to restart the SOLID *Embedded Engine* process to activate the changes.

To Remove a Network Name from the Server

1. Open the `solid.ini` file located in the working directory of your SOLID *Embedded Engine* process.
2. View the parameter `Listen` in the `[Com]` section.
3. Remove the network name from the list of network names.
4. Save the changes.

You need to restart the SOLID *Embedded Engine* process to activate the changes.

NOTE: The modifications to network names does not become active immediately after editing the `solid.ini` file.. You must restart the SOLID *Embedded Engine* process.

HINT: You can disable a network name using option -d after the protocol name in the network name:

```
tcp -d hobbes 1313, shmem -d solid
```

Network Name for Clients

The network name of a client consists of a *communication protocol*, an optional *host computer name* and a *server name* . By this combination the client specifies the server it will establish a connection to. The communication protocol and the server name must match the ones that the server is using in its network listening name. Most protocols need additionally the host computer name to be specified if the client and server are running on different machines. All components of the client's network name are case insensitive.

The client's network names are defined in the configuration file `solid.ini` in the `[Com]` section with the `Connect` parameter. The `solid.ini` file should be located in the application program's working directory or in the directory set by the `SOLIDDIR` environment variable.

The following connect line in the `solid.ini` of the application workstation will connect an application (client) using the TCP/IP protocol to a SOLID server running on a host computer named 'spiff' and listening with the name (port number in this case) '1313'.

```
[Com]
Connect = tcpip spiff 1313
```

If the `Connect` parameter is not found in the configuration file `solid.ini` the client uses the environment dependent default instead. The defaults for the `Listen` and `Connect` parameters are selected so that the application (client) will always connect to a local SOLID server listening with a default network name. So the local communication (inside one machine) does not necessarily need a configuration file for establishing a connection.

NOTE 1. When the connection is requested by client program using the `SQLConnect` function the network name of the server is given as a Data Source Name parameter for that function. If the given name is not an empty string, its contents are used as a network name and the `Connect` parameter in the configuration file is omitted. If an empty string is passed, the possibly existing `Connect` parameter is used.

NOTE 2. In the Windows (95, 98, NT) operating system, the connection can be made by using the SOLID *ODBC Driver.* When a client program is using the SOLID *ODBC Driver,* the network name of the server can be used as the ODBC Data Source Name and the `Connect` parameter in the configuration file is not used.

Communication Protocols

A client process and SOLID *Embedded Engine c*ommunicate with each other by using computer networks and network protocols. A network operating system - for example, IBM LAN Server or Novell NetWare - is not necessarily needed. You only need a functioning communication protocol for both ends. Supported communication protocols depend on the type of computer and network you are using.

The following paragraphs describe the supported communication protocols and common environments that may be used and also show the required forms of network names for the various protocols.

Shared Memory

Usually the fastest way two processes can exchange information is to use Shared Memory. This can be used only when the embedded engine and application processes are both running in the same computer. The Shared Memory protocol uses a shared memory location for moving data from one process to another.

To use the Shared Memory protocol in SOLID *Embedded Engine,* select `ShMem` from the list of protocols in SOLID *Remote Control* and enter server name. The server name has to be unique only in this computer.

The Format Used in the `solid.ini` File

```
Server      Listen = shmem <server name>
Client      Connect = shmem <server name>
```

NOTE 1. Server names must be character strings less than 128 characters long.

TCP/IP

The TCP/IP protocol is typically used for communicating to a server process running under a UNIX operating system. When starting an server using the TCP/IP protocol, you must reserve a port number for it. You will find reserved port numbers in the `/etc/services` file of your system. Select a free number greater than 1024 since smaller numbers are usually reserved for the operating system.

To use the TCP/IP protocol, select `TCP/IP` in the list of protocols in SOLID *Remote Control* and enter a non-reserved port number.

The Format Used in the `solid.ini` File

Server `Listen = tcpip <server port number>`

Client `Connect = tcpip [host computer name] <server port number>`

NOTE 1: If the server is running in the same computer with the client program, the host computer name need not be specified. The client computer has to have the used host name listed in its `etc/hosts` file or it must be recognized by the DNS (Domain Name Server). You can also give the host computer's TCP/IP address in dotted decimal format (e.g.: 194.53.94.97) instead of its host name.

NOTE 2: On Windows 95, Windows 98, Windows NT and UNIX the TCP/IP protocol is usually included in the operating system. On other environments (like VAX/VMS) the TCP/IP software needs to be installed to the system. For a list of supported TCP/IP software, contact your SOLID Embedded Engine dealer.

NOTE 3: Using option `-i<ip-address>or -i<host name>` SOLID *Embedded Engine* listens only to the specified IP-address or host name. For example, a server with the following setting in `solid.ini`

```
[com]
Listen = tcp -i127.0.0.1 1313
```

accepts connection requests only from inside the same machine, either referred by IP-address 127.0.0.1 or with the name 'localhost', if the DNS is correctly configured.

UNIX Pipes

The UNIX domain sockets (UNIX Pipes, Named Pipes, portals) are typically used when communicating between two processes running in the same UNIX machine. UNIX Pipes usually have a very good throughput. They are also more secure than TCP/IP since Pipes can only be accessed from applications that run on the computer where the server executes.

When starting a server using UNIX Pipes, you must reserve a unique listening name (inside that machine) for the server, for instance, 'solid'. Because UNIX Pipes handle the UNIX domain sockets as standard file system entries, there is always a corresponding file created for every listened pipe. In SOLID *Embedded Engine's* case, the entries are created under the path '/tmp'. Our example listening name 'solid' creates the directory '/tmp/solunp_SOLID' and shared files into that directory. The '/tmp/solunp_' is a constant prefix for all created objects while the latter part ('SOLID' in this case) is the listening name in upper case format.

The Format Used in the `solid.ini` File

Server `Listen = upipe <server name>`
Client `Connect = upipe <server name>`

NOTE 1: Server and clientprocesses must run in the same machine in order to use UNIX Pipes for communication.

NOTE 2: The server process must have a "write" permission to the directory '/tmp'.

NOTE 3: The client accessing UNIX Pipes must have an "execute" permission to the directory '/tmp'.

NOTE 4: The directory '/tmp' must exist.

NOTE 5: UNIX Pipes cannot be used in SCO UNIX.

NetBIOS

The NetBIOS protocol is commonly used in the Windows (95, 98, NT) operating systems.

To use NetBIOS protocol, select `NetBIOS` in the list of available protocols in SOLID *Remote Control* Network page, and enter a non-reserved server name.

The Format Used in the `solid.ini` File

Server `Listen = netbios [-aLANA_NUMBER] <server name>`
Client `Connect = netbios [-aLANA_NUMBER] <server name>`

NOTE 1. The server name must be a character string at most 16 characters long. It may not begin with an asterisk (*).

NOTE 2. In the above format the optional -aLANA_NUMBER is used to override the default value of the LANA number.

NOTE 3. In Windows NT the available LANA numbers can be checked using the Network Setup found in the Control Panel. The default value 0 may not be generally very good. You should choose the one(s) where the protocol stack matches the other computers you are using. The LANA number (Network Route: Nbf->Elnk3->Elnk31) that uses NetBEUI as a

transport usually functions quite smoothly when used for SOLID communication.

NOTE 4. The server names have to be unique in the whole network. Establishing a connection or starting the listener using the NetBIOS protocol may be somewhat slow because of the checks needed for uniqueness.

NOTE 5. SOLID *Embedded Engine* and SOLID Client versions 2.2 and newer use all available LANA numbers by default. This makes it unnecessary to specify explicitly which LANA number the application or embedded engine should use. For backward compatibility the parameter '-aLANA_NUMBER' remains available.

Named Pipes

Named Pipes is a protocol commonly used in the Windows (95, 98, NT) operating systems.

Windows 95 and Windows 98 support Named Pipes only in client end communication. Windows NT supports Named Pipes both in server and clint communication.

The Format Used in the `solid.ini` File

Server `Listen = nmpipe <server name>`

Client `Connect = nmpipe [host computer name] <server name>`

NOTE 1: The server names must be character strings at most 50 characters long.

NOTE 2: If the server is running in the same computer with the application program, the host computer name should not be specified.

NOTE 3: In order to connect to the SOLID *Embedded Engine* for Windows NT through Named Pipes, the user must have at least the same rights as the user, who started the server. For example if an administrator starts the server only, users with administrator's rights are able to connect to the server through Named Pipes. Similarly if a user with normal user's rights starts theserver all users with greater rights are able to connect the server through Named Pipes. If a user doesn't have proper rights, SOLID Communication Error 21306 message will be given.

NOTE 4: It is not recommended to use the Named Pipes communication from SOLID *Remote Control*. The asynchronous nature of SOLID *Remote Control* communication may cause problems with Named Pipes.

DECnet

The DECnet protocol is used to connect to an embedded engine running on a OpenVMS system. To use this protocol in Windows NT, Windows 98 or Windows 95, you need to have PATHWORKS 32 installed to your client computer.

To use the DECnet protocol, select `DECnet` in the list of protocols in SOLID *Remote Control* Network Page and enter a non-reserved server name.

The Format Used in the `solid.ini` File

```
Server      Listen = decnet <server name>
Client      Connect = decnet <node name> <server name>
```

NOTE: To establish a connection the DECnet node name of the server machine is configured to your node database. The node name can be given either as a node number such as '1.1' or as a node name such as 'VAX1'.

IPX/SPX

The IPX/SPX protocol is used to communicate with SOLID *Embedded Engine* for Novell Netware.

SOLID *Embedded Engine* for Novell Netware starts listening with the default listening name `SOLID` if no listening name is specified in the configuration file `solid.ini`. When SOLID *Embedded Engine* starts, it prints out the network and node information of the server machine.

The SOLID server listening name can be given as a character string or as a socket number. If the given network name is a valid socket number, that is, hex number with exactly 4 characters (e.g. 400F) SOLID *Embedded Engine* starts listening in the given port. If the network name could not be interpreted as a socket number it is treated as a server name character string and is published using Novell NetWare SAP (Service Advertising Protocol).

Connecting to a SOLID *Embedded Engine* using SAP needs specifying only the correct server name in `Connect` parameter. If the server is listening using some given port, the full NLM server info (see comment below) has to be given.

To use the IPX/SPX protocol, select `IPX/SPX` in the list of protocols in SOLID *Remote Control* and enter a non-reserved server name.

The Format Used in the `solid.ini` File

```
Server      Listen = spx {<server name> | <socket number>}
Client      Connect = spx {<NLM server info> |
                    <server name>}
```

NOTE 1. The server names must be less than 48 characters long.

NOTE 2. In the above format, <NLM server info> stands for a string containing the network number, the node number and the socket number separated by colons. For example, <NLM

server info> for network 1, node 1, socket number 1313 is 00000001:000000000001:1313. You can abbreviate the information by removing the leading zeros. The previous embedded engine info could thus also be written as 1:1:1313.

<server name> stands for an alphanumeric string.

NOTE 3. The possibility to use socket numbers as the listening name is supported mainly for historical reasons. SAPing is intended to be the primary method.

NOTE 4. After removing a network name or shutting down SOLID *Embedded Engine* using SOLID *Remote Control* the server name used may still remain reserved for up to one minute although everything completes successfully. The error 'network name in use' is displayed if SOLID *Embedded Engine* is restarted immediately. This is a 'normal' NetWare SAP feature and happens more often if your network consists of more than one NetWare server. Propagating the SAP cancellation packets to every network node may take a while.

A Summary of Protocols

The following tables summarize the possible operating systems and required forms for network names for the various communication protocols.

NOTE: The following tables contain the protocols and operating systems that were available when this guide was printed. For an updated list, contact your SOLID *Embedded Engine* dealer.

Embedded Engine Protocols and Network Names

Protocol	Server OS	Network name in `solid.ini` file
Shared Memory	Windows 95 Windows 98 Windows NT	Listen = shmem <server>
NetBIOS	Windows 95 Windows 98 Windows NT	Listen = netbios <server>
Named Pipes	Windows NT	Listen = nmpipe <server>
IPX/SPX	Novell Netware	Listen = spx <server> Listen = spx <socket number>
TCP/IP	Windows 95 Windows 98 Windows NT UNIX	Listen = tcpip <port>
UNIX Pipes	UNIX	Listen = upipe <server>

Application Protocols and Network Names

Protocol	Client OS	Network name in `solid.ini` file
Shared Memory	Windows 95 Windows 98 Windows NT	Connect = shmem <server>
NetBIOS	Windows 95 Windows 98 Windows NT	Connect = netbios <server>
Named Pipes	Windows 95 Windows 98 Windows NT	Connect = nmpipe [host] <server>
IPX/SPX	Novell Netware Windows 95 [1] Windows 98 [1] Windows NT [1]	Connect = spx <server> Connect = spx <NLM server info>
TCP/IP	Windows 95 Windows 98 Windows NT UNIX	Connect = tcpip [host] <port>

UNIX Pipes	UNIX	Connect = upipe <server>
DECnet	Windows 95 [2] Windows 98 [2] Windows NT [2]	Connect = decnet <host> <server>

1) requires Novell's Netware Client for Windows 95 and Windows NT

2) requires Digital PATHWORKS 32 for Windows 95 and Windows NT

Logical Data Source Names

SOLID Clients support Logical Data Source Names. These names can be used for giving a database a descriptive name. This name can be mapped to a network name in three ways:

1. Using the parameter settings in the application's `solid.ini` file.
2. Using the Windows operating systems registry settings.
3. Using settings in a solid.ini file located in the Windows directory.

This feature is available on all supported platforms. However, on non-Windows platforms, only the first method is available.

A SOLID Client attempts to open the file `solid.ini` first from the directory set by the `SOLIDDIR` environment variable. If the file is not found from the path specified by this variable or if the variable is not set, an attempt is made to open the file from the current working directory.

To define a Logical Data Source Name using the `solid.ini` file, you need to create a `solid.ini` file containing the section `[Data Sources]`. In that section you need to enter the 'logical name' and 'network name' pairs that you want to define. The syntax of the parameters is the following:

```
[Data Sources]
<logical name> = <network name>, <Description>
```

In the description field, you may enter comments on the purpose of this logical name.

If, for example, you want to define a logical name for the application 'My_application', and the database is located in a UNIX server that you want to connect to by using TCP/IP. You should include the following lines to the solid.ini file, which you need to place in the working directory of your application:

```
[Data Sources]
My_application = tcpip irix 1313, Sample data source
```

When your application now calls the Data Source ‘My_application’, the SOLID Client maps this to a call to ‘tcpip irix 1313’.

On Windows platforms (Windows 95, Windows 98 and Windows NT), the registry can be used to map Data Sources. These follow the standards of mapping ODBC Data Sources on a system.

In Windows 95, Windows 98 and Windows NT, a Data Source may be defined in the Windows Registry. The entry is searched from the path “software\odbc\odbc.ini”

1. first under the root HKEY_CURRENT_USER and if not found,
2. under the root HKEY_LOCAL_MACHINE.

The order of resolving a Data Source name in Windows systems is the following:

1. Look for the Data Source Name from the `solid.ini` file in the current working directory, under the section `[Data Source]`
2. Look for the Data Source Name from the following registry path `HKEY_CURRENT_USER\software\odbc\odbc.ini\DSN`
3. Look for the Data Source Name from the following registry path `HKEY_LOCAL_MACHINE\software\odbc\odbc.ini\DSN`

In case an application uses normal ODBC Data Sources, the network name is mapped normally using the methods that are provided in the ODBC Driver Manager.

7
Configuration

This chapter describes how to configure the SOLID *Embedded Engine* to meet your environment, performance, and operation needs. It includes SOLID *Embedded Engine* parameters and their settings. The topic Managing Parameters in this chapter gives you step-by-step instructions on how to view and set the parameter values on the `Parameters` page in SOLID *Remote Control*.

Configuration File and Default Settings

When SOLID *Embedded Engine* is started, it attempts to open the configuration file `solid.ini` first from the directory set by `SOLIDDIR` environment variable. If the file is not found from the path specified by this variable or if the variable is not set, an attempt is made to open the file from the current working directory.

The configuration values for the embedded engine parameters are included in this file. If the file does not exist, SOLID *Embedded Engine* will use default settings for the parameters. Also, if a value for a parameter is not set in the `solid.ini` file, SOLID *Embedded Engine* will use a default value for the parameter. The default values depend on the operating system you are using.

Generally, default settings offer good performance and operability, but in some cases modifying some parameter values can improve performance.

Most Important Parameters

The following paragraphs will explain the most important SOLID *Embedded Engine* parameters and their default settings. See *Appendix B, "Configuration Parameters"* of this manual for a description of all parameters.

[Com]
Connect
Listen

The parameter `Connect` in the `[Com]` section defines a network name for an application program. The application program will establish a connection to an embedded engine program with a similar `Listen` network name. The format for these parameters is explained in the chapter *Communication protocols*.

If the connect information is defined in the application program with the `SQLConnect` function, this parameter is ignored. In the Windows operating systems the connection can be made by using SOLID *ODBC driver*. When an application program is using a SOLID *ODBC driver* the ODBC Data Source Name is used and the `Connect` parameter has no effect. The `solid.ini` file, which includes the `Connect` parameter, must be located in the application program's working directory or in the directory set by `SOLIDDIR` environment variable.

The following connect line will connect a client program using the TCP/IP protocol to a SOLID Embedded Engine running in a computer named 'spiff' and server port number '1313'.

connect = tcpip spiff 1313

[IndexFile]
FileSpec_[1...N]

In SOLID *Embedded Engine* data and indexes are stored in the same logical files. The term 'index file' is used here as a synonym for the term 'database file'.

The `FileSpec` parameter describes the location and the maximum size of the index file (database file). You can use it to define the location and maximum value the index file may grow to.

The `FileSpec` parameter accepts the following three arguments:

- database file name
- max filesize
- device number (optional)

You can also use the `FileSpec` parameter to divide the index file into multiple files and onto multiple disks. To do this, specify another `FileSpec` parameter identified by the number `2`. The index file will be written to the second file if it grows over the maximum value of

the first `FileSpec` parameter. The default value for this parameter is solid.db, 2147483647 (which equals 2 GB expressed in bytes).

```
FileSpec_1=SOLID.DB 2147483647
```

In the following example, the parameters divide the index file on the disks C:, D: and E: to be split after growing larger than 1 GB (=1073741824 bytes).

```
FileSpec_1=c:\soldb\solid.1 1073741824  1
FileSpec_2=D:\soldb\solid.2 1073741824  2
FileSpec_3=G:\soldb\solid.3 1073741824  3
```

NOTE. The index file locations entered must be valid path names in the server's operating system. For example, if the server runs on a UNIX operating system, path separators must be slashes instead of backslashes.

Although the database files reside in different directories, the file names must be unique. In the above example, it is assumed that C:, D: and E: partitions reside on separate physical disks.

Splitting the index file on multiple disks will increase the performance of the server because multiple disk heads will access the data in your index file. There is no limit to the number of index files you may use.

If the database file is split into multiple physical disks, then multithreaded SOLID Embedded Engine is capable of assigning a separate disk I/O thread for each device. This way the server can perform database file I/O in a parallel manner.

[General]
BackupDirectory

Backups of the database, log files and the configuration file `solid.ini` are copied to the backup directory. The default directory '`backup`' is a directory relative to your SOLID directory. For example if the parameter is

```
BackupDirectory= bu
```

then the backup will be written to a directory that is a sub-directory of the SOLID directory. You may also specify a absolute path name for the directory. For example:

```
BackupDirectory=e:\backup\solid
```

The backup directory must exist and it must have enough disk space for the backup files. It can be set to any existing directory except the database file directory, the log file directory or the working directory.

NOTE. The backup directory entered must be a valid path name in the server's operating

system! For example if the server runs on a UNIX operating system, path separators must be slashes instead of backslashes.

[Logging]
FileNameTemplate

The transaction log files are created automatically to the directory specified and by using the filename structure specified by the parameter `FileNameTemplate` in the `Logging` section. For example, the following setting

```
FileNameTemplate = d:\logdir\sol#####.log
```

instructs SOLID *Embedded Engine* to create log files to directory `d:\logdir` and to name them sequentially starting from `sol00001.log`.

[Sorter]
TmpDir_[1...N]

The `TmpDir[1...N]` parameter in the `Sorter` section specifies the directory (or directories) that can be used for the external sorter algorithm which is used for sorting processes that do not fit in main memory. All temporary files used by the external sort are created in this directory (or directories) and are automatically deleted. Setting this parameter enables the use of external sorter.

[IndexFile]
CacheSize

The `CacheSize` parameter (the default value depends on the server operating system) defines the amount of main memory the server allocates for the cache. Although SOLID *Embedded Engine* is able to run with a small cache size, a larger cache size speeds up the server. The cache size needed depends on the size of the index file, the number of connected users, and the nature of the operations executed against the server.

[Srv]
Threads

The `Threads` parameter in the `[Srv]` section defines the amount of threads the SOLID *Embedded Engine* will use in addition to the communication, I/O and log manager threads. The default value is two threads for embedded engine use. The optimum number of threads depends on the number of processors available. Finding the value that provides the best performance requires experimentation. A good formula to start with is:

threads= (2 x number of processors) + 1

[SQL]
Info

The `Info` parameter in the `[SQL]` section specifies the tracing level on the SQL parser and optimizer as an integer between 0 (no tracing) and 9 (extensive trace outputting). Trace information will be output to the file named `soltrace.out` in the SOLID directory.

[Com]
Trace
TraceFile

These parameters control the outputting of network trace information vital to solving possible network problems. By setting the parameter `Trace` to the value `Yes`, SOLID *Embedded Engine* starts logging trace information on network messages to the file specified in the `TraceFile` parameter.

Managing Parameters

SOLID *Embedded Engine* parameters and their values can be viewed and modified by editing the `solid.ini` file in the SOLID directory.

To View and Set Configuration Parameter Values

1. Open the `solid.ini` file located in the working directory of your SOLID *Embedded Engine* process.
2. View the value of the parameter
3. If necessary add the section, parameter and parameter's value.
4. Save the changes.

You need to restart the SOLID *Embedded Engine* process to activate the changes.

The parameters displayed are the parameters currently active in the server. If you have not set a parameter value, the displayed value is the default value for the parameter. The default values are set at start-up and depend on the operating system SOLID *Embedded Engine* runs on.

NOTE 1. To force a parameter value change to take effect you must shut down and restart the SOLID *Embedded Engine* process.

NOTE 2. The new parameter values are not checked by the server. Setting an unreasonable value for a parameter may result in an operation failure the next time the server process is started. Do not set a parameter to a random value unless you know what you are doing. Use the default parameter values as an indication on the value range.

Constant Parameter Values

The values of some parameters were set when the database was created and they cannot be modified afterwards.

If you want to use different constant values, you have to create a new database. Before creating a new database, set new constant values by editing the `solid.ini` file in the SOLID directory.

The example below sets a new block size for the index file by adding the following lines to the `solid.ini` file:

```
[Indexfile]
Blocksize=4096
```

After editing and saving the `solid.ini` file, move the old database and log files, and start SOLID *Embedded Engine*. The server program will create a new database with the new constant values from the `solid.ini` file.

8

Performance Tuning

This chapter discusses techniques that you can use to improve the performance of SOLID *Embedded Engine*.

Tuning SQL Statements and Applications

Tuning the SQL statements, especially in applications where complex queries are involved, is generally the most efficient means of improving the database performance.

You should tune your application before tuning the RDBMS because:

- during application design you have control over the SQL statements and data to be processed
- you can improve performance even if you are not familiar with the internal working of the RDBMS you are going to use
- if your application is not tuned well, it will not run well even on a well-tuned RDBMS

So, find out what data your application processes, what are the SQL statements used and what operations the application performs on the data.

Using SOLID Server Diagnostic Tools

SOLID *Embedded Engine* provides the following tools that may be helpful in tuning applications:

- the SQL info facility
- the EXPLAIN PLAN statement

Read *Chapter 9, "Diagnostics and Troubleshooting"* for additional information on how to use these tools.

Indexes

You can use indexes to improve the performance of queries. A query that references an indexed column in its WHERE clause can use the index. If the query selects only the indexed column, the query can read the indexed column value directly from the index, rather than from the table.

If a table has a primary key, SOLID *Embedded Engine* orders the rows on disk in the order of the values of the primary key. Otherwise the rows are ordered using the ROWID, that is, the rows are stored on disk in the order they are inserted into the database.

Indexes improve the performance of queries that select a small percentage of rows from a table. You should consider using indexes for queries that select less than 15% of table rows.

Full table scan

If a query does not use an index, SOLID *Embedded Engine* must perform a full table scan to execute the query. This involves reading all rows of a table sequentially. Each row is examined to determine whether it meets the criteria of the query's WHERE clause. Finding a single row with an indexed query can be substantially faster than finding the row with a full table scan. On the other hand, a query that selects more than 15% of a table's rows may be performed faster by a full table scan than by an indexed query.

To perform a full table scan, every block in the table is read. For each block, every row stored in the block is read. To perform an indexed query the rows are read in the order in which they appear in the index, regardless of which blocks contain them. If a block contains more than one selected row it may be read more than once. So, there are cases when a full table scan requires less I/O than an indexed query.

Concatenated indexes

An index can be made up of more than one column. Such an index is called a concatenated index. It is recommended to use concatenated indexes when possible.

Whether or not a SQL statement uses a concatenated index is determined by the columns contained in the WHERE clause of the SQL statement. A query can use a concatenated index if it references a leading portion of the index in the WHERE clause. A leading portion of an index refers to the first column or columns specified in the CREATE INDEX statement.

Example:

```
create index job_sal_deptno on emp(job, sal, deptno);
```

This index can be used by these queries:

```
select * from emp where job = 'clerk' and sal =
```

```
   800 and deptno = 20;
select * from emp where sal = 1250 and job = salesman;
select job, sal from emp where job = 'manager' ;
```

The following query does not contain the first column of the index in its WHERE clause and cannot use the index:

```
select * from emp where sal = 6000;
```

Choosing columns to index

The following list gives guidelines in choosing columns to index:

- index columns that are used frequently in WHERE clauses
- index columns that are used frequently to join tables
- index columns that are used frequently in ORDER BY clauses
- index columns that have few of the same values or unique values in the table.
- do not index small tables (tables that use only a few blocks) because a full table scan may be faster than an indexed query
- if possible choose a primary key that orders the rows in the most appropriate order
- if only one column of the concatenated index is used frequently in WHERE clauses, place that column first in the CREATE INDEX statement
- if more than one column in concatenated index is used frequently in WHERE clauses, place the most selective column first in the CREATE INDEX statement

Tuning Memory Allocation

Tuning Your Operating System

Your operating system may store information in

- real memory
- virtual memory
- expanded storage
- disk

Your operating system may also move information from one location to another. Depending on your operating system, this movement is called paging or swapping. Many operating systems page and swap to accommodate large amounts of information that do not fit into real

memory. However, this takes time. Excessive paging or swapping can reduce the performance of your operating system and indicates that your system's total memory may not be large enough to hold everything for which you have allocated memory. You should either increase the amount of total memory or decrease the amount of database cache memory allocated.

Database Cache

The information used by SOLID *Embedded Engine* is stored either in memory or on disk. Since memory access is faster than disk access, it is desirable for data requests to be satisfied by access to memory rather than access to disk.

The basic element of the database server memory management system is a pool of central memory buffers of equal size. The size of the memory buffers and their amount can be configured to meet the demands of different application environments.

Database cache uses available memory to store information that is read from the hard disk. When an application next time requests this information, the data is read from memory instead of from the hard disk. The default value of cache depends on the platform used and can be changed by changing the `CacheSize` parameter. Increasing the value is recommended when there are several concurrent users.

The following values can be used as a starting point:

- a dedicated server with 16 MB RAM: Cachesize 4 MB
- a dedicated server with 32 MB RAM: Cachesize 10 MB
- a dedicated server with 64 MB RAM: Cachesize 30 MB

NOTE. You should increase the value of `Cachesize` very carefully. Too large a value leads to very poor performance.

Tuning I/O

The performance of many software systems is inherently limited by disk I/O. Often CPU activity must be suspended while I/O activity completes.

Distributing I/O

Disk contention occurs when multiple processes try to access the same disk simultaneously. To avoid this, move files from heavily accessed disks to less active disks until they all have roughly the same amount of I/O.

Follow these guidelines:

- use a separate disk for log files

- divide your database into several files and place each of these database files on a separate disk
- consider using a separate disk for the external sorter

Sorting

SOLID *Embedded Engine* does all sorting by default in memory. The amount of memory used for sorting is determined by the parameter `SORTARRAYSIZE` in the `[SQL]` section. If the amount of data to be sorted does not fit into the allocated memory, you may want to increase the value of the parameter `SORTARRAYSIZE`. If there is not enough memory to increase the value of `SORTARRAYSIZE` you should activate external sort that stores intermediate information to disk.

The external disk sort is activated by adding the following section and parameters in the configuration file `solid.ini`:

```
[sorter]
TmpDir_1 = c:\tmp
```

Additional sort directories are added with similar definitions:

```
[sorter]
TmpDir_1 = c:\tmp
TmpDir_2 = d:\tmp
TmpDir_3 = e:\tmp
```

Defining more than one sorter temporary directory on separate physical disks significantly improves sort performance by balancing the I/O load to multiple disks.

Tuning Checkpoints

Checkpoints affect:

- recovery time performance
- runtime performance

Frequent checkpoints can reduce the recovery time in the event of a system failure. If the checkpoint interval is small, then relatively few changes to the database are made between checkpoints and relatively few changes must be recovered.

Checkpoints cause SOLID *Embedded Engine* to perform I/O, so they momentarily reduce the runtime performance. This overhead is usually small.

9

Diagnostics and Troubleshooting

This chapter provides information on the following SOLID *Embedded Engine* diagnostic tools:

- SQL info facility and the EXPLAIN PLAN statement used to tune your application and identify inefficient SQL statements in your application.
- Network trace facility used to trace the server communication
- Ping facility used to trace client communication

You can use these facilities to observe performance, troubleshooting, and produce high quality problem reports.

In addition, this chapter describes how, using SOLID *Embedded Engine's* diagnostic tools, you can capture all relevant information about a problem quickly produce a problem report under various categories, such as SQL API, ODBC Driver, JDBC Driver, etc.

Observing Performance

The SQL Info Facility

Run your application with the SQL Info facility enabled. The SQL Info facility generates information for each SQL statement processed by SOLID *Embedded Engine*.

SQL Info levels

Info value	Information
0	no output
1	table, index, and view info in SQL format
2	SQL execution graphs
3	some SQL estimate info, Solid selected key name
4	all SQL estimate info, Solid selected key info
5	Solid info also from discarded keys
6	Solid table level info
7	SQL info from every fetched row
8	Solid info from every fetched row

The SQL Info facility is turned on by setting a non-zero value to the `Info` parameter in the `[SQL]` section of the configuration file. The output is written to a file named `sol-trace.out` in the SOLID directory.

Example:

```
[SQL]
info = 1
```

The SQL Info facility can also be turned on with the following SQL statement (this sets SQL Info on only for the client that executes the statement):

```
SET SQL INFO ON LEVEL info-value FILE file-name
```

and turned off with the following SQL statement:

```
SET SQL INFO OFF
```

Example:

```
SET SQL INFO ON LEVEL 1 FILE 'my_query.txt'
```

The EXPLAIN PLAN Statement

The syntax of the EXPLAIN PLAN statement is:

```
EXPLAIN PLAN FOR sql-statement
```

The EXPLAIN PLAN statement is used to show the execution plan that the SQL optimizer has selected for a given SQL statement. An execution plan is a series of primitive operations, and an ordering of these operations, that SOLID *Embedded Engine* performs to execute the statement. Each operation in the execution plan is called a unit.

Unit	Description
JOIN UNIT	Join unit joins two or more tables. The join can be done by using loop join or merge join. Note that the join unit is generated also for queries that reference only a single table. In that case no join is executed in the join unit, the join unit just passes the rows without manipulating them.
TABLE UNIT	Table unit is used to fetch the data rows from a table. Table unit is always the last unit in the chain, since it is responsible for fetching the actual data from the index or table.
ORDER UNIT	Order unit is used to order rows for grouping or to satisfy ORDER BY. The ordering can be done in memory or using an external disk sorter.
GROUP UNIT	Group unit is used to do grouping and aggregate calculation.

Explain Plan Table Columns

The table returned by the EXPLAIN PLAN statement contains the following columns.

Column name	Description
ID	The output row number, used only to guarantee that the rows are unique.

UNIT_ID	This is the internal unit id in the SQL interpreter. Each unit has a different id. The unit id is a sparse sequence of numbers, because the SQL interpreter generates unit ids also for those units that are removed during the optimization phase. If more than one row has the same unit id it means that those rows belong to the same unit. For formatting reasons the info from one unit may be divided into several different rows.
PAR_ID	Parent unit id for the unit. The parent id number refers to the id in the UNIT_ID column.
JOIN_PATH	For join unit there is a join path which specifies which tables are joined in the join unit and the join order for tables. The join path number refers to the unit id in the UNIT_ID column. It means that the input to the join unit comes from that unit. The order in which the tables are joined is the order in which the join path is listed. The first listed table is the outermost table in a loop join.
UNIT_TYPE	Unit type is the execution graph unit type.
INFO	Info column gives additional info. It may contain e.g. index usage, the database table name and constraints used in the database engine to select rows. Note that the constraints listed here may not match those constraints given in the SQL statement.

The following texts may exist in the INFO column for different types of units.

Unit type	Text in Info column	Description
TABLE UNIT	<tablename>	The table unit refers to table <tablename>.
TABLE UNIT	<constraints>	The constraints that are passed to the database engine are listed. If for example in joins the constraint value is not known in advance, the constraint value is displayed as NULL.
TABLE UNIT	SCAN TABLE	Full table scan is used to search for rows.

TABLE UNIT	SCAN <indexname>	Index <indexname> is used to search for rows. If all selected columns are found from an index, sometimes it is faster to scan the index instead of the clustering key because the index has fewer disk blocks.
TABLE UNIT	PRIMARY KEY	The primary key is used to search rows. This differs from SCAN in that the whole table is not scanned because there is a limiting constraint to the primary key attributes.
TABLE UNIT	INDEX <indexname>	Index <indexname> is used to search for rows. For every matching index row, the actual data row is fetched separately.
TABLE UNIT	INDEX ONLY <indexname>	Index <indexname> is used to search for rows. All selected columns are found from the index, so the actual data rows are not fetched separately.
JOIN UNIT	MERGE JOIN	Merge join is used to join the tables.
JOIN UNIT	LOOP JOIN	Loop join is used to join the tables.
ORDER UNIT	NO ORDERING REQUIRED	No ordering is required, the rows are retrieved in correct order from the database engine.
ORDER UNIT	EXTERNAL SORT	External sorter is used to sort the rows. To enable external sorter, the temporary directory name must be specified in the Sorter section of the configuration file.

ORDER UNIT	FIELD <n> USED AS PARTIAL ORDER	Internal sorter (in-memory sorter) is used for sorting and the rows retrieved from the database engine are partially sorted with column number <n>. The partial ordering helps the internal sorter to avoid multiple passes over the data.
ORDER UNIT	NO PARTIAL SORT	Internal sorter is used for sorting and the rows are retrieved in random order from the database engine.

Example 1

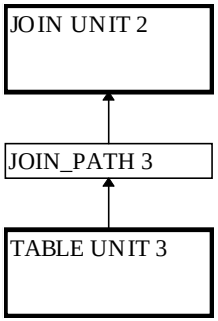
```
EXPLAIN PLAN FOR SELECT * FROM TENKTUP1 WHERE UNIQUE2_NI BETWEEN 0 AND
99;
```

ID	UNIT_ID	PAR_ID	JOIN_ PATH	UNIT_ TYPE	INFO
1	2	1	3	JOIN UNIT	
2	3	2	0	TABLE UNIT	TENKTUP1
3	3	2	0		FULL SCAN
4	3	2	0		UNIQUE2_NI <= 99
5	3	2	0		UNIQUE2_NI >= 0
6	3	2	0		

Execution graph:

JOIN UNIT 2 gets input from TABLE UNIT 3

TABLE UNIT 3 for table TENKTUP1 does a full table scan with constraints UNIQUE2_NI <= 99 and UNIQUE2_NI >= 0



Example 1. Execution graph

Example 2

```
EXPLAIN PLAN FOR SELECT * FROM TENKTUP1, TENKTUP2 WHERE TENKTUP1.UNIQUE2
> 4000 AND TENKTUP1.UNIQUE2 < 4500 AND TENKTUP1.UNIQUE2 =
TENKTUP2.UNIQUE2;
```

ID	UNIT_ID	PAR_ID	JOIN_ PATH	UNIT_ TYPE	INFO
1	6	1	9	JOIN UNIT	MERGE JOIN
2	6	1	10		
3	9	6	0	ORDER UNIT	NO ORDER-ING REQUIRED
4	8	9	0	TABLE UNIT	TENKTUP2
5	8	9	0		PRIMARY KEY
6	8	9	0		UNIQUE2 < 4500
7	8	9	0		UNIQUE2 > 4000
8	8	9	0		
9	10	6	0	ORDER UNIT	NO ORDER-ING REQUIRED
10	7	10	0	TABLE UNIT	TENKTUP1

11	7	10	0		PRIMARY KEY
12	7	10	0		UNIQUE2 < 4500
13	7	10	0		UNIQUE2 > 4000
14	7	10	0		

Execution graph:

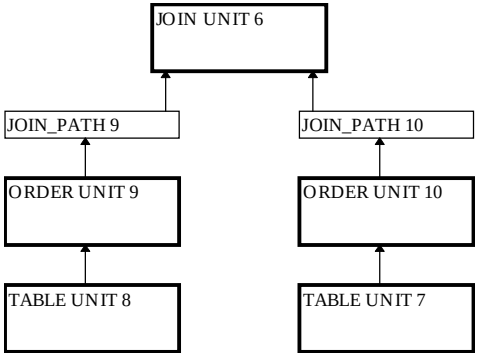
JOIN UNIT 6 the input from order units 9 and 10 are joined using merge join algorithm

ORDER UNIT 9 orders the input from TABLE UNIT 8. Since the data is retrieved in correct order, no real ordering is needed

ORDER UNIT 10 orders the input from TABLE UNIT 7. Since the data is retrieved in correct order, no real ordering is needed

TABLE UNIT 8: rows are fetched from table TENKTUP2 using primary key. Constraints UNIQUE2 < 4500 and UNIQUE2 > 4000 are used to select the rows

TABLE UNIT 7: rows are fetched from table TENKTUP1 using primary key. Constraints UNIQUE2 < 4500 and UNIQUE2 > 4000 are used to select the rows



Example 2. Execution graph

Tracing Communication between Client and Server

SOLID *Embedded Engine* provides following tools for observing the communication between an application and an embedded engine:

- the Network Trace facility

- the Ping facility

You can use these tools to analyze the functionality of the networking between an application and an embedded engine. The network trace facility should be used when you want to know why a connection is not established to an embedded engine. The ping facility is used to determine how fast packets are transferred between an application and an embedded engine.

The Network Trace Facility

Network tracing can be done on the embedded engine computer, on the application computer or on both computers concurrently. The trace information is written to the default trace file or file specified in the `TraceFile` parameter.

The default name of the output file is SOLTRACE.OUT. This file will be written to the current working directory of the server or client depending on which end the tracing is started.

The file contains information about:

- loaded DLLs
- network addresses
- possible errors

The Network Trace facility is turned on by editing the configuration file

```
[Com]
Trace ={Yes|No}
; default No
TraceFile = file-name
; default soltrace.out
```

or by using the environment variables `SOLTRACE` and `SOLTRACEFILE` to override the definitions in the configuration file. Setting of `SOLTRACE` and `SOLTRACEFILE` environment variables have the same effect as the parameters `Trace` and `TraceFile` in the configuration file.

NOTE: Defining the `TraceFile` configuration parameter or the `SOLTRACEFILE` environment variable automatically turns on the Network trace facility.

A third alternative to turn on the Network trace facility is to use the option -t and/or -o*filename* as a part of the network name. The option -t turns on the Network trace facility. The option -o turns on the facility and defines the name of the trace output file.

Example 1. Defining Parameter Trace in the Configuration File

```
[Com]
Connect = nmp SOLID
Listen = nmp SOLID
Trace = Yes
```

Example 2. Defining Environment Variables

```
set SOLTRACE = Yes

or

set SOLTRACEFILE = trace.out
```

Example 3. Using Network Name Options

```
[Com]
Connect  = nmp -t solid
Listen = nmp -t solid

or

[Com]
Connect = nmp -oclient.out solid
Listen = nmp -oserver.out solid
```

The Ping Facility

The Ping facility can be used to test the performance and functionality of the networking. The Ping facility is built in all SOLID client applications and is turned on with the network name option -p*level*.

The output file will be written to the current working directory of the computer where the parameter is given. The default name of the output file is SOLTRACE.OUT.

Clients can always use the Ping facility at level 1. Levels 2, 3, 4 or 5 may only be used if the server is set to use the Ping facility at least at the same level.

The Ping facility levels are:

Setting	Function	Description
0	no operation	do nothing, default

1	check that server is alive	exchange one 100 byte message
2	basic functional test	exchange messages of sizes 0.1K, 1K, 2K..30K, increment 1K
3	basic speed test	exchange 100 messages of sizes 0.1K, 1K, 8K and display each sub-result and total time
4	heavy speed test	exchange 100 messages of sizes 0.1K, 1K, 2K, 4K, 8K, 16K and display each sub-result and total time
5	heavy functional test	exchange messages of sizes 1..30K, increment 1 byte

Example 1

The client turns on the Ping facility by using the following network name:

```
nmp -p1 -oping.out SOLID
```

This runs the Ping facility at the level 1 into a file named SOLTRACE.OUT. This test checks if the server is alive and exchanges one 100 byte message to the server.

After the Ping facility has been run, the client exits with the following message:

```
SOLID Communication return code xxx: Ping test successful/failed,
results are in file FFF.XX
```

Example 2

If the server is using the following listen parameter

```
[Com]
Listen = nmp -p3 SOLID
```

clients can run the Ping facility at levels 1, 2 and 3, but not 4 and 5.

NOTE. Ping clients running at level greater than 3 may cause heavy network traffic and may cause slowness of application using the network. They will also slow down ordinary SQL clients connected to the same SOLID *Embedded Engine*.

Problem Reporting

SOLID *Embedded Engine* offers sophisticated diagnostic tools and methods for producing high quality problem reports with very limited effort. Use the diagnostic tools to capture all the relevant information about the problem.

All problem reports should contain the following files and information:

- solid.ini
- license number
- solmsg.out
- solerror.out
- soltrace.out
- problem description
- steps to reproduce the problem
- all error messages and codes
- contact information, preferably email address of the contact person

Problem Categories

Most problems can be divided into the following categories:

- SOLID *SQL API*
- SOLID *ODBC Driver*
- UNIFACE driver for SOLID *Embedded Engine*
- Communication problems between the application and SOLID *Embedded Engine*

The following pages include a detailed instructions to produce proper problem report for each problem type. Please follow the guidelines carefully.

SOLID SQL API Problems

If the problem concerns the performance of SOLID *SQL API* or a specific SQL statement, you should run SQL info facility at level 4 and include the generated `soltrace.out` file into your problem report. This file contains the following information:

- create table statements
- create view statements

- create index statements
- SQL statement(s)

SOLID ODBC Driver Problems

If the problem concerns the performance of SOLID ODBC Driver, please include the following information:

- SOLID *ODBC Driver* name, version, and size
- ODBC Driver Manager version and size

If the problem concerns the cooperation of SOLID *Embedded Engine* and any third party standard software package, please include the following information:

- Full name of the software
- Version and language
- Manufacturer
- Error messages from the third party software package

Use ODBC trace option to get a log of the ODBC statements and include it to your problem report.

SOLID JDBC Driver Problems

If the problem is related to the SOLID *JDBC Driver*, please include the following information into your problem report:

- Exact version of JDK used
- Size and date of the SOLIDDriver class package
- Contents of DriverManager.setLogStream(someOutputStream) output, if available
- Call stack (that is, Exception.printStackTract() output) of the application, if an Exception has occurred n the application

UNIFACE Driver for SOLID *Embedded Engine* Problems

If the problem concerns the performance of SOLID UNIFACE Driver, please include following information:

- SOLID *UNIFACE Driver* version and size
- UNIFACE version and platform

- Contents of the UNIFACE message frame
- Error codes from the driver, $STATUS, $ERROR
- All necessary files to reproduce the problem (TRXs, SQL scripts, USYS.ASN etc.)

Communication between a Client and Server

If the problem concerns the performance of the communication between a client and server use the Network trace facility and include the generated trace files into your problem report. Please include the following information:

- SOLID communication DLLs used: version and size
- other communication DLLs used: version and size
- description of the network configuration

A

Error Codes

Error Categories

SQL Errors

These errors are caused by erroneous SQL statements and are detected by the SOLID SQL Parser. Administrative actions are not needed.

Database Errors

These errors are detected by the SOLID *Embedded Engine* and may demand administrative actions.

System Errors

These errors are detected by the operating system and demand administrative actions.

Table Errors

These errors are caused by erroneous SQL statements and detected by the SOLID *Embedded Engine*. Administrative actions are not needed.

Server Errors

These errors are caused by erroneous administrative actions or client requests. They may demand administrative actions.

Communication Errors

These errors are caused by network errors or faulty configuration of the SOLID *Embedded Engine* software. These errors demand administrative actions.

Procedure Errors

These errors are caused by errors in the definition or execution of a stored procedure. Administrative actions are not needed.

See also:

See *Appendix C, "Data Types"* and *Appendix D, "SOLID SQL Syntax"* for more information.

SOLID SQL Errors

Error code	Description
SQL Error 1	Parsing error ‘syntax error’
	The SQL parser could not parse the SQL string. Check the syntax of the SQL statement and try again.
SQL Error 2	Table <table> can not be opened
	You may not have privileges to access the table and its data.
SQL Error 3	Table <table> can not be created
	Table can not be created. You may not have privileges for this operation.
SQL Error 4	Illegal type definition <column>
	A column type in your CREATE TABLE statement is illegal. Use a legal type for the column.
SQL Error 5	Table <table> can not be dropped
	Table can not be dropped. Only the owner (i.e. the creator) can drop it.
SQL Error 6	Illegal value specified for column <column>
	The value specified for column is invalid. Check the value for the column.
SQL Error 7	Insert failed
	The server failed to do the insertion. You may not have INSERT privilege on the table or it may be locked.
SQL Error 8	Delete failed
	The server failed to do the deletion. You may not have DELETE privilege on the table or the row may be locked.
SQL Error 9	Row fetch failed
	The server failed to fetch a row. You may not have SELECT privilege on the table or there may be an exclusive lock on the row.
SQL Error 10	View <view> can not be created

	You cannot create this view. You may not have SELECT privilege on one or more tables in the query-specification of your CREATE VIEW statement.
SQLError11	View <view> cannot be dropped.
	You cannot drop this view. Only the owner (i.e. the creator) of the view can drop it.
SQLError12	Illegal view definition <view>
	The view definition is illegal. Check the syntax of the definition.
SQLError13	Illegal column name <column>
	Column name is illegal. Check that the name is not a reserved name.
SQL Error 14	Call to function <function> failed
	Function call to function failed. Check the arguments and their types.
SQL Error 15	Arithmetics error
	An arithmetics error occurred. Check the operators, values and types.
SQL Error 16	Update failed
	The server failed to update a row. There may a lock on a row.
SQL Error 17	View is not updatable
	This view is not updatable. UPDATE, INSERT and DELETE operations are not allowed.
SQL Error 18	Inserted row does not meet check option condition
	You tried to insert a row, but one or more of the column values do not meet column constraint definition.
SQL Error 19	Updated row does not meet check option condition
	You tried to update a row, but one or more of the column values do not meet column constraint definition.
SQL Error 20	Illegal CHECK constraint

	A check constraint given to the table is illegal. Check the types of the check constraint of this table.
SQL Error 21	Insert failed because of CHECK constraint
	You tried to insert a row, but the values do not meet the check option conditions.
SQL Error 22	Update failed because of CHECK constraint
	You tried to update a row, but the values do not meet the check option conditions.
SQL Error 23	Illegal DEFAULT value
	The DEFAULT value for the column given is illegal.
SQL Error 25	Duplicate columns in INSERT column list
	You have included a column in column list twice. Remove duplicate columns.
SQL Error 26	At least one column definition required in CREATE TABLE
	You need to specify at least one column definition in a CREATE TABLE statement.
SQL Error 27	Illegal REFERENCES column list
	There are wrong number of columns in your REFERENCES list.
SQL Error 28	Only one PRIMARY KEY allowed in CREATE TABLE
	You can use only one PRIMARY KEY in CREATE TABLE.
SQL Error 29	GRANT failed
	Granting privileges failed. You may not have privileges for this operation.
SQL Error 30	REVOKE failed
	Revoking privileges failed. You may not have privileges for this operation.
SQL Error 31	Multiple instances of a privilege type
	You tried to grant privileges to a role or a user. You have included multiple instances of a privilege type in the list of privileges.

SQL Error 32	Illegal constant <constant>
	Illegal constant was found. Check the syntax of the statement.
SQL Error 33	Column name list of illegal length
	You have entered different number of columns in CREATE VIEW statement to the view and to the table.
SQL Error 34	Conversion between types failed
	An expression in UPDATE statement has illegal type for a column.
SQL Error 35	Column names not allowed in ORDER BY for UNION
	You can not use column name in an ORDER BY for UNION statement.
SQL Error 36	Nested aggregate functions
	Nested aggregate functions can not be used. For example: SUM(AVG(<column>)).
SQL Error 37	Aggregate function with no arguments
	An aggregate function was entered with no arguments. For example: SUM().
SQL Error 38	Set operation between different row types
	You have tried to execute a set operation of tables with incompatible row types. The row types in a set operation must be compatible.
SQL Error 39	COMMIT WORK failed
	Committing a transaction failed.
SQL Error 40	ROLLBACK WORK failed
	Rolling back a transaction failed.
SQL Error 41	Savepoint could not be created
	A savepoint could not be created.
SQL Error 42	Could not create index <index>
	An index could not be created. You may not have privileges for this operation. You need to be an owner of the table or have SYS_ADMIN_ROLE to have privileges to create index for the table.

SQL Error 43	Could not drop index <index>
	An index could not be dropped. You may not have privileges for this operation. You need to be an owner of the table or have SYS_ADMIN_ROLE to have privileges to drop index from the table.
SQL Error 44	Could not create schema <schema>
	A schema could not be created.
SQL Error 45	Could not drop schema <schema>
	A schema could not be dropped.
	You tried to use an ORDER BY column that does not exist. Refer to an existing column in the ORDER BY specification.
SQL Error 47	Maximum length of identifier is 31
	You have exceeded the maximum length for the identifier.
SQL Error 48	Subquery returns more than one row
	You have used a subquery that returns more than one row. Only subqueries returning one row may be used in this situation.
SQL Error 49	Illegal expression <expression>
	You tried to insert or update a table using an aggregate function (SUM, MAX, MIN or AVG) as a value. This is not allowed.
SQL Error 50	Ambiguous column name <column>
	You have referenced a column which is exists in more than one table. Use syntax <table>.<column> to indicate which table you want to use.
SQL Error 51	Non-existent function <function>
	You tried to use a function which does not exist.
SQL Error 52	Non-existent cursor <cursor>
	You tried to use a cursor which is not created.
SQL Error 53	Function call sequence error
	A function was called in wrong order. Check the sequence and success of the function calls.

SQL Error 54	Illegal use of a parameter
	A parameter was used illegally. For example: SELECT * FROM TEST WHERE ? < ?;
SQL Error 55	Illegal parameter value
	A parameter has an illegal value. Check the type and value of the parameter.
SQL Error 56	Only ANDs and simple condition predicates allowed in UPDATE CHECK
	All search condition predicates are not supported.
SQL Error 57	Opening the cursor did not succeed
	Server failed to open a cursor. You may not have cursor open at this moment.
SQL Error 58	Column <column> is not referenced in group-by-clause
	You tried to group rows using <column>. All columns in group-by-clause must be listed in your select-list. A star (‘*’) notation is not allowed with GROUP BY.
SQL Error 59	Comparison between incompatible types
	You tried to compare values which have incompatible types. Incompatible types are for example an integer and a date value.
SQL Error 60	Reference to the insert table not allowed in the source query
	You have referenced in subquery a table where you are inserting values. This is not allowed.
SQL Error 61	Reference to the update table not allowed in subquery
	You have referenced in subquery a table where you are updating values. This is not allowed.
SQL Error 62	Reference to the delete table not allowed in subquery
	You have referenced in subquery a table where you are deleting values. This is not allowed.
SQL Error 63	Subquery returns more than one column
	You have used a subquery that returns more than one column. Only subqueries returning one column may be used.

SQL Error 64	Cursor <cursor> not updatable The cursor opened is not updatable.
SQL Error 65	Insert or update tried on pseudo column You tried to update a pseudo column (ROWID, ROWVER). Pseudo columns are not updatable.
SQL Error 66	Could not create user <user> A user could not be created. You may not have privileges for this operation.
SQL Error 67	Could not alter user <user > A user could not be altered. You may not have privileges for this operation.
SQL Error 68	Could not drop user <user > A user could not be dropped. You may not have privileges for this operation.
SQL Error 69	Could not create role <role> A role could not be created. You may not have privileges for this operation.
SQL Error 70	Could not drop role <role> A role could not be dropped. You may not have privileges for this operation.
SQL Error 71	Grant <role> failed Granting role failed. You may not have privileges for this operation.
SQL Error 72	Revoke <role> failed Revoking role failed. You may not have privileges for this operation.
SQL Error 73	Comparison of vectors of different length You have tried to compare row value constructors that have different number of dimensions. For example you have compared (a,b,c) to (1,1).
SQL Error 74	Expression * not compatible with aggregate expression

	The aggregate expression can not be used with * columns. Specify columns using their names when used with this aggregate expression. This usually happens when GROUP BY expression is used with the * columns.
SQL Error 75	Illegal reference to table <table>
	You have tried to reference a table which is not in the FROM list. For example: SELECT T1.* FROM T2.
SQL Error 76	Ambiguous table name <table>
	You have used the syntax <table>.<column name>ambiguously. For example: SELECT T1.* FROM T1 A,T1 B WHERE A.F1=0;
SQL Error 77	Illegal use of aggregate expression
	You tried to use aggregate expression illegally. For example: SELECT ID FROM TEST WHERE SUM(ID) = 3;
SQL Error 78	Row fetch failed
	The server failed to fetch a row. You may not have SELECT privilege on the table or there may be an exclusive lock on the row.
SQL Error 79	Subqueries not allowed in CHECK constraint
	You tried to use subquery in a check constraint.
SQL Error 80	Sorting failed
	External sorter is out of disk space or cache memory. Modify parameters in configuration file `solid.ini`.
SQL Error 81	SET syntax results in error
SQL Error 82	Improper type used with LIKE
SQL Error 83	Syntax error
SQL Error 84	Parser error <statement>
SQL Error 85	Incorrect number of values for INSERT

SQL Error 86	Illegal ROWNUM constraint

SOLID Database Errors

Error code	Description
Database Error 10001	Key value is not found Internal error: a key value can not be found from the database index.
Database Error 10002	Operation failed This is an internal error indicating that the index of the table accessed is in inconsistent state. Try to drop and create the index again to recover the error.
Database Error 10004	Redefinition Unexpected failure occurred in the database engine. This error may also occur during recovery: either an index or a view has been redefined during recovery. The server is not able to do the recovery. Delete log files and start the server again.
Database Error 10005	Unique constraint violation You have violated a unique constraint. This happens when you have tried to insert or update a column which has a unique constraint and the value inserted or updated is not unique. This may also occur when you create users, tables or roles having same names in separate transactions.
Database Error 10006	Concurrency conflict, two transactions updated or deleted the same row Two separate transactions have modified a same row in the database simultaneously. This has resulted in a concurrency conflict.
Database Error 10007	Transaction is not serializable The transaction committed is not serialisable.
Database Error 10010	No checkpoint in database This error occurs when the server has crashed in the middle of creating a new database. Delete the database and log files and try to create the database again.

Database Error 10011	Database headers are corrupted The headers in the database are corrupted. This may be caused by a disk error or other system failure. Restore the database from the backup.
Database Error 10012	Node split failed This is an internal error.
Database Error 10013	Transaction is read-only You have tried to write inside a transaction that is set read-only. Remove the write operation or unset the read-only mode in the transaction.
Database Error 10014	Resource is locked This error occurs when you are trying to use a key value in an index which has been concurrently dropped.
Database Error 10016	Log file is corrupted One of the log files of the database is corrupted. You can not use these log files. Delete them and start the server again.
Database Error 10017	Too long key value The maximum length of the key value has been exceeded. The maximum value is one third of the size of the index leaf.
Database Error 10019	Backup is active You have tried to start a backup when a backup process is already in progress.
Database Error 10020	Checkpoint creation is active You have tried to start a checkpoint when a checkpoint creation is already in progress.
Database Error 10021	Failed to delete log file The deletion of a log file in making a backup has failed. Reasons for the failure can be: The log file has already been deleted from the operating system. The log file has a read-only attribute.

Database Error 10023	Wrong log file, maybe the log file is from another database The log file in the database directory is from another SOLID Embedded Engine database. Copy the correct log files to the database directory.
Database Error 10024	Illegal backup directory The backup directory is either an empty string or a dot indicating that the backup will be created in the current directory.
Database Error 10026	Transaction is timed out An idle transaction has exceeded the maximum idle transaction time. The transaction has been aborted. The maximum value is set in parameter AbortTimeOut in SRV section. The default value is 120 minutes.
Database Error 10027	No active search Internal error.
Database Error 10028	Referential integrity violation, foreign key values exist You tried to delete a row that is referenced from a foreign key.
Database Error 10029	Referential integrity violation, referenced column values do not exist The definition of a foreign key does not uniquely identify a row in the referenced table.
Database Error 10030	Backup directory 'directory name' does not exist Backup directory is not found. Check the name of the backup directory.
Database Error 10031	Transaction detected a deadlock, transaction is rolled back Deadlock detected. If necessary, begin transaction again.
Database Error 10032	Wrong database block size specified The block size of the database file differs from the block-size given in the configuration file `solid.ini`.
Database Error 10033	Primary key unique constraint violation

	Your primary key definition is not unique.
Database Error 10034	Sequence name <sequence> conflicts with an existing entity
	Choose a unique name for a sequence. The specified name is already used.
Database Error 10035	Sequence does not exist
	Check the name of the sequence.
Database Error 10036	Data dictionary operation is active for accessed sequence
	Create or drop operation is active for the accessed sequence. Try again.
Database Error 10037	Can not store sequence value, the target data type is illegal
	The valid target data types are INTEGER and BINARY.
Database Error 10038	Illegal column value for descending index
	Corrupted data found in descending index. Drop the index and create it again.
Database Error 10040	Log file write failure, probably the disk containing the log files is full
	Shut down the server and reserve more disk space for log files.
Database Error 10041	Database is read-only
Database Error 10042	Database index check failed, the database file is corrupted
Database Error 10043	Database free block list corrupted, same block twice in free list
Database Error 10044	Primary key can not contain blob attributes
Database Error 10046	Operation failed, data dictionary operation is active
Database Error 10047	Replicated transaction is aborted

Database Error 10048	Replicated transaction contains schema changes, operation failed
Database Error 10049	Slave server not available any more, transaction aborted
Database Error 10050	Replicated row contains BLOb columns that cannot be replicated

SOLID Utility Errors

Error code	Description
System Error 11000	File open failure The server is unable to open the database file. Reason for the failure can be: The database file has been set read-only. You do not have rights to open the database file in write mode. Another SOLID Embedded Engine is using the database file. Correct the error and try again.
System Error 11001	File write failure Server is unable to write to the disk. The database files may have a read-only attribute set or you may not have rights to write to the disk. Add rights or unset read-only attribute and try again.
System Error 11002	File write failed, disk full Server failed to write to the disk, because the disk is full. Free disk space or move the database file to another disk. You can also split the database file to several disks using the FileSpec_[1-N] parameter in IndexFile section.
System Error 11003	File write failed, configuration exceeded Writing to the database file failed, because the maximum database file size set in FileSpec_[1-N] parameter is exceeded.
System Error 11004	File read failure An error occurred reading a file. This may indicate a disk error in your system.
System Error 11005	File read beyond end of file Internal error.
System Error 11006	File read failed, illegal file address

	An error occurred reading a file. This may indicate a disk error in your system.
System Error 11007	File lock failure
	The server failed to lock the database file. This error occurs in the Windows version, if you do not have SHARE.EXE loaded. To correct the failure: 1. Exit Windows 2. Load SHARE.EXE 3. Delete the database file SOLID.DB and log files. 4. Start Windows and launch SOLID Embedded Engine.
System Error 11008	File unlock failure
	Server failed to unlock a file.
System Error 11009	File free block list corrupted
	Internal error.
System Error 11010	Too long file name
	Filename specified in parameter FileSpec_[1-N] is too long. Change the name to a proper file name.
System Error 11011	Duplicate file name specification
	Filename specified in parameter FileSpec_[1-N] is not unique. Change the name to a proper file name.
System Error 11012	License information not found, exiting from SOLID Embedded Engine
	Check the existence of your `solid.lic` file.
System Error 11013	License information is corrupted
	Your `solid.lic` file has been corrupted.
System Error 11014	Database age limit of evaluation license expired
System Error 11015	Evaluation license expired
System Error 11016	License is for different CPU architecture

System Error 11017	License is for different OS environment
System Error 11018	License is for different version of this OS
System Error 11019	License is not valid for this server version
System Error 11020	License information is corrupted
System Error 11021	Problem with Your license, please contact SOLID Information Technology Ltd. immediately
System Error 11022	Desktop license is only for local <protocol> communication, cannot use protocol <protocol> for listening
System Error 11024	Desktop license is only for local communication, cannot use name <name> for listening

SOLID Table Errors

Error code	Description
Table Error 13001	Illegal character constant <constant> An illegal character constant was found in the SQL statement.
Table Error 13002	Type CHAR not allowed for arithmetics You have entered a calculation having a character type constant. Character constants are not supported in arithmetics.
Table Error 13003	Aggregate function <function> not available for ordinary call Aggregate functions can not be used for ordinary function calls.
Table Error 13004	Illegal aggregate function <parameter> parameter An illegal parameter has been given to an aggregate function. Aggregate function parameters can only be column names or numbers.
Table Error 13005	SUM and AVG not supported for CHAR type Aggregate functions SUM and AVG are not supported for character type parameters.
Table Error 13006	SUM or AVG not supported for DATE type Aggregate functions SUM and AVG are not supported for date type parameters.
Table Error 13007	Function <function> is not defined The function you tried to use is not defined.
Table Error 13009	Division by zero A division by zero has occurred.
Table Error 13011	Table <table> does not exist You have referenced a table which does not exist or you do not have REFERENCES privilege on the table.
Table Error 13013	Table name <table> conflicts with an existing entity

	Choose a unique name for a table. The specified name is already used.
Table Error 13014	Index <index> does not exist
	You have referenced an index which does not exist.
Table Error 13015	Column <column> does not exist on table <table>
	You have referenced a column in a table which does not exist.
Table Error 13016	User does not exist
	You have referenced a user which does not exist.
Table Error 13018	Join table is not supported
	Joined tables are not supported in this version of SOLID Embedded Engine.
Table Error 13019	Transaction savepoints are not supported
	Transaction savepoints are not supported in this version of SOLID Embedded Engine.
Table Error 13020	Default values are not supported
	Default column values are not supported in this version of SOLID Embedded Engine.
Table Error 13021	Foreign keys are not supported
	Foreign keys are not supported in this version of SOLID Embedded Engine.
Table Error 13022	Descending keys are not supported
	Descending keys are not supported in this version of SOLID Embedded Engine.
Table Error 13023	Schema is not supported
	Schema is not supported in this version of SOLID Embedded Engine.
Table Error 13025	Update through a cursor with no current row
	You have tried to update using cursor, but you do not have current row in the cursor.
Table Error 13026	Delete through a cursor with no current row

	You have tried to delete using cursor, but you do not have current row in the cursor.
Table Error 13028	View <view> does not exist
	You have referenced a view which does not exist.
Table Error 13029	View name <view> conflicts with an existing entity
	Choose a unique name for a view. The specified name is already used.
Table Error 13030	No value specified for NOT NULL column <column>
	You have not specified a value for a column which is defined NOT NULL.
Table Error 13031	Data dictionary operation is active for accessed table or key
	You can not access the table or key, because a data dictionary operation is currently active. Try again after the data dictionary operation has completed.
Table Error 13032	Illegal type <type>
	You have tried to create a table with a column having an illegal type.
Table Error 13033	Illegal parameter <parameter> for type <type>
	The type of the parameter you entered is illegal in this column.
Table Error 13034	Illegal constant <constant>
	You have entered an illegal constant.
Table Error 13035	Illegal INTEGER constant <constant>
	You have entered an illegal integer type constant. Check the syntax of the statement and try again.
Table Error 13036	Illegal DECIMAL constant <constant>
	You have entered an illegal decimal type constant. Check the decimal number and try again.
Table Error 13037	Illegal DOUBLE PREC constant <constant>
	You have entered an illegal double precision type constant. Check the number and try again.
Table Error 13038	Illegal REAL constant <constant>

	You have entered an illegal real type constant. Check the real number and try again.
Table Error 13039	Illegal assignment
	You have tried to assign an illegal value for a column.
Table Error 13040	Aggregate <function> function is not defined
	The aggregate function you tried to use is not supported.
Table Error 13041	Type DATE not allowed for arithmetics
	DATE type columns or constants are not allowed in arithmetics.
Table Error 13042	Power arithmetic not allowed for NUMERIC and DECIMAL data type
	Decimal and numeric data types do not support power arithmetics.
Table Error 13043	Illegal date constant <constant>
	A date constant is illegal. The correct form for date constants is: YYYY-MM-DD.
Table Error 13045	Reference privileges are not supported
	Reference privileges are not supported in this version of SOLID Embedded Engine.
Table Error 13046	Illegal user name <user>
	User name entered is not legal. A legal user name is at least 2 and at most 31 characters in length. A user name may contain characters from A to Z, numbers from 0 to 9 and underscore character '_'.
Table Error 13047	No privileges for operation
	You have no privileges for the attempted operation.
Table Error 13048	No privileges to grant privileges for table <table>
	You have no privileges to grant privileges for the table.
Table Error 13049	Column privileges cannot be granted WITH GRANT OPTION
	Granting column privileges WITH GRANT OPTION is not supported in this version of SOLID Embedded Engine.

Table Error 13050	Too long constraint value Maximum constraint length has been exceeded. Maximum constraint length is 255 characters.
Table Error 13051	Illegal column name <column> You have tried to create a table with an illegal column name.
Table Error 13052	Illegal comparison operator <operator> for a pseudo column <column> You have tried to use an illegal comparison operator for a pseudo column. Legal comparison operators for pseudo columns are: equality '=' and non-equality '<>'.
Table Error 13053	Illegal data type for a pseudo column You have tried to use an illegal data type for a pseudo column. Data type of pseudo columns is BINARY.
Table Error 13054	Illegal pseudo column data, maybe data is not received using pseudo column You have tried to compare pseudo column data with non-pseudo column data. Pseudo column data can only be compared with data received from a pseudo column.
Table Error 13055	Update not allowed on pseudo column Updates are not allowed on pseudo columns.
Table Error 13056	Insert not allowed on pseudo column Inserts are not allowed on pseudo columns.
Table Error 13057	Index name <index> already exists You have tried to create an index, but an index with the same name already exists. Use another name for the index.
Table Error 13058	Constraint checks were not satisfied on column <column> Column has constraint checks which were not satisfied during an insert or update.
Table Error 13059	Reserved system name <name> You tried to use a name which is a reserved system name such as PUBLIC and SYS_ADMIN_ROLE.

Table Error 13060	User name <user> not found You tried to reference a user name which is not created.
Table Error 13061	Role name <role> not found You tried to reference a role name which is not created.
Table Error 13062	Admin option is not supported Admin option is not supported in this version of SOLID Embedded Engine.
Table Error 13063	Name <name> already exists You tried to use a role or user which already exists. User names and role names must all be different i.e. you can not have a user named HOBBES and a role named HOBBES.
Table Error 13064	Not a valid user name <user> You tried to create an invalid user name. A valid user name has at least 2 characters and at most 31 characters.
Table Error 13065	Not a valid role name <role> You tried to create an invalid role name. A valid user name has at least 2 characters and at most 31 characters.
Table Error 13066	User <user> not found in role <role> You tried to revoke a role from a user and the user did not have that role.
Table Error 13067	Too short password You have entered a too short password. Password length must be at least 3 characters.
Table Error 13068	Shutdown is in progress You are unable to complete this operation, because server shutdown is in progress.
Table Error 13070	Numerical overflow A numerical overflow has occurred. Check the values and types of numerical variables.
Table Error 13071	Numerical underflow A numerical underflow has occurred. Check the values and types of numerical variables.

Table Error 13072	Numerical value out of range A numerical value is out of range. Check the values and types of numerical variables.
Table Error 13073	Math error A mathematical error has occurred. Check the mathematics in the statement and try again.
Table Error 13074	Illegal password You have tried to enter an illegal password.
Table Error 13075	Illegal role name <role> You have tried to enter an illegal role name. A legal role name is at least 2 and at most 31 characters in length. A user role may contain characters from A to Z, numbers from 0 to 9 and underscore character '_'.
Table Error 13076	NOT NULL must not be specified for added column <column> You have tried to add a column to a table using ALTER TABLE statement. NOT NULL constraint is not allowed in ALTER TABLE statement when the table already includes data.
Table Error 13077	Last column can not be dropped You have tried to drop the final column in a table. This is not allowed; at least one column must remain in the table.
Table Error 13078	Column already exist on table You have tried to create a column which already exists in a table.
Table Error 13079	Illegal search constraint Check the search engine. There may be mismatch between data types.
Table Error 13080	Incompatible types, can not modify column <column> from You have tried to modify column to a data type that is incompatible with the original definition, such as VAR-
Table Error 13081	Descending keys are not supported for binary columns

	You can not define descending key for a binary column.
Table Error 13082	Function <function>: parameter * not supported
	You can not use parameter star (*) with ODBC Scalar Functions.
Table Error 13083	Function <function>: Too few parameters
	The function expects more parameters. Check the function call.
Table Error 13084	Function <function>: Too many parameters
	The function expects fewer parameters. Check the function call.
Table Error 13085	Function <function>: Run-time failure
	An error was detected during the execution of the function. Check the parameters.
Table Error 13086	Function <function>: type mismatch in parameter <parameter number>
	A erroneous type of parameter detected in the given position of the function call. Check the function call.
Table Error 13087	Function <function>: illegal value in parameter <parameter number>
	An illegal value for a parameter detected in the given position of the function call. Check the function call.
Table Error 13090	Foreign key column <column> data type not compatible with referenced column data type
	References specification error. Check that the column data type are compatible between referencing and referenced tables.
Table Error 13091	Foreign key does not match to the primary key or unique constraint of the referenced table
	References specification error. Check that the column data type are compatible between referencing and referenced tables and that the foreign key is unique for the referenced table.
Table Error 13092	Event name <event> conflicts with an existing entity

	Choose a unique name for an event. The specified name is already used.
Table Error 13093	Event <event>does not exist
	You referenced to a nonexistent event. Check the name of event.
Table Error 13094	Duplicate column <column> in primary key definition
	Duplicate columns are not allowed in a table-constraint-definition. Remove duplicate columns from the definition.
Table Error 13095	Duplicate column <column> in unique constraint definition
	Duplicate columns are not allowed in a table-constraint-definition. Remove duplicate columns from the definition.
Table Error 13096	Duplicate column <column> in index definition
	Duplicate columns are not allowed in CREATE INDEX statement. Remove duplicate columns.
Table Error 13097	Primary key columns must be NOT NULL
	Error in a column-constraint-definition. Define primary key columns NOT NULL. For example: CREATE TABLE DEPT (DEPTNO INTEGER NOT NULL, DNAME VARCHAR, PRIMARY KEY(DEPTNO));
Table Error 13098	Unique constraint columns must be NOT NULL
	Error in a column-constraint-definition. Define unique columns NOT NULL. For example: CREATE TABLE DEPT4 (DEPTNO INTEGER NOT NULL, DNAME VARCHAR, UNIQUE(DEPTNO));
Table Error 13099	No REFERENCES privileges to referenced columns in table <table>
	You do not have privileges to reference to the table.
Table Error 13100	Illegal table mode combination
	You have defined illegal combination of locking. Check locking type of tables.
Table Error 13101	Only execute privileges can be used with procedures
Table Error 13102	Execute privileges can be used only with procedures
Table Error 13103	Illegal grant or revoke operation

Table Error 13104	Sequence name <sequence> conflicts with an existing entity Choose a unique name for a sequence. The specified name is already used.
Table Error 13105	Sequence <sequence>does not exist You referenced to a nonexistent sequence. Check the name of sequence.
Table Error 13106	Foreign key reference exists to table <table>
Table Error 13107	Illegal set operation You tried to execute a non-existent set operation.
Table Error 13108	Comparison between incompatible types <datatype> and <datatype>
Table Error 13109	There are schema objects for this user, drop failed
Table Error 13110	NULL values given for NOT NULL column <column>
Table Error 13111	Ambiguous entity name <name>
Table Error 13112	Foreign keys are not supported with main memory tables
Table Error 13113	Illegal arithmetics between types <datatype> and <datatype>
Table Error 13114	String operations are not allowed on values stored as BLObs or CLObs

SOLID *Embedded Engine* Errors

Error code	Description
Server Error 14501	Operation failed
	This error occurs when a timed command fails. Check the arguments of timed commands.
Server Error 14502	RPC parameter is invalid
	A network error has occurred.
Server Error 14503	Communication error
	A communication error has occurred.
Server Error 14504	Duplicate cursor name <cursor>
	You have tried to declare a cursor with a cursor name which is already in use. Use another name.
Server Error 14505	Connect failed, illegal user name or password
	You have entered either a user name or a password that is not valid.
Server Error 14506	Server is closed, no new connections allowed
	You have tried to connect to a closed server. Connecting was aborted.
Server Error 14507	Maximum number of licensed user connections exceeded
	You have tried to connect to a server which has all licenses currently in use. Connecting was aborted.
Server Error 14508	The operation has timed out
	You have launched an operation that has been aborted.
Server Error 14509	Version mismatch
	A version mismatch has occurred. The client and server are different versions. Use same versions in the client and the server.
Server Error 14510	Communication write operation failed
	A write operation failed. This indicates a network problem. Check your network settings.
Server Error 14511	Communication read operation failed

	A read operation failed. This indicates a network problem. Check your network settings.
Server Error 14512	There are users logged to the server
	You can not shutdown the server now. There are users connected to the server.
Server Error 14513	Backup process is active
	You can not shutdown the server now. The backup process is active
Server Error 14514	Checkpoint creation is active
	You can not shutdown the server now. The checkpoint creation is active.
Server Error 14515	Invalid user id
	You tried to drop a user, but the user id is not logged in to the server.
Server Error 14516	Invalid user name
	You tried to drop a user, but the user name is not logged in to the server.
Server Error 14517	Someone has updated the at commands at the same time, changes not saved
	You tried to update timed commands at the same time another user was doing the same. Your changes will not be saved.
Server Error 14518	Connection to the server is broken, connection lost
	Possible network error. Reconnect to the server.
Server Error 14519	The user was thrown out from the server, connection lost
	Possible network error.
Server Error 14521	Failed to create a new thread for the client
Server Error 14529	The operation timed out

SOLID Communication Errors

Error code	Description
Communication Error 21300	Protocol <protocol> is not supported
	Protocol is not supported.
Communication Error 21301	Cannot load the dynamic link library <library> or one of its components
	The server was unable to load the dynamic link library or a component needed by this library. Check the existence of necessary libraries and components.
Communication Error 21302	Wrong version of dynamic link library <library>
	The version of this library is wrong. Update this library to a newer version.
Communication Error 21303	Network adapter card is missing or needed <protocol> software is not running
	The network adapter card is missing or not functioning.
Communication Error 21304	Out of <protocol> resources
	The network protocol is out of resources. Increase the protocols resources in the operating system.
Communication Error 21305	An empty or incomplete network name was specified
	The network name specified is not legal. Check the network name.
Communication Error 21306	Server <network name> not found, connection failed
	The server was not found. 1) Check that the server is running. 2) Check that the network name is valid. 3) Check that the server is listening given network name.
Communication Error 21307	Invalid connect info <network name>
	The network name given as the connect info is not legal. Check the network name.
Communication Error 21308	Connection is broken (<protocol> <read/write> operation failed with code <internal code>)

	The connection using the protocol is broken. Either a read or a write operation has failed with and internal error code <internal code>.
Communication Error 21309	Failed to accept a new client connection, out of <protocol> resources
	The server was not able to establish a new client connection. The protocol is out of resources. Increase the protocol's resources in the operating system.
Communication Error 21310	Failed to accept a new client connection, listening of <network name> interrupted
	The server was not able to establish a new client connection. The listening has been interrupted.
Communication Error 21311	Failed to start a selecting thread for <network name>
	A thread selection has failed for <network name>.
Communication Error 21312	Listening info <network name> already specified for this server
	A network name has already been specified for this server. A server can not use a same network name more than once.
Communication Error 21313	Already listening with the network name <network name>
	You have tried to add a network name to a server when it is already listening with that network name. A server can not use a same network name more than once.
Communication Error 21314	Cannot start listening, network name <network name> is used by another process
	The server can not start listening with the given network name. Another process in this computer is using the same network name.
Communication Error 21315	Cannot start listening, invalid listening info <network name>
	The server can not start listening with the given listening info. The given network name is invalid. Check the syntax of the network name.
Communication Error 21316	Cannot stop the listening of <network name>. There are clients connected

	You can not stop listening of this network name. There are clients connected to this server using this network name.
Communication Error 21317	Failed to save the listen information into the configuration file
	The server failed to save this listening information to the configuration file. Check the file access rights and format of the configuration file.
Communication Error 21318	Operation failed because of an unusual <protocol> return code <code>
	Possible network error. Create connection again.
Communication Error 21319	RPC request contained an illegal version number
	Either the message was corrupted or there may be a mismatch between server and client versions.
Communication Error 21320	Called RPC service is not supported in the server
	There maybe a mismatch between server and client versions.
Communication Error 21321	Protocol %s is not valid, try using switch '-a' for specifying another adapter id instead of %d
	This is returned if the NetBIOS LAN adapter id given in listen/connect string is not valid.
Communication Error 21322	The host machine given in connect info '%s' was not found
	This is returned in clients if the host machine name given in connect info is not valid.
Communication Error 21323	Protocol <protocol> can not be used for listening in this environment.
	This message is displayed if the server end communication using specified protocol is not supported.
Communication Error 21324	The process does not have the privilege to create a mailbox

SOLID Communication Warnings

Error code	Description
Warning Code 21100	Illegal value <value> for configuration parameter <parameter>, using default
	An illegal value was given to the parameter <parameter>. The server will use a default value for this parameter.
Warning Code 21101	Invalid protocol definition <protocol> in configuration file
	The protocol is defined illegally in the configuration file. Check the syntax of the definition.

SOLID Procedure Errors

Error code	Description
Procedure Error 23001	Undefined symbol <symbol> You have used a symbol that has not been defined in a procedure definition.
Procedure Error 23002	Undefined cursor <cursor> You have used a cursor that has not been defined in a procedure definition.
Procedure Error 23003	Illegal SQL operation <operation>
Procedure Error 23004	Syntax error: parse error, line <line number> Check the syntax of your procedure.
Procedure Error 23005	Procedure <procedure> not found
Procedure Error 23006	Wrong number of parameters for procedure <procedure>
Procedure Error 23007	Procedure name <value> conflicts with an existing entity. Choose a unique name for a procedure. The specified name is already used.
Procedure Error 23009	Event <event> does not exist, line <line number>
Procedure Error 23010	Incompatible event <event> parameter type, line <line number>
Procedure Error 23011	Wrong number of parameter for event <event>, line <line number>
Procedure Error 23012	Duplicate wait for event <event>, line <line number>

Procedure Error 23013	Undefined sequence <sequence>
Procedure Error 23014	Duplicate sequence name <sequence>
Procedure Error 23015	Sequence <sequence> not found
Procedure Error 23016	Incompatible variable type in call to sequence <sequence>, line <line number>
Procedure Error 23017	Duplicate symbol <symbol> You have duplicate definitions for a symbol.
Procedure Error 23018	Procedure owner <owner>not found
Procedure Error 23019	Duplicate cursor name '<cursor>'
Procedure Error 23020	Illegal option <option> for WHENEVER SQLERROR … statement
Procedure Error 23021	RETURN ROW not allowed in procedure with no return type, line <line number>
Procedure Error 23022	SQL String variable <variable> must be of character data type, line <line number>
Procedure Error 23023	Call syntax error: <syntax>, line <line number>
Procedure Error 23501	Cursor <cursor> is not open
Procedure Error 23502	Illegal number of columns in EXECUTE ... <procedure> in cursor <cursor>

Procedure Error 23503	Previous SQL operation <operation> failed in cursor <cursor>
Procedure Error 23504	Cursor <cursor> is not executed
Procedure Error 23505	Cursor <cursor> is not a SELECT statement
Procedure Error 23506	End of table in cursor <cursor>
Procedure Error 23507	Illegal type conversion in cursor <cursor> from type <data type> to type <data type>
Procedure Error 23508	Illegal assignment, line <line number>
Procedure Error 23509	In <procedure> line <line number> Stmt <statement> was not in error state in RETURN SQLERROR OF ...
Procedure Error 23510	In <procedure> line <line number> Transaction cannot be set read only, because it has written already
Procedure Error 23511	In <procedure> line <line number> USING part is missing for dynamic parameters for <procedure>
Procedure Error 23512	In <procedure> line <line number> USING list is too short for <procedure>
Procedure Error 23513	In <procedure> line<line number> Comparison between incompatible types <data type> and <data type>

Procedure Error 23514	In <procedure> line <line number> type <data type> is illegal for logical expression
Procedure Error 23515	In <procedure> line <line number> assignment of parameter <parameter> in <list> list failed
Procedure Error 23516	In CALL <procedure> assignment of parameter <parameter> failed

SOLID Sorter Errors

Error code	Description
Sorter Error 24001	Sort failed due to insufficient configured TmpDir space
Sorter Error 24002	Sort failed due to insufficient physical TmpDir space
Sorter Error 24003	Sort failed due to insufficient sort buffer space
Sorter Error 24004	Sort failed due to too long row (internal failure)
Sorter Error 24005	Sort failed due to I/O error

B

Configuration Parameters

By managing the parameters of your SOLID *Embedded Engine*, you can modify the environment, performance, and operation of the server.

When SOLID *Embedded Engine* is started, it attempts to open the configuration file `solid.ini` in the current directory. The configuration values for the server parameters are included in this file. If the file does not exist, SOLID *Embedded Engine* will use the default settings for the parameters. Also, if a value for a parameter is not set in the `solid.ini` file, SOLID *Embedded Engine* will use a default value for the parameter. The default values depend on the operating system you are using.

Generally, the default settings offer the best performance and operability, but in some special cases modifying a parameter will improve performance. You can change the parameters either by using the SOLID *Remote Control* parameter page or by editing the configuration file `solid.ini`.

General Section

[General]	Description	Default
MaxOpenFiles	the maximum number of files kept concurrently open during SOLID Embedded Engine sessions	OS depend.
BackupDirectory	the directory for backup files	no default
BackupCopyLog	if set to `yes`, backup operation will copy log files to the backup directory	yes
BackupDeleteLog	if set to `yes`, old log files will be deleted after backup operation	yes
BackupCopyIniFile	if set to `yes`, `solid.ini` file will be copied to the backup directory	yes
Checkpoint Interval	the number of inserts made in the database that causes automatic checkpoint creation	5000
MergeInterval	the number of index inserts made in the database that causes the merge process to start	Cache size depend.
Readonly	if set to `yes`, database is set to read-only mode	no
LongSequential SearchLimit	the number of sequential fetches after which search is treated as long sequential search	500
SearchBuffer Limit	the maximum percentage of search buffers from the total buffered memory reserved for open cursors	50
Transaction HashSize	the hash table size for incomplete transactions	Cache size depend.

IndexFile Section

[IndexFile]	Description	Default
FileSpec_[1-N]	the file name followed with maximum size (in bytes) of that database file, for example: `c:\sol1.db 2000000` This parameter also has an optional parameter after the maxsize: physical drive number. The number value itself is not essential, but it is used as a hint for I/O threads on which I/O requests can be parallelized. This file must be stored to a local drive using local disk names to avoid problems with network I/O and to achieve better performance.	solid.db 2147483647
BlockSize	the block size of the index file in bytes; use multiple of 2 KB: minimum 2048, maximum 16384	8192
CacheSize	the size of database cache memory for the server in bytes; the minimum 512 kb	OS depend.
ExtendIncrement	the number of blocks that is allocated at one time when SOLID *Embedded Engine* needs to allocate more space for the database file	50
ReadAhead	sets the number of prefetched index leafs during long sequential searches	4
PreFlushPercent	Percentage of page buffer which is kept clean by preflush thread	5

Logging Section

[Logging]	Description	Default
LogEnabled	whether logging is enabled or not	yes
BlockSize	the block size of log files	2048
MinSplitSize	when this file size is reached, logging will be continued to the following log file after the next checkpoint	1 MB
FileNameTemplate	the path and naming convention used when creating log files; template characters are replaced with sequential numbering; for example: c:\solid\log\sol#####.log This file must be stored to a local drive using local disk names to avoid problems with network I/O and to achieve better performance.	sol#####.log
DigitTemplateChar	the template character that will be replaced in the name template of the log file	#

Communication Section

[Com]	Description	Default
Listen	the network name for server; the protocol and name that SOLID Embedded Engine uses when starting listening to the network	OS depend.
Connect	the network name for client; the protocol and name that a SOLID Embedded Engine client uses for server connection; in a Windows environment ODBC Data Source Name overrides the value of this parameter	OS depend.
MaxPhysMsgLen	the maximum length of a single physical network message in bytes; longer network messages will be split into smaller messages of this size	OS depend.
ReadBufSize	the buffer size in bytes for the data read from the network	OS depend.
WriteBufSize	the buffer size in bytes for the data written into the network	OS depend.
Trace	if parameter set to yes, trace information on network messages is written to a file specified with the TraceFile parameter	no
TraceFile	if parameter Trace is set to yes, trace information on network messages is written to a file specified with this parameter	soltrace.out

Data Sources

[Data Sources]	Description	Default
<logical name> = <network name>, <Description>	These parameters can be used to give a logical name to a SOLID *Embedded Engine.*	

Server Section

[Srv]	Description	Default
RowsPerMessage	the number of rows returned from the server in one network message	10
ConnectTimeOut	specifies the continuous idle time in minutes after that an connection is dropped; negative or zero value means infinite	480
AbortTimeOut	specifies the time in minutes after that an idle transaction is aborted; negative or zero value means infinite	120
Threads	the number of threads used for database access in SOLID *Embedded Engine*.	OS depend.
Echo	if set to `yes`, contents of `solmsg.out` file are displayed also at the server's command window	no
Name	the informal name of the server, equivalent to the -n command line option	
AllowConnect	if set to no only connections from Remote Control are allowed	yes
MessageLogSize	defines the maximum size of the solmsg.out file in bytes. The deafult is 60 KB.	OS depend.
MaxOpenCursors	The maximum number of cursors that a database client can have simultaneously open.	1000

SQL Section

[SQL]	Description	Default
Info	set the level of informational messages [0-8] printed from the server; information is written into file defined by parameter InfoFileName, (0=no info, 8=all info)	0
SQLInfo	set the level of informational SQL level messages [0-8] ; information is written into file defined by parameter InfoFileName, (0=no info, 8=all info)	no default
InfoFileName	default global info file name	SOLTRACE.OUT
InfoFileSize	maximum size of the info file. The default is 1 MB	no default
InfoFileFlush	if set to yes, flushes info file after every write operation	yes
SortArraySize	the size of the array that SQL uses when ordering result set; for optimal performance this should be as big as the biggest retrieved result set that cannot be ordered by key values; for large sorts use external sorter	OS depend.
ProcedureCache	the size of cache memory for parsed procedures in number of procedures	5
MaxNestedProcedures	The maximum number of allowed nested procedures.	16
MaxBlobExpressionSize	The maximum size of LONG VARCHAR columns in KBs that can be used in string functions	64

Sorter Section

[Sorter]	Description	Default
MaxCacheUse Percent	maximum percentage of cache pages used for sorting; range from 10% to 50%	
MaxMemPerSort	maximum memory available in bytes for one sort	
MaxFilesTotal	maximum number of files used for sorting	
TmpDir_[1-N]	name of the directory that contains temporary files created during sorting	no default

C

Data Types

Supported Data Types in SOLID *Embedded Engine*

The tables in this appendix list the supported data types by category. the following abbreviations are used in each table.

Abbreviation	Description
DEFLEN	the defined length of the column; e.g. for CHAR(24) the precision and length is 24
DEFPREC	the defined precision; e.g. for NUMERIC(10,3) it is 10
DEFSCALE	the defined scale; e.g. for NUMERIC(10,3), it is 3
MAXLEN	the maximum length of column
N/A	not applicable

Character Data Types

Data type	Size	Precision	Scale	Length	Display size
CHAR, WCHAR	2 G*	DEFLEN	N/A	DEFLEN	DEFLEN
VARCHAR, WVARCHAR	2 G**	DEFLEN	N/A	DEFLEN	DEFLEN
LONG VAR-CHAR, LONG WVARCHAR	2 G	MAXLEN	N/A	MAXLEN	MAXLEN

* default is 1

** default is 254

Numeric Data Types

Data type	Range	Precision	Scale	Length	Display size
DECIMAL	±3.6e16	16	DEFSCALE	18	18
NUMERIC	±3.6e16	DEFPREC	DEFSCALE	DEFPREC +2	DEFPREC +2
TINYINT	[-128, 127] [0, 255]	3	0	1 (bytes)	4 (signed) 3 (unsigned)
SMALLINT	[-32768, 32767] [0, 65535]	5	0	2 (bytes)	6 (signed) 5 (unsigned)
INTEGER	$[-2^{31}, 2^{31-1}]$ $[0, 2^{32-1}]$	10	0	4 (bytes)	11 (signed) 10 (unsigned)
REAL	±1.7014117 e38	7	N/A	4 (bytes)	13
FLOAT	±8.9884657 e307	15	N/A	8 (bytes)	22
DOUBLE PRECISION	±8.9884657 e307	15	N/A	8 (bytes)	22

Binary Data Types

Data type	Size	Precision	Scale	Length	Display size
BINARY	2 G*	DEFLEN	N/A	DEFLEN	DEFLEN x 2
VARBINARY	2 G**	DEFLEN	N/A	DEFLEN	DEFLEN x 2
LONG VAR-BINARY	2 G	MAXLEN	N/A	MAXLEN	MAXLEN x 2

* default is 1

** default is 254

Date Data Type

Data type	Range	Precision	Scale	Length	Display size
DATE	N/A	10*	N/A	6**	10*

* the number of characters in the yyyy-mm-dd format
** the size of the DATE_STRUCT structure

Time Data Type

Data type	Range	Precision	Scale	Length	Display size
TIME	N/A	8*	N/A	6**	8*

* the number of characters in the hh:mm:ss format
** the size of the TIME_STRUCT structure

Timestamp Data Type

Data type	Range	Precision	Scale	Length	Display size
TIMESTAMP	N/A	19*	9	16**	19/29***

* the number of characters in the 'yyyy-mm-dd hh:mm:ss.ffffffff' format
** the size of the TIMESTAMP_STRUCT structure
*** size is 29 with a decimal fraction part

The Smallest Possible Non-zero Numbers

Data type	Value
DOUBLE	2.2250738585072014e-308
REAL	1.175494351e-38

Description of Different Column Values in the Tables

The range of a numeric column refers to the minimum and maximum values the column can store. The size of character columns refers to the maximum length of data that can be stored in the column of that data type.

The precision of a numeric column refers to the maximum number of digits used by the data type of the column. The precision of a non-numeric column refers to the defined length of the column.

The scale of a numeric column refers to the maximum number of digits to the right of the decimal point. Note that for the approximate floating point number columns, the scale is undefined, since the number of digits to the right of the decimal point is not fixed.

The length of a column is the maximum number of bytes returned to the application when data is transferred to its default C type. For character data, the length does not include the null termination byte. Note that the length of a column may differ from the number of bytes needed to store the data on the data source.

The display size of a column is the maximum number of bytes needed to display data in character form.

D

SOLID SQL Syntax

The SOLID *Embedded Engine* SQL syntax is based on the ANSI X3H2-1989 level 2 standard including important ANSI X3H2-1992 (SQL2) extensions. User and role management services missing from previous standards are based on the ANSI SQL3 draft.

This appendix presents a simplified description of the SQL statements including some examples. The same information is included in the **SOLID Programmer Guide.**

ADMIN COMMAND

```
ADMIN COMMAND 'command-name'
command-name ::= BACKUP | BACKUPLIST | CLOSE |
   ERRORCODE | EXIT | HELP | MAKECP | MESSAGES |
   SHUTDOWN | MONITOR | OPEN | PARAMETERS |
   PERFMON | PID | REPORT | SHUTDOWN | STATUS |
   STATUS BACKUP | THROWOUT | TRACE | USERLIST |
   VERSION
```

Usage

This SQL extension executes administrator commands. Syntax for the extension is:

```
ADMIN COMMAND 'command-name'
```

where `command-name` is a SOLID *Remote Control* (Teletype) command string. The result set contains two columns: RC INTEGER and TEXT VARCHAR(254). Integer column RC is a command return code (0 if success), varchar column TEXT is the command reply. The TEXT field contains same lines that a displayed on SOLID *Remote Control* (Teletype) screen, one line per one result row.

Note that all options of the ADMIN COMMAND are not transactional and cannot be rolled back.

Following is a description of the syntax for each ADMIN COMMAND command option:

Option Syntax	Description
ADMIN COMMAND 'backup [*backup_directory*]'	Makes a backup of the database. The default backup directory is the one defined in configuration parameter **General.Backup.Directory**. The backup directory may also be given as an argument. For example, **backup abc** creates backup on directory ‘abc’. All directory definitions are relative to the SOLID *Embedded Engine* working directory.
ADMIN COMMAND 'backuplist'	Displays a status list of last backups.
ADMIN COMMAND 'close'	Closes server from new connections; no new connections are allowed.
ADMIN COMMAND 'errorcode *SOLID_error_code'*	Displays a description of an error code. Give the code number as an argument. For example, ‘e**rrorcode 10033**
ADMIN COMMAND 'help'	Displays available commands.

ADMIN COMMAND 'makecp'	Makes a checkpoint.
ADMIN COMMAND 'messages'	Displays server messages.
ADMIN COMMAND 'monitor {**on** | **off**} [user *username* | *user id*]'	Sets server monitoring on and off. Monitoring logs user activity and SQL calls to SOLTRACE.OUT file
ADMIN COMMAND 'open'	Opens server for new connections; new connections are allowed.
ADMIN COMMAND 'parameters [*name*]'	Displays server parameter values. For example: ■ **parameter** used alone displays all parameters. ■ **parameter genera**l displays all parameters from section "general." ■ **parameter general.readonly** displays a single parameter "readonly" from section "general."
ADMIN COMMAND 'perfmon [**-c**]'	Returns performance statistics from the server. The -c option returns all values as counter. By default, some values are averages/second.
ADMIN COMMAND 'pid'	Returns server process id.
ADMIN COMMAND 'report *filename'*	Generates a report of server info to a file given as an argument.
ADMIN COMMAND 'shutdown'	Stops SOLID *Embedded Engine*.
ADMIN COMMAND 'status'	Displays server statistics.
ADMIN COMMAND 'status backup'	Displays status of the last started backup. The status can be one of the following: ■ If the last backup was successful or any backups have not been requested, the output is 0 SUCCESS. ■ If the backup is in process; for example, started but not ready yet, the output is 14003 ACTIVE. ■ If the last backup failed, the output is: *errorcode* ERROR where the *errcode* shows the reason for the failure
ADMIN COMMAND 'throwout {*username* | *userid* | **all'**	Exits users from SOLID *Embedded Engine*. To exit a specified user, give the user id as an argument. To throw out all users, use the keyword ALL as an argument.
ADMIN COMMAND 'trace {on | **off**} **sql** | **rpc** | **sync'**	Sets server trace on or off. This command is similar to the **monitor** command, but traces different entities and a different levels. By default, the output is witten to the SOLTRACE.OUT file.

ADMIN COMMAND 'userlist [**-l**] [*name* | *id*]'	Displays a list of users. option -l displays more detailed output.
ADMIN COMMAND 'version'	Displays server version info.

Example

```
ADMIN COMMAND 'USERLIST';
```

ALTER TABLE

```
ALTER TABLE base-table-name
  {ADD [COLUMN] column-identifier data-type |
   DROP [COLUMN] column-identifier |
   RENAME [COLUMN]
      column-identifier column-identifier |
   MODIFY [COLUMN]
      column-identifier data-type} |
   MODIFY SCHEMA schema-name |
   SET {OPTIMISTIC | PESSIMISTIC}
```

Usage

The structure of a table may be modified through the `ALTER TABLE` statement. Within the context of this statement, columns may be added, modified, or removed.

The owner of a table can be changed using the ALTER TABLE base-table-name MODIFY SCHEMA schema-name statement. This statement gives all rights to the new owner of the table including creator rights. The old owner's access rights to the table, excluding the creator rights, are preserved.

Individual tables can be set to optimistic or pessimistic with the command ALTER TABLE base-table-name `SET {OPTIMISTIC | PESSIMISTIC}`. By default, all tables are optimistic. A database-wide default can be set in the `General` section of the configuration file with the parameter `Pessimistic = yes`.

Example

```
ALTER TABLE TEST ADD X INTEGER;
ALTER TABLE TEST RENAME COLUMN X Y;
ALTER TABLE TEST MODIFY COLUMN X SMALLINT;
ALTER TABLE TEST DROP COLUMN X;
```

ALTER USER

```
ALTER USER user-name IDENTIFIED BY password
```

Usage

The password of a user may be modified through the `ALTER USER` statement.

Example

```
ALTER USER MANAGER IDENTIFIED BY O2CPTG;
```

CALL

```
CALL procedure-name [(parameter [, parameter ...])]
```

Usage

Stored procedures are called with statement `CALL`.

Example

```
CALL proctest;
```

COMMIT

```
COMMIT WORK
```

Usage

The changes made in the database are made permanent by `COMMIT` statement. It terminates the transaction.

Example

```
COMMIT WORK;
```

CREATE EVENT

```
CREATE EVENT event-name
   [(parameter-definition
    [, parameter-definition ...])]
```

Usage

Event alerts are used to signal an event in the database. Events are simple objects with a name. The use of event alerts removes resource consuming database polling from applications.

An event object is created with the SQL statement

`CREATE EVENT event-name [parameter-list]`
The name can be any user-specified alphanumeric string. The parameter list specifies parameter names and parameter types. The parameter types are normal SQL types.

Events are dropped with the SQL statement

`DROP EVENT event-name`
Events are triggered and received inside stored procedures. Special stored procedure statements are used to trigger and receive events.

The event is triggered with the stored procedure statement

`POST EVENT event-name [parameters]`
Event parameters must be local variables or parameters in the stored procedure where the event is triggered. All clients that are waiting for the posted event will receive the event.

To make a procedure wait for an event to happen, the `WAIT EVENT` construct is used in a stored procedure:

```
wait-event-statement ::=
   WAIT EVENT
      [event-specification ...]
   END WAIT

event-specification ::=
   WHEN event-name (parameters) BEGIN
      statements
   END EVENT
```

Each connection has its own event queue. To specify the events to be collected in the event queue command REGISTER EVENT event-name (parameters) is used. Events are removed from the event queue with command UNREGISTER EVENT event-name (parameters).

Example of a procedure that waits for an event:

```
"create procedure event-wait(i1 integer)
returns (result varchar)
begin
declare i integer;
declare c char(4);

i := 0;

wait event
   when test1 begin
      result := 'event1';
      return;
   end event

   when test2(i) begin
   end event

   when test3(i, c) begin
  end event
end wait

if i <> 0 then
   result := 'if';
   post event test1;
else
   result := 'else';
   post event test2(i);
   post event test3(i, c);
end if
end";
```

The creator of an event or the database administrator can grant and revoke access rights. Access rights can be granted to users and roles. The select access right gives waiting access to an event. The insert access right gives triggering access to an event.

Example

```
CREATE EVENT ALERT1(I INTEGER, C CHAR(4));
```

CREATE INDEX

```
CREATE [UNIQUE] INDEX index-name
   ON base-table-name
      (column-identifier [ASC | DESC]
       [, column-identifier [ASC | DESC]] ...)
```

Usage

Creates an index for a table based on the given columns. Keyword UNIQUE specifies that columns being indexed must contain unique values. Keywords ASC and DESC specify whether the given columns should be indexed in ascending or descending order. If not specified ascending order is used.

Example

```
CREATE UNIQUE INDEX UX_TEST ON TEST (I);
CREATE INDEX X_TEST ON TEST (I, J);
```

CREATE PROCEDURE

```
CREATE PROCEDURE procedure-name
   [(parameter-definition
      [, parameter-definition ...])]
   [RETURNS (parameter-definition
      [, parameter-definition ...])]
   BEGIN procedure-body END;

parameter-definition ::= parameter-name data-type

procedure-body ::= [declare-statement; ...]
   procedure-statement; [procedure-statement; ...]

declare-statement ::= DECLARE variable-name
```

```
   data-type

procedure-statement ::= prepare-statement |
   exec-statement | fetch-statement |
   control-statement | post-statement |
   wait-event-statement | wait-register-statement

prepare-statement ::= EXEC SQL PREPARE
   cursor-name sql-statement
```

```
execute-statement ::=
   EXEC SQL EXECUTE
      cursor-name
      [USING (variable [, variable ...])]
      [INTO (variable [, variable ...])] |
   EXEC SQL {CLOSE | DROP} cursor-name |
   EXEC SQL {COMMIT | ROLLBACK} WORK |
   EXEC SQL SET TRANSACTION {READ ONLY | READ WRITE} |
   EXEC SQL WHENEVER SQLERROR {ABORT | ROLLBACK [WORK],
      ABORT}
   EXEC SEQUENCE sequence-name.CURRENT INTO variable |
   EXEC SEQUENCE sequence-name.NEXT INTO variable |
   EXEC SEQUENCE sequence-name SET VALUE USING variable

fetch-statement ::= EXEC SQL FETCH cursor-name

post-statement ::= POST EVENT event-name [parameters]

wait-event-statement ::=
   WAIT EVENT
      [event-specification ...]
   END WAIT

event-specification ::=
   WHEN event-name (parameters) BEGIN
      statements
   END EVENT

wait-register-statement ::=
   REGISTER event-name (parameters) |
   UNREGISTER event-name (parameters)

control-statement ::=
   SET variable-name = value |
   variable-name := value |
   WHILE expression
      LOOP procedure-statement... END LOOP |
```

```
LEAVE |
IF expression THEN procedure-statement ...
   [ ELSEIF procedure-statement ... THEN] ...
   ELSE procedure-statement ... END IF |
RETURN | RETURN SQLERROR OF cursor-name | RETURN ROW
```

Usage

Stored procedures are simple programs, or procedures, that are executed in the server. The user can create a procedure that contains several SQL statements or a whole transaction and execute it with a single call statement. Usage of stored procedures reduces network traffic and allows more strict control to access rights and database operations.

Procedures are created with the statement

CREATE PROCEDURE name body

and dropped with the statement

DROP PROCEDURE name

Procedures are called with the statement

CALL name [parameter ...]

Procedures can take several input parameters and return a single row or several rows as a result. The result is built from specified output parameters. Procedures are thus used in ODBC in the same way as the `SQL SELECT` statement.

Procedures are owned by the creator of the procedure. Specified access rights can be granted to other users. When the procedure is run, it has the creator's access rights to database objects.

The stored procedure syntax is a proprietary syntax modeled from SQL3 specifications and dynamic SQL. Procedures contain control statements and SQL statements.

The following control statements are available in the procedures:

Control statement	Description
set *variable* = *expression*	Assigns a value to a variable. The value can be either a literal value (e.g., 10 or 'text') or another variable. Parameters are considered as normal variables.
variable := *expression*	Alternate syntax for assigning values to variables.

while *expr* loop *statement-list* end loop	Loops while expression is true.
leave	Leaves the innermost while loop and continues executing the procedure from the next statement after the keyword end loop.

if
expr
then
statement-list1
else
statement-list2
end if
	Executes *statements-list1* if expression *expr* is true; otherwise, executes *statement-list2*.

if *expr1* then *statement-list1* elseif *expr2* then *statement-list2* end if	If *expr1* is true, executes *statement-list1*. If *expr2* is true, executes *statement-list2*. The statement can optionally contain multiple *elseif* statements and also an *else* statement.
return	Returns the current values of output parameters and exits the procedure. If a procedure has a one *return row* statement, *return* behaves like *return norow.*
return sqlerror of *cursor-name*	Returns the sqlerror associated with the cursor and exits the procedure.
return row	Returns the current values of output parameters and continues execution.

return norow	Returns the end of the set and exits the procedure.

All SQL DML and DDL statements can be used in procedures. Thus the procedure can, for example, create tables or commit a transaction. Each SQL statement in the procedure is atomic.

Preparing SQL Statements

The SQL statements are first prepared with the statement

EXEC SQL PREPARE *cursor sql-statement*

The *cursor* specification is a cursor name that must be given. It can be any unique cursor name inside the transaction. Note that if the procedure is not a complete transaction, other open cursors outside the procedure may have conflicting cursor names.

Executing Prepared SQL Statements

The *SQL statement* is executed with the statement

EXEC SQL EXECUTE *cursor* [*opt-using*] [*opt-into*]

The optional *opt-using* specification has the syntax

USING (*variable-list*)

where *variable-list* contains a list of procedure variables or parameters separated by a comma. These variables are input parameters for the SQL statement. The SQL input parameters are marked with the standard question mark syntax in the prepare statement. If the SQL statement has no input parameters, the USING specification is ignored.

The optional *opt-into* specification has the syntax

INTO (*variable-list*)

where *variable-list* contains the variables that the column values of the SQL SELECT statement are stored into. The INTO specification is effective only for SQL SELECT statements.

After the execution of UPDATE, INSERT and DELETE statements an additional variable is available to check the result of the statement. Variable SQLROWCOUNT contains the number of rows affected by the last statement.

Fetching Results

Rows are fetched with the statement

EXEC SQL FETCH *cursor*

If the fetch completed successfully, the column values are stored into the variables defined in the *opt-into* specification.

Checking for Errors

The result of each EXEC SQL statement executed inside a procedure body is stored into the variable SQLSUCCESS. This variable is automatically generated for every procedure. If the previous SQL statement was successful, a value one is stored into SQLSUCCESS. After a failed SQL statement, a value zero is stored into SQLSUCCESS.

EXEC SQL WHENEVER SQLERROR {ABORT | [ROLLBACK [WORK], ABORT}

is used to decrease the need for IF NOT SQLSUCCESS THEN tests after every executed SQL statement in a procedure. When this statement is included in a stored procedure all return values of executed statements are checked for errors. If statement execution returns an error, the procedure is automatically aborted. Optionally the transaction can be rolled back.

The error string of latest failed SQL statement is stored into variable SQLERRSTR.

Using Transactions

EXEC SQL {COMMIT | ROLLBACK} WORK

is used to terminate transactions.

EXEC SQL SET TRANSACTION {READ ONLY | READ WRITE}

is used to control the type of transactions.

Using Sequencer Objects and Event Alerts

Refer to the usage of the CREATE SEQUENCE and CREATE EVENT statements.

Procedure Stack Functions

The following functions may be used to analyze the current contents of the procedure stack: PROC_COUNT(), PROC_NAME(N), PROC_SCHEMA(N).

PROC_COUNT() returns the number of procedures in the procedure stack. This includes the current procedure.

PROC_NAME(N) returns the Nth procedure name is the stack. First procedure position is zero.

PROC_SCHEMA(N) returns the schema name of the Nth procedure in procedure stack.

Example 1

```
"create procedure test2(tableid integer)
```

```
   returns (cnt integer)
begin
   exec sql prepare c1 select count(*) from
      sys_tables where id > ?;
   exec sql execute c1 using (tableid) into
      (cnt);
   exec sql fetch c1;
end";
```

Example 2

```
-- This procedure can only be used with SOLID Embedded Engine    --
--  version 2.2 or later.                                         --
"create procedure return_tables
returns (name varchar)
begin
        exec sql whenever sqlerror rollback, abort;
        exec sql prepare c1 select table_name
             from sys_tables;
        exec sql execute c1 into (name);
        while sqlsuccess loop
            exec sql fetch c1;
            if not sqlsuccess
            then leave;
            end if
            return row;
        end loop;
        exec sql close c1;
end";
```

CREATE ROLE

```
CREATE ROLE role-name
```

Usage

Creates a new user role.

Example

```
CREATE ROLE GUEST_USERS;
```

CREATE SEQUENCE

```
CREATE [DENSE] SEQUENCE sequence-name
```

Usage

Sequencer objects are objects that are used to get sequence numbers.

Using a dense sequence guarantees that there are no holes in the sequence numbers. The sequence number allocation is bound to the current transaction. If the transaction rolls back, also the sequence number allocations are rolled back. The drawback of dense sequences is that the sequence is locked out from other transactions until the current transaction ends.

Using a sparse sequence guarantees uniqueness of the returned values, but they are not bound to the current transaction. If a transaction allocates a sparse sequence number and later rolls back, the sequence number is simply lost.

The advantage of using a sequencer object instead of a separate table is that the sequencer object is specifically fine-tuned for fast execution and requires less overhead than normal update statements.

Sequence values can be incremented and used within SQL statements. These constructs can be used in SQL:

sequence-name.CURRVAL

sequence-name.NEXTVAL

Sequences can also be used inside stored procedures. The current sequence value can be retrieved using the following stored procedure statement:

EXEC SEQUENCE sequence-name.CURRENT INTO variable

The new sequence value can be retrieved using the following stored procedure statement:

EXEC SEQUENCE sequence-name.NEXT INTO variable

Sequence values can be set with the following stored procedure statement:

EXEC SEQUENCE sequence-name SET VALUE USING variable

Select access rights are required to retrieve the current sequence value. Update access rights are required to allocate new sequence values. These access rights are granted and revoked in the same way as table access rights.

Examples

```
CREATE DENSE SEQUENCE SEQ1;

INSERT INTO ORDER (id) VALUES (order_sequence.NEXTVAL);
```

CREATE TABLE

```
CREATE TABLE base-table-name
   (column-element [, column-element] ...)
base-table-name ::= base-table-identifier |
   schema-name.base-table-identifier

column-element ::= column-definition |
   table-constraint-definition

column-definition ::= column-identifier
   data-type
      [column-constraint-definition
         [column-constraint-definition] ...]

column-constraint-definition ::=
   NOT NULL | NOT NULL UNIQUE |
   NOT NULL PRIMARY KEY | CHECK (check-condition)

table-constraint-definition ::=
   UNIQUE (column-identifier
      [, column-identifier] ...) |
   PRIMARY KEY (column-identifier
      [, column-identifier] ...) |
   CHECK (check-condition) |
```

```
FOREIGN KEY (column-identifier
   [, column-identifier] ...)
   REFERENCES table-name
    (column-identifier [, column-identifier] ...)
```

Usage

Tables are created through the CREATE TABLE statement. The CREATE TABLE statement requires a list of the columns created, the data types, and, if applicable, sizes of values within each column, in addition to other related alternatives (such as whether or not null values are permitted).

Example

```
CREATE TABLE DEPT (DEPTNO INTEGER NOT NULL, DNAME VARCHAR, PRIMARY
KEY(DEPTNO));

CREATE TABLE DEPT2 (DEPTNO INTEGER NOT NULL PRIMARY KEY, DNAME VARCHAR);

CREATE TABLE DEPT3 (DEPTNO INTEGER NOT NULL UNIQUE, DNAME VARCHAR);

CREATE TABLE DEPT4 (DEPTNO INTEGER NOT NULL, DNAME VARCHAR,
UNIQUE(DEPTNO));

CREATE TABLE EMP (DEPTNO INTEGER, ENAME VARCHAR, FOREIGN KEY (DEPTNO)
REFERENCES DEPT (DEPTNO));

CREATE TABLE EMP2 (DEPTNO INTEGER, ENAME VARCHAR, CHECK (ENAME IS NOT
NULL), FOREIGN KEY (DEPTNO) REFERENCES DEPT (DEPTNO));
```

CREATE USER

```
CREATE USER user-name IDENTIFIED BY password
```

Usage

Creates a new user with a given password.

Example

```
CREATE USER HOBBES IDENTIFIED BY CALVIN;
```

CREATE VIEW

```
CREATE VIEW viewed-table-name
   [(column-identifier
      [, column-identifier]... )]
   AS query-specification
```

Usage

A view can be viewed as a virtual table; that is, a table that does not physically exist, but rather is formed by a query specification against one or more tables.

Example

```
CREATE VIEW TEST_VIEW
   (VIEW_I, VIEW_C, VIEW_ID)
   AS SELECT I, C, ID FROM TEST;
```

DELETE

```
DELETE FROM table-name
   [WHERE search-condition]
```

Usage

Depending on your search condition the specified row(s) will be deleted from a given table.

Example

```
DELETE FROM TEST WHERE ID = 5;
DELETE FROM TEST;
```

DELETE (positioned)

```
DELETE FROM table-name WHERE CURRENT OF cursor-name
```

Usage

The positioned DELETE statement deletes the current row of the cursor.

Example

```
DELETE FROM TEST WHERE CURRENT OF MY_CURSOR;
```

DROP EVENT

```
DROP EVENT event-name
```

Usage

The DROP EVENT statement removes the specified event from the database.

Example

```
DROP EVENT EVENT-TEST;
```

DROP INDEX

```
DROP INDEX index-name
```

Usage

The `DROP INDEX` statement removes the specified index from the database.

Example

```
DROP INDEX UX_TEST;
```

DROP PROCEDURE

```
DROP PROCEDURE procedure-name
```

Usage

The `DROP PROCEDURE` statement removes the specified procedure from the database.

Example

```
DROP PROCEDURE PROCTEST;
```

DROP ROLE

```
DROP ROLE role-name
```

Usage

The `DROP ROLE` statement removes the specified role from the database.

Example

```
DROP ROLE GUEST_USERS;
```

DROP SEQUENCE

```
DROP SEQUENCE sequence-name
```

Usage

The DROP SEQUENCE statement removes the specified sequence from the database.

Example

```
DROP SEQUENCE SEQ1;
```

DROP TABLE

```
DROP TABLE base-table-name
```

Usage

The DROP TABLE statement removes the specified table from the database.

Example

```
DROP TABLE TEST;
```

DROP USER

```
DROP USER user-name
```

Usage

The DROP USER statement removes the specified user from the database.

Example

```
DROP USER HOBBES;
```

DROP VIEW

```
DROP VIEW viewed-table-name
```

Usage

The DROP VIEW statement removes the specified view from the database.

Example

```
DROP VIEW TEST_VIEW;
```

EXPLAIN PLAN FOR

```
EXPLAIN PLAN FOR sql-statement
```

Usage

The EXPLAIN PLAN FOR statement shows the selected search plan for the specified SQL statement.

Example

```
EXPLAIN PLAN FOR select * from tables;
```

GRANT

```
GRANT {ALL | grant-privilege
   [, grant-privilege]...}
   ON table-name
   TO {PUBLIC | user-name [, user-name]... |
      role-name [, role-name]... }
   [WITH GRANT OPTION]

GRANT role-name TO user-name

grant-privilege ::= DELETE | INSERT | SELECT |
   UPDATE [( column-identifier
      [, column-identifier]... )] |
   REFERENCES [( column-identifier

      [, column-identifier]... )]
GRANT EXECUTE ON procedure-name
   TO {PUBLIC | user-name [, user-name]... |
      role-name [, role-name]... }
```

```
GRANT {SELECT | INSERT} ON event-name
   TO {PUBLIC | user-name [, user-name]... |
      role-name [, role-name]... }

GRANT {SELECT | UPDATE} ON sequence-name
   TO {PUBLIC | user-name [, user-name]... |
      role-name [, role-name]... }
```

Usage

The GRANT statement is

1. used to grant privileges to the specified user or role.
2. used to grant privileges to the specified user by giving

the user the privileges of the specified role.

If you do use the optional WITH GRANT OPTION, you give permission for the user(s) to whom you are granting the privilege to pass it on to other users.

Example

```
GRANT GUEST_USERS TO CALVIN;
GRANT INSERT, DELETE ON TEST TO GUEST_USERS;
```

INSERT

```
INSERT INTO table-name [(column-identifier
      [, column-identifier]...)]
   VALUES (insert-value[, insert-value]... )
```

Usage

There are several variations of the INSERT statement. In the simplest instance, a value is provided for each column of the new row in the order specified at the time the table was defined (or altered). In the preferable form of the INSERT statement the columns are specified as part of the statement and they needn't to be in any specific order as long as the orders of the column and value lists match with one another.

Example

```
INSERT INTO TEST (C, ID) VALUES (0.22, 5);
INSERT INTO TEST VALUES (0.35, 9);
```

INSERT (Using Query)

```
INSERT INTO table-name [( column-identifier
   [, column-identifier]... )]
   query-specification
```

Usage

The query specification creates a virtual table. Using the INSERT statement the rows of created virtual table are inserted into the specified table (the degree and data types of the virtual table and inserted columns must match).

Example

```
INSERT INTO TEST (C, ID) SELECT A, B FROM INPUT_TO_TEST;
```

REVOKE (Role from User)

```
REVOKE {role-name [, role-name]... }
   FROM {PUBLIC | user-name [, user-name]... }
```

Usage

The REVOKE statement is used to take a role away from users.

Example

```
REVOKE GUEST_USERS FROM HOBBES;
```

REVOKE (Privilege from Role or User)

```
REVOKE
   {revoke-privilege [, revoke-privilege]... }
   ON table-name
   FROM {PUBLIC | user-name [, user-name]... |
      role-name [, role-name]... }

revoke-privilege ::= DELETE | INSERT |
   SELECT |
   UPDATE [( column-identifier
      [, column-identifier]... )] |
   REFERENCES

REVOKE EXECUTE ON procedure-name
   FROM {PUBLIC | user-name [, user-name]... |
      role-name [, role-name]... }

REVOKE {SELECT | INSERT} ON event-name FROM
   {PUBLIC | user-name [, user-name]... |
       role-name [, role-name]... }

REVOKE {SELECT | INSERT} ON sequence-name
   FROM {PUBLIC | user-name [, user-name]... |
      role-name [, role-name]... }
```

Usage

The REVOKE statement is used to take privileges away from users and roles.

Example

```
REVOKE INSERT ON TEST FROM GUEST_USERS;
```

ROLLBACK

```
ROLLBACK WORK
```

Usage

The changes made in the database are discarded by ROLLBACK statement. It terminates the transaction.

Example

```
ROLLBACK WORK;
```

SELECT

```
SELECT [ALL | DISTINCT] select-list
   FROM table-reference-list
   [WHERE search-condition]
   [GROUP BY column-name [, column-name]... ]
   [HAVING search-condition]
   [[UNION | INTERSECT | EXCEPT] [ALL]
      select-statement]...
   [ORDER BY {unsigned integer | column-name}
      [ASC|DESC]]
```

Usage

The SELECT statement is used to retrieve information.

IMPORTANT NOTE:
SOLID provides a consistent view of data within one transaction; that is, it sees the database as it was at the moment it was started. This is implemented by the multiversion SOLID Bonsai Tree that stores the active data, that is, data that has been written to the database since the beginning of the oldest active transaction in central memory. Also a SELECT begins a new transaction and if not committed or rolled back, it remains active thus causing the Bonsai Tree to grow.

New data is merged to the main storage tree as soon as no transaction needs to see the old versions of the rows. To ensure the efficient operation of the Bonsai Tree, also commit read-only transactions as soon as all rows are retrieved. This releases the read level and allows the merge process to keep the Bonsai Tree smaller.

Using AUTOCOMMIT does not help. This is because SOLID cannot immediately commit SELECTs since the rows need to be retrieved by the client application first. In AUTOCOMMIT mode, the next SQL statement processing triggers the commit for previous SELECT statement. But if that next statement never comes, the transaction is left open until the connection timout expires.

Example

```
SELECT ID FROM TEST;
SELECT DISTINCT ID, C FROM TEST WHERE ID = 5;
SELECT DISTINCT ID FROM TEST ORDER BY ID ASC;
SELECT NAME, ADDRESS FROM CUSTOMERS UNION SELECT NAME, DEP FROM
PERSONNEL;
```

SET

```
SET SQL INFO {ON | OFF} [FILE {file-name |
      "file-name" | 'file-name'}]
   [LEVEL info-level]

SET SQL SORTARRAYSIZE {array-size | DEFAULT}

SET SQL JOINPATHSPAN {path-span | DEFAULT}

SET SQL CONVERTORSTOUNIONS
   {YES [COUNT value] | NO | DEFAULT}

SET LOCK TIMEOUT timeout-in-seconds

SET STATEMENT MAXTIME minutes

SET TRANSACTION READ ONLY

SET TRANSACTION READ WRITE

SET TRANSACTION CHECK WRITESET

SET TRANSACTION CHECK READSET

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

SET TRANSACTION ISOLATION LEVEL
   REPEATABLE READ

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
```

Usage

All the settings are read per user session unlike the settings in the `solid.ini` file which are automatically read each time SOLID Embedded Engine is started.

In `SQL INFO` the default file is a global `soltrace.out` shared by all users. If the file name is given, all future `INFO ON` settings will use that file unless a new file is set. It is recommended that the file name is given in single quotes, because otherwise the file name is converted to uppercase. The info output is appended to the file and the file is never truncated, so after the info file is not needed anymore, the user must manually delete the file. If the file open fails, the info output is silently discarded.

The default `SQL INFO LEVEL` is `4`. A good way to generate useful info output is to set info on with a new file name and then execute the SQL statement using `EXPLAIN PLAN FOR` syntax. This method gives all necessary estimator information but does not generate output from the fetches which may generate a huge output file.

The sort array is used for in memory sorts in the SQL interpreter. The minimum value for `SORTARRAYSIZE` is `100`. If a smaller value is given, minimum value 100 will be used. If large sorts are needed, it is recommended that the external sorter facility is used (in Sorter section in solid.ini) instead on using very large `SORTARRAYSIZE`.

The `COUNT` parameter in `SQL CONVERTORSTOUNIONS` tells how many ors are converted to unions. The default is 10 which should be enough in most cases.

SET STATEMENT MAXTIME sets connection specific maximum execution time in minutes. Setting is effective until a new maximum time if set. Zero time means no maximum time, which is also the default.

The `SET TRANSACTION` settings are borrowed from ANSI SQL. It sets the transaction isolation level.

Example

```
SET SQL INFO ON FILE 'sqlinfo.txt' LEVEL 5
```

SET SCHEMA

```
SET SCHEMA {USER | 'user-name'}
```

Usage

From version 2.2 SOLID *Embedded Engine* supports SQL89 style schemas for database entity name qualifying. All created database entities belong to a schema, and different schemas may contain entities with same name.

The default schema can be changed with the SET SCHEMA statement. Schema can be change to the current user name by using the SET SCHEMA USER statement. Alternatively schema can be set to ‘user-name’ which must be a valid user name in the database.

The algorithm to resolve entity names [schema-name.]table-identifier is the following:

1. If schema-name is given then table-identifier is searched only from that schema.
2. If schema-name is not given, then
   a. First table-identifier is searched from default schema. Default schema is initially the same as user name, but can be changed with SET SCHEMA statement
   b. Then table-identifier is searched from all schemas in the database. If more than one entity with same table-identifier and type (table, procedure, ...) is found, a new error code 13110 (Ambiguous entity name table-identifier) is returned.

   The SET SCHEMA statement effects only to default entity name resolution and it does not change any access rights to database entities. It sets the default schema name for unqualified names in statements that are prepared in the current session by an execute immediate statement or a prepare statement.

Example

```
SET SCHEMA 'CUSTOMERS'
```

UPDATE (Positioned)

```
UPDATE table-name
   SET [table-name.]column-identifier = {expression |
      NULL}
   [, [table-name.]column-identifier = {expression |
      NULL}]...
   WHERE CURRENT OF cursor-name
```

Usage

The positioned UPDATE statement updates the current row of the cursor. The name of the cursor is defined using ODBC API function named SQLSetCursorName.

Example

```
UPDATE TEST SET C = 0.33
WHERE CURRENT OF MYCURSOR
```

UPDATE (Searched)

```
UPDATE table-name
   SET [table-name.]column-identifier = {expression |
      NULL}
   [, [table-name.]column-identifier = {expression |
      NULL}]...
   [WHERE search-condition]
```

Usage

The UPDATE statement is used to modify the values of one or more columns in one or more rows, according the search conditions.

Example

```
UPDATE TEST SET C = 0.44 WHERE ID = 5
```

Table-reference

Table-reference

table-reference-list	::= table-reference [, table-reference …]
table-reference	::= table-name [[AS] correlation-name] | derived-table [[AS] correlation-name [(derived-column-list)]] | joined-table
table-name	::= table-identifier | schema-name.table-identifier
derived-table	::= subquery
derived-column-list	::= column-name-list
joined-table	::= cross-join | qualified-join | (joined-table)
cross-join	::= table-reference CROSS JOIN table-reference

qualified-join	::= table-reference [NATURAL] [join-type] JOIN table-reference [join-specification]
join-type	::= INNER | outer-join-type [OUTER] | UNION
outer-join-type	::= LEFT | RIGHT | FULL
join-specification	::= join-condition | named-columns-join
join-condition	::= ON search-condition
named-columns-join	::= USING (column-name-list)
column-name-list	::= column-identifier [{ , column-identifier} …]

Query-specification

Query-specification

query-specification	::= SELECT [DISTINCT | ALL] select-list table-expression
select-list	::= * | select-sublist [{, select-sublist} ...]
select-sublist	::= derived-column | [table-name | table-identifier].*
derived-column	::= expression [[AS] column-alias]]
table-expression	::= FROM table-reference-list [WHERE search-condition] [GROUP BY column-name-list [[UNION | INTERSECT | EXCEPT] [ALL] [CORRESPONDING [BY (column-name-list)]] query-specification] [HAVING search-condition]

Search-condition

Search-condition

search-condition	::= search-item | search-item { AND | OR } search-item

search-item	::= [NOT] { search-test | (search-condition) }
search-test	::= comparison-test | between-test | like-test | null-test | set-test | quantified-test | existence-test
comparison-test	::= expression { = | <> | < | <= | > | >= } { expression | subquery }
between-test	::= column-identifier [NOT] BETWEEN expression AND expression
like-test	::= column-identifier [NOT] LIKE value [ESCAPE value]
null-test	::= column-identifier IS [NOT] NULL
set-test	::= expression [NOT] IN ({ value [,value]... | subquery })
quantified-test	::= expression { = | <> | < | <= | > | >= } [ALL | ANY | SOME] subquery
existence-test	::= EXISTS subquery
subquery	::= (query-specification)

Check-condition

Check-condition

check-condition	::= check-item | check-item { AND | OR } check-item
check-item	::= [NOT] { check-test | (check-condition) }
check-test	::= comparison-test | between-test | like-test | null-test | list-test
comparison-test	::= expression { = | <> | < | <= | > | >= } { expression | subquery }
between-test	::= column-identifier [NOT] BETWEEN expression AND expression
like-test	::= column-identifier [NOT] LIKE value [ESCAPE value]

null-test	::= column-identifier IS [NOT] NULL
list-test	::= expression [NOT] IN ({ value [,value]...})

Expression

Expression

expression	::= expression-item | expression-item { + | - | * | / } expression-item
expression-item	::= [+ | -] { value | column-identifier | function | case-expression | cast-expression | (expression) }
value	::= literal | USER | variable
function	::= set-function | null-function | string-function | numeric-function | datetime-function | system-function | datatypeconversion-function
set-function	::= COUNT (*) | { AVG | MAX | MIN | SUM | COUNT } ({ ALL | DISTINCT } expression)
null-function	::= { NULLVAL_CHAR( ) | NULLVAL_INT( ) }
datatypeconversion-function	::= CONVERT_CHAR(value-exp) | CONVERT_DATE(value-exp) | CONVERT_DECIMAL(value-exp) | CONVERT_DOUBLE(value-exp) | CONVERT_FLOAT(value-exp) | CONVERT_INTEGER(value-exp) | CONVERT_LONGVARCHAR(value-exp) | CONVERT_NUMERIC(value-exp) | CONVERT_REAL(value-exp) | CONVERT_SMALLINT(value-exp) | CONVERT_TIME(value-exp) | CONVERT_TIMESTAMP(value-exp) | CONVERT_TINYINT(value-exp) | CONVERT_VARCHAR(value-exp)
case-expression	::= case-abbreviation | case-specification

case-abbreviation	::= NULLIF(value-exp, value-exp) | COALESCE(value-exp {, value-exp}…)
case-specification	::= CASE value-exp WHEN value-exp THEN {value-exp} [WHEN value-exp THEN {value-exp} …] ELSE {value-exp} END
cast-expression	::= CAST (value-exp AS -data-type)

String Function

Function	Purpose
ASCII(str)	Returns the integer equivalent of string str
CHAR(code)	Returns the character equivalent of code
CONCAT(str1, str2)	Concatenates str2 to str1
str1 { + | || } str2	Concatenates str2 to str1
INSERT(str1, start, length, str2)	Merges strings by deleting length characters from str1 and inserting str2
LCASE(str)	Converts string str to lowercase
LEFT(str, count)	Returns leftmost count characters of string str
LENGTH(str)	Returns the number of characters in str
LOCATE(str1, str2 [, start])	Returns starting position of str1 within str2
LTRIM(str)	Removes leading spaces of str
POSITION (str1 IN str2)	Returns starting position of str1 within str2
REPEAT(str, count)	Returns characters of str repeated count times
REPLACE(str1, str2, str3)	Replaces occurrences of str2 in str1 with str3
RIGHT(str, count)	Replaces the rightmost count characters of string str
RTRIM(str)	Removes trailing spaces in str
SPACE(count)	Returns a string str of count spaces

SUBSTRING(str, start, length)	Derives substring from str beginning at start
UCASE(str)	Converts str to uppercase

Numeric Function

Function	Purpose
ABS(numeric)	Absolute value of numeric
ACOS(float)	Arccosine of float
ASIN(float)	Arcsine of float
ATAN(float)	Arctangent of float
ATAN2(float1, float2)	Arctangent of the x and y coordinates, specified by float1 and float2, respectively, as an angle, expressed in radians
CEILING(numeric)	Smallest integer greater than or equal to numeric
COS(float)	Cosine of float
COT(float)	Cotangent of float
DEGREES(numeric)	Converts numeric radians to degrees
EXP(float)	Exponential value of float
FLOOR(numeric)	Largest integer less than or equal to numeric
LOG(float)	Natural logarithm of float
LOG10(float)	Base 10 log of float
MOD(integer1, integer2)	Modulus of integer1 divided by integer2
PI()	Pi as a floating point number
POWER(numeric, integer)	Value of numeric raised to the power of integer
RADIANS(numeric)	Number of radians converted from numeric
ROUND(numeric, integer)	Numeric rounded to integer
SIGN(numeric)	Sign of numeric
SQRT(float)	Square root of float
TAN(float)	Tangent of float
TRUNCATE(numeric, integer)	Numeric truncated to integer

Date Time Function

Function	Purpose
CURDATE()	Returns the current date
CURTIME()	Returns the current time
DAYNAME(date)	Returns a string with the day of the week
DAYOFMONTH(date)	Returns the day of the month as an integer between 1 and 31
DAYOFWEEK(date)	Returns the day of the week as an integer between 1 and 7, where 1 represents Sunday
DAYOFYEAR(date)	Returns the day of the year as an integer between 1 and 366
EXTRACT (date field FROM date-exp)	Isolates a single field of a datetime or a interval and converts it to a number.
HOUR(time-exp)	Returns the hour as an integer between 0 and 23
MINUTE(time-exp)	Returns the minute as an integer between 0 and 59
MONTH(date)	Returns the month as an integer between 1 and 12
MONTHNAME(date)	Returns the month name as a string
NOW()	Returns the current date and time as a timestamp
QUARTER(date)	Returns the quarter as an integer between 1 and 4
SECOND(time-exp)	Returns the second as an integer between 0 and 59
TIMESTAMPADD(interval, integer-exp, timestamp-exp)	Calculates a timetamp by adding integer-exp intervals of type interval to timestamp-exp
	Keywords used to express valid TIMESTAMPADD interval values are:
	SQL_TSI_FRAC_SECOND SQL_TSI_SECOND SQL_TSI_MINUTE SQL_TSI_HOUR SQL_TSI_DAY SQL_TSI_WEEK SQL_TSI_MONTH SQL_TSI_QUARTER SQL_TSI_YEAR

TIMESTAMPDIFF(interval, timestamp-exp1, timestamp-exp2)	Returns the integer number of intervals by which timestamp-exp2 is greater than timestamp-exp1 Keywords used to express valid TIMESTAMPDIFF interval values are: SQL_TSI_FRAC_SECOND SQL_TSI_SECOND SQL_TSI_MINUTE SQL_TSI_HOUR SQL_TSI_DAY SQL_TSI_WEEK SQL_TSI_MONTH SQL_TSI_QUARTER SQL_TSI_YEAR
WEEK(date)	Returns the week of the year as an integer between 1 and 52
YEAR(date)	Returns the year as an integer

System Function

Function	Purpose
IFNULL(exp, value)	If exp is null, returns value; if not, returns exp
USER()	Returns the user authorization name
UIC()	Returns the connection id associated with the connection

Data-type

Data-type

data-type	::= {BINARY | CHAR [length] | DATE | DECIMAL [(precision [, scale])] | DOUBLE PRECISION | FLOAT [(precision)] | INTEGER | LONG VARBINARY | LONG VARCHAR | LONG WVARCHAR | NUMERIC [(precision [, scale])] | REAL | SMALLINT | TIME | TIMESTAMP [(timestamp precision)] | TINYINT | VARBINARY | VARCHAR [(length)] } | WCHAR | WVARCHAR [length]

Date and Time Literals

Date/time literal	
date-literal	´YYYY-MM-DD´
time-literal	´HH:MM:SS´
timestamp-literal	´YYYY-MM-DD HH:MM:SS´

Pseudo Columns

The following pseudo columns may also be used in the select-list of a SELECT statement:

Pseudo column	Type	Explanation
ROWVER	VARBINARY(254)	Version of the row in a table.
ROWID	VARBINARY(10)	Persistent id for a row in a table.

ROWNUM	DECIMAL(16,2)	Row number indicates the sequence in which a row was selected from a table or set of joined rows. The first row selected has a ROWNUM of 1, the second row has 2, etc. ROWNUM is chiefly useful for limiting the number of rows returned by a query (e.g., WHERE ROWNUM < 10).

NOTE! Since ROWID and ROWVER refer to a single row, they may only be used with queries that return rows from a single table.

E

System Views and System Tables

System Views

SOLID *Embedded Engine* supports views specified in the X/Open SQL Standard.

COLUMNS

The COLUMNS system view identifies the columns which are accessible to the current user.

Column name	Data type	Description
TABLE_CATALOG	VARCHAR	reserved for future use
TABLE_SCHEMA	VARCHAR	the name of the schema containing TABLE_NAME
TABLE_NAME	VARCHAR	the name of the table or view
COLUMN_NAME	VARCHAR	the name of the column of the specified table or view
DATA_TYPE	VARCHAR	the data type of the column
SQL_DATA_TYPE_NUM	SMALLINT	ODBC compliant data type number
CHAR_MAX_LENGTH	INTEGER	maximum length for a character data type column; for others NULL

NUMERIC_PRECISION	INTEGER	the number of digits of mantissa precision of the column, if DATA_TYPE is approximate numeric data type, NUMERIC_PREC_RADIX indicates the units of measurement; for other numeric types contains the total number of decimal digits allowed in the column; for character data types NULL
NUMERIC_PREC_RADIX	SMALLINT	the radix of numeric precision if DATA_TYPE is one of the approximate numeric data types; otherwise NULL
NUMERIC_SCALE	SMALLINT	total number of significant digits to the right of the decimal point; for INTEGER and SMALLINT 0; for others NULL
NULLABLE	CHAR	if column is known to be not nullable 'NO'; otherwise 'YES'
NULLABLE_ODBC	SMALLINT	ODBC, if column is known to be not nullable '0'; otherwise '1'
REMARKS	LONG VARCHAR	reserved for future use

SERVER_INFO

The SERVER_INFO system view provides attributes of the current database system or server.

Column name	Data type	Description
SERVER_ATTRIBUTE	VARCHAR	identifies an attribute of the server
ATTRIBUTE_VALUE	VARCHAR	the value of the attribute

TABLES

The TABLES system view identifies the tables accessible to the current user.

Column name	Data type	Description
TABLE_CATALOG	VARCHAR	reserved for future use
TABLE_SCHEMA	VARCHAR	the name of the schema containing TABLE_NAME
TABLE_NAME	VARCHAR	the name of the table or view
TABLE_TYPE	VARCHAR	the type of the table
REMARKS	LONG VARCHAR	reserved for future use

System Tables

SQL_LANGUAGES

The SQL_LANGUAGES system table lists the SQL standards and SQL dialects which are supported.

Column name	Data type	Description
SOURCE	VARCHAR	the organization that defined this specific SQL version
SOURCE_YEAR	VARCHAR	the year the relevant standard was approved
CONFORMANCE	VARCHAR	the conformance level at which conformance to the relevant standard
INTEGRITY	VARCHAR	indicates whether the Integrity Enhancement Feature is supported
IMPLEMENTATION	VARCHAR	identifies uniquely the vendor's SQL language; NULL if SOURCE is 'ISO'
BINDING_STYLE	VARCHAR	the binding style 'DIRECT', *EMBED' or 'MODULE'
PROGRAMMING_LANG	VARCHAR	the host language used

SYS_ATTAUTH

Column name	Data type	Description
REL_ID	INTEGER	table id
UR_ID	INTEGER	user or role id
ATTR_ID	INTEGER	column id
PRIV	INTEGER	privilege info
GRANT_ID	INTEGER	grantor id
GRANT_TIM	TIMESTAMP	grant time

SYS_CARDINAL

Column name	Data type	Description
REL_ID	INTEGER	the relation id as in SYS_TABLES
CARDIN	INTEGER	the number of rows in the table
SIZE	INTEGER	the size of the data in the table
LAST_UPD	TIMESTAMP	the timestamp of the last update in the table

SYS_COLUMNS

Column name	Data type	Description
ID	INTEGER	unique column identifier
REL_ID	INTEGER	the relation id as in SYS_TABLES
COLUMN_NAME	VARCHAR	the name of the column
COLUMN_NUMBER	INTEGER	the number of the column in the table (in creation order)
DATA_TYPE	VARCHAR	the data type of the column
SQL_DATA_TYPE_NUM	SMALLINT	ODBC compliant data type number

DATA_TYPE_NUMBER	INTEGER	internal data type number
CHAR_MAX_LENGTH	INTEGER	maximum length for a CHAR field
NUMERIC_PRECISION	INTEGER	numeric precision
NUMERIC_PREC_RADIX	SMALLINT	numeric precision radix
NUMERIC_SCALE	SMALLINT	numeric scale
NULLABLE	CHAR	are NULL values allowed (Yes, No)
NULLABLE_ODBC	SMALLINT	ODBC, are NULL values allowed (1,0)
FORMAT	VARCHAR	reserved for future use
DEFAULT_VAL	VARBINARY	reserved for future use
ATTR_TYPE	INTEGER	user defined (0) or internal (>0)
REMARKS	LONG VARCHAR	reserved for future use

SYS_EVENTS

Column name	Data type	Description
ID	INTEGER	unique event identifier
EVENT_NAME	VARCHAR	the name of the event
EVENT_PARAMCOUNT	INTEGER	number of parameters
EVENT_PARAMTYPES	LONG VARBINARY	types of parameters
EVENT_TEXT	VARCHAR	the body of the event
EVENT_SCHEMA	VARCHAR	the owner of the event
CREATIME	TIMESTAMP	creation time
TYPE	INTEGER	reserved for future use

SYS_FORKEYPARTS

Column name	Data type	Description
ID	INTEGER	foreign key identifier
KEYP_NO	INTEGER	keypart number
ATTR_NO	INTEGER	column number
ATTR_ID	INTEGER	column identifier
ATTR_TYPE	INTEGER	column type
CONST_VALUE	VARBINARY	possible internal constant value; otherwise NULL

SYS_FORKEYS

Column name	Data type	Description
ID	INTEGER	foreign key identifier
REF_REL_ID	INTEGER	referenced table identifier
CREATE_REL_ID	INTEGER	creator table identifier
REF_KEY_ID	INTEGER	referenced key identifier
REF_TYPE	INTEGER	reference type
KEY_SCHEMA	VARCHAR	creator name
KEY_NREF	INTEGER	number of referenced key parts

SYS_INFO

Column name	Data type	Description
PROPERTY	VARCHAR	the name of the property
VALUE_STR	VARCHAR	value as a string
VALUE_INT	INTEGER	value as an integer

SYS_KEYPARTS

Column name	Data type	Description
ID	INTEGER	unique key identifier
REL_ID	INTEGER	the relation id as in SYS_TABLES
KEYP_NO	INTEGER	keypart identifier
ATTR_ID	INTEGER	column identifier
ATTR_NO	INTEGER	the number of the column in the table (in creation order)
ATTR_TYPE	INTEGER	the type of the column
CONST_VALUE	VARBINARY	constant value or NULL
ASCENDING	CHAR	is the key ascending (Yes) or descending (No)

SYS_KEYS

Column name	Data type	Description
ID	INTEGER	unique key identifier
REL_ID	INTEGER	the relation id as in SYS_TABLES
KEY_NAME	VARCHAR	the name of the key
KEY_UNIQUE	CHAR	is the key unique (Yes, No)
KEY_NONUNIQUE_ODBC	SMALLINT	ODBC, is the key NOT unique (1, 0)
KEY_CLUSTERING	CHAR	is the key a clustering key (Yes, No)
KEY_PRIMARY	CHAR	is the key a primary key (Yes, No)
KEY_PREJOINED	CHAR	reserved for future use
KEY_SCHEMA	VARCHAR	the owner of the key

KEY_NREF	INTEGER	internal system specific information

SYS_PROCEDURES

Column name	Data type	Description
ID	INTEGER	unique procedure identifier
PROCEDURE_NAME	VARCHAR	procedure name
PROCEDURE_TEXT	LONG VARCHAR	procedure body
PROCEDURE_BIN	LONG VARBINARY	compiled form of the procedure
PROCEDURE_SCHEMA	VARCHAR	the owner
CREATIME	TIMESTAMP	creation time
TYPE	INTEGER	reserved for future use

SYS_RELAUTH

Column name	Data type	Description
REL_ID	INTEGER	relation id
UR_ID	INTEGER	user or role id
PRIV	INTEGER	privilege info
GRANT_ID	INTEGER	grantor id
GRANT_TIM	TIMESTAMP	grant time
GRANT_OPT	CHAR	grant option info

SYS_SEQUENCES

Column name	Data type	Description
SEQUENCE_NAME	VARCHAR	sequence name
ID	INTEGER	unique id
DENSE	CHAR	is the sequence dense or sparse

SEQUENCE_SCHEMA	VARCHAR	the schema name
CREATIME	TIMESTAMP	creation time

SYS_SYNONYM

Column name	Data type	Description
TARGET_ID	INTEGER	reserved for future use
SYNON	INTEGER	reserved for future use

SYS_TABLEMODES

Column name	Data type	Description
ID	INTEGER	relation id
MODE	VARCHAR	special mode info
MODIFY_TIME	TIMESTAMP	last modify time
MODIFY_USER	VARCHAR	last user that modified

SYS_TABLES

Column name	Data type	Description
ID	INTEGER	unique table identifier
TABLE_NAME	VARCHAR	the name of the table
TABLE_TYPE	VARCHAR	the type of the table (BASE TABLE or VIEW)
TABLE_SCHEMA	VARCHAR	the owner of the table
TABLE_CATALOG	VARCHAR	reserved for future use
CREATIME	TIMESTAMP	the creation time of the table
CHECKSTRING	LONG VARCHAR	possible check option defined for the table
REMARKS	LONG VARCHAR	reserved for future use

SYS_TYPES

Column name	Data type	Description
TYPE_NAME	VARCHAR	the name of the data type
DATA_TYPE	SMALLINT	ODBC, data type number
PRECISION	INTEGER	ODBC, the precision of the data type
LITERAL_PREFIX	VARCHAR	ODBC, possible prefix for literal values
LITERAL_SUFFIX	VARCHAR	ODBC, possible suffix for literal values
CREATE_PARAMS	VARCHAR	ODBC, the parameters needed to create a column of the data type
NULLABLE	SMALLINT	ODBC, can the data type contain NULL values
CASE_SENSITIVE	SMALLINT	ODBC, is the data type case sensitive
SEARCHABLE	SMALLINT	ODBC, the supported search operations
UNSIGNED_ATTRIBUTE	SMALLINT	ODBC, is the data type unsigned
MONEY	SMALLINT	ODBC, whether the data is a money data type
AUTO_INCREMENT	SMALLINT	ODBC, whether the data type is autoincrementing
LOCAL_TYPE_NAME	VARCHAR	ODBC, has the data type another implementation defined name
MINIMUM_SCALE	SMALLINT	ODBC, the minimum scale of the data type
MAXIMUM_SCALE	SMALLINT	ODBC, the maximum scale of the data type

SYS_UROLE, SYS_USERS

These tables are for the system's internal use only.

SYS_VIEWS

Column name	Data type	Description
V_ID	INTEGER	unique identifier for this view
TEXT	LONG VARCHAR	view definition
CHECKSTRING	LONG VARCHAR	possible CHECK OPTION defined for the view
REMARKS	LONG VARCHAR	reserved for future use

F

SOLID SQL API Reserved Words

The following words are reserved in several SQL standards: ODBC 2.1, X/Open and SQL Access Group SQL CAE specification, Database Language - SQL: ANSI X3H2 (SQL-92). Some words are used by SOLID SQL. Applications should avoid using any of these keywords for other purposes. The following table contains also potential reserved words; these markings are enclosed in parenthesis.

Reserved word	ODBC	X/Open SQL	ANSI SQL2	SOLID SQL
ABSOLUTE	•		•	
ACTION			•	
ADA	•			
ADD	•	•	•	•
ADMIN				•
AFTER			(•)	•
ALIAS			(•)	
ALL	•	•	•	•
ALLOCATE	•	•	•	
ALTER	•	•	•	•
AND	•	•	•	•
ANY	•	•	•	•
ARE	•		•	

AS	•	•	•	•
ASC	•	•	•	•
ASSERTION	•		•	
ASYNC			(•)	•
AT	•		•	
AUTHORIZATION	•		•	•
AVG	•	•	•	
BEFORE			(•)	•
BEGIN	•	•	•	•
BETWEEN	•	•	•	•
BINARY				•
BIT	•		•	
BIT_LENGTH	•		•	
BOOLEAN			(•)	
BOTH			•	
BREADTH			(•)	
BY	•	•	•	•
CALL			(•)	•
CASCADE	•	•	•	•
CASCADED	•		•	
CASE	•		•	
CAST	•		•	
CATALOG	•		•	
CHAR	•	•	•	•
CHAR_LENGTH	•		•	

CHARACTER	•	•	•	
CHARACTER_LENGTH	•		•	
CHECK	•	•	•	•
CLOSE	•	•	•	•
COALESCE	•		•	
COBOL	•			
COLLATE	•		•	
COLLATION	•		•	
COLUMN	•		•	•
COMMIT	•	•	•	•
COMMITTED				•
COMPLETION			(•)	
CONNECT	•	•	•	
CONNECTION	•	•	•	
CONSTRAINT	•		•	
CONSTRAINTS	•		•	
CONTINUE	•	•	•	
CONVERT	•		•	
CORRESPONDING	•		•	
COUNT	•	•	•	
CREATE	•	•	•	•
CROSS			•	
CURRENT	•	•		•
CURRENT_DATE	•		•	
CURRENT_TIME	•		•	

CURRENT_TIMESTAMP	•		•	
CURRENT_USER			•	
CURSOR	•	•	•	•
CYCLE			(•)	
DATA			(•)	
DATE	•		•	•
DAY	•		•	
DEALLOCATE	•	•	•	
DEC	•	•	•	
DECIMAL	•	•	•	•
DECLARE	•	•	•	
DEFAULT		•	•	•
DEFERRABLE	•		•	
DEFERRED	•		•	
DELETE	•	•	•	•
DEPTH			(•)	
DESC	•	•	•	•
DESCRIBE	•	•	•	
DESCRIPTOR	•	•	•	
DIAGNOSTICS	•	•	•	
DICTIONARY	•		(•)	
DISCONNECT	•	•	•	
DISPLACEMENT	•			
DISTINCT	•	•	•	•
DOMAIN	•		•	

DOUBLE	•	•	•	•
DROP	•	•	•	•
EACH			(•)	
ELSE	•		•	•
ELSEIF			(•)	•
END	•	•	•	•
END-EXEC	•		•	
EQUALS			(•)	
ESCAPE	•		•	•
EVENT				•
EXCEPT	•		•	•
EXCEPTION	•	•	•	
EXEC	•	•	•	•
EXECUTE	•	•	•	•
EXISTS	•	•	•	•
EXPLAIN				•
EXTERNAL	•		•	
EXTRACT	•		•	
FALSE	•		•	
FETCH	•	•	•	•
FIRST	•		•	
FLOAT	•	•	•	•
FOR	•	•	•	•
FOREIGN	•	•	•	•
FORTRAN	•			

FOUND	•	•	•	
FROM	•	•	•	•
FULL	•		•	
GENERAL			(•)	
GET	•	•	•	
GLOBAL	•		•	
GO	•		•	
GOTO	•	•	•	
GRANT	•	•	•	•
GROUP	•	•	•	•
HAVING	•	•	•	•
HOUR	•		•	
IDENTIFIED				•
IDENTITY	•		•	
IF			(•)	•
IGNORE	•		(•)	
IMMEDIATE	•	•	•	
IN	•	•	•	•
INCLUDE	•	•		
INDEX	•	•		•
INDICATOR	•		•	
INITIALLY	•		•	
INNER	•		•	•
INPUT	•		•	
INSENSITIVE	•		•	

INSERT	•	•	•	•
INT		•	•	•
INTEGER	•	•	•	•
INTERSECT	•		•	•
INTERVAL	•		•	
INTO	•	•	•	•
IS	•	•	•	•
ISOLATION	•		•	•
JOIN	•		•	•
KEY	•	•	•	•
LANGUAGE	•		•	
LAST	•		•	
LEADING			•	
LEAVE			(•)	•
LEFT	•		•	•
LESS			(•)	
LEVEL	•		•	•
LIKE	•	•	•	•
LIMIT			(•)	
LOCAL	•		•	
LOCK				•
LONG				•
LOOP			(•)	•
LOWER	•		•	
MAINMEMORY				•

MATCH	•		•	
MAX	•	•	•	
MIN	•	•	•	
MINUTE	•		•	
MODIFY			(•)	•
MODULE	•		•	
MONTH	•		•	
MUMPS	•			
NAMES	•		•	
NATIONAL	•		•	
NATURAL			•	
NCHAR	•		•	
NEW			(•)	•
NEXT	•		•	•
NO			•	
NONE	•		(•)	
NOT	•	•	•	•
NULL	•	•	•	•
NULLIF	•		•	
NUMERIC	•	•	•	•
OBJECT			(•)	
OCTET_LENGTH	•		•	
OF	•	•	•	•
OFF	•		(•)	
OID			(•)	

OLD			(•)	
ON	•	•	•	•
ONLY	•		•	•
OPEN	•	•	•	
OPERATION			(•)	
OPERATORS			(•)	
OPTIMISTIC				•
OPTION	•		•	•
OR	•	•	•	•
ORDER	•	•	•	•
OTHERS			(•)	
OUTER	•		•	•
OUTPUT	•		•	
OVERLAPS	•		•	
PARAMETERS			(•)	
PARTIAL	•		•	
PASCAL	•			
PENDANT			(•)	
PESSIMISTIC				•
PLAN				•
PLI	•			
POSITION	•		•	
POST				•
PRECISION	•	•	•	•
PREORDER			(•)	

PREPARE	•	•	•	•
PRESERVE	•		•	
PRIMARY	•	•	•	•
PRIOR	•		•	
PRIVATE			(•)	
PRIVILEGES	•		•	•
PROCEDURE	•		•	•
PROTECTED			(•)	
PUBLIC	•	•	•	•
READ			•	•
REAL		•	•	•
RECURSIVE			(•)	
REF			(•)	
REFERENCES		•	•	•
REFERENCING			(•)	•
REGISTER				•
RELATIVE			•	
RENAME				•
REPEATABLE				•
REPLACE			(•)	
RESIGNAL			(•)	
RESTRICT	•	•	•	•
RETURN			(•)	•
RETURNS			(•)	•
REVOKE	•	•	•	•

RIGHT	•		•	
ROLE			(•)	•
ROLLBACK	•	•	•	•
ROUTINE			(•)	
ROW			(•)	
ROWID				•
ROWNUM				•
ROWVER				•
ROWS	•		•	
SAVEPOINT			(•)	•
SCHEMA	•		•	•
SCROLL	•		•	
SEARCH			(•)	
SECOND	•		•	
SECTION	•	•	•	
SELECT	•	•	•	•
SENSITIVE			(•)	
SEQUENCE	•		(•)	•
SERIALIZABLE				•
SESSION			•	
SESSION_USER			•	
SET	•	•	•	•
SIGNAL			(•)	
SIMILAR			(•)	
SIZE	•		•	

SMALLINT	•	•	•	•
SOME	•		•	•
SPACE				
SQL	•	•	•	•
SQLCA	•	•		
SQLCODE	•		•	
SQLERROR	•	•	•	•
SQLEXCEPTION			(•)	
SQLSTATE	•		•	
SQLWARNING	•	•	(•)	
START				•
STRUCTURE			(•)	
SUBSTRING	•		•	
SUM	•	•	•	
SYSTEM	•			
SYSTEM_USER			•	
TABLE	•	•	•	•
TEMPORARY	•		•	
TEST			(•)	
THEN	•		•	•
THERE			(•)	
TIME	•		•	•
TIMEOUT				•
TIMESTAMP	•		•	•
TIMEZONE_HOUR	•		•	

TIMEZONE_MINUTE	•		•	
TINYINT				•
TO	•	•	•	•
TRAILING			•	
TRANSACTION	•		•	•
TRANSLATE	•		•	
TRANSLATION	•		•	
TRIGGER			(•)	•
TRIM			•	
TRUE	•		•	
TYPE			(•)	
UNDER			(•)	
UNION	•	•	•	•
UNIQUE	•	•	•	•
UNKNOWN	•		•	
UNREGISTER				•
UPDATE	•	•	•	•
UPPER	•		•	
USAGE	•		•	
USER	•	•	•	•
USING	•	•	•	•
VALUE	•	•	•	
VALUES	•	•	•	•
VARBINARY				•
VARCHAR	•	•	•	•

VARIABLE			(•)	
VARYING	•	•	•	
VIEW	•	•	•	•
VIRTUAL			(•)	
VISIBLE			(•)	
WAIT			(•)	•
WHEN	•		•	•
WHENEVER	•	•	•	
WHERE	•	•	•	•
WHILE			(•)	•
WITH	•	•	•	•
WITHOUT			(•)	
WORK	•	•	•	•
WRITE			•	•
WCHAR				•
WVARCHAR				•
YEAR	•		•	
ZONE			•	

G

SOLID *Embedded Engine* Command Line Options

General Options

Option	Description	Examples
-c<dir>	Changes working directory.	
-f	Starts server in foreground.	
-h	Displays help.	
-m	Monitors users' messages and SQL statements.	
-n<name>	Sets server name.	
-s{start|install|remove}, name, fullexepath, [autostart]	The Windows NT version of SOLID *Embedded Engine* is by default an icon exe version. Using the option -sstart, SOLID Embedded Engine can be started as a service executable and started and stopped from the service manager. If SOLID *Embedded Engine* is started without the-sstart option, it starts as an icon exe like the w16 and w95 versions. The service version of SOLID *Embedded Engine* cannot interact with the display and cannot create a new database. The service version writes warning and error messages also to the NT event log. SOLID *Embedded Engine* can also install and remove services using this command line option.	SOLID.EXE -s"install,SOLID, D:\SOLID\SOLID.EXE -sstart -cD:\SOLID" SOLID.EXE -s"install,SOLID, D:\SOLID\SOLID.EXE -sstart -Cd:\SOLID,autostart" SOLID.EXE -s"remove,SOLID"

-U<username>	See option -x execute or -x exit. If used without the -x option, specifies the username for the database being created.	
-P<password>	See option -x execute or -x exit. If used without the -x option, specifies the given password for the database being created.	
-x autoconvert	Converts database format to version 3.0 and starts server process	
-x convert	Converts database format to version 3.0 and exists	
-x execute:<input file>	Prompts for the database administrator's user name and password, creates a new database, executes SQL statements from a file, and exists. The options -U and -P can be used to give the database the administrator's user name and password.	solid.exe -x execute:init.sql solid.exe -x execute:init.sql -Udba -Pdba
-x exit	Prompts for the database administrator's user name and password, creates a new database, and exists. Options -U and -P can be used to give the database administrator's user name and password.	solid.exe -x exit solid.exe -x exit -Udba -Pdba
-x forcerecovery	Does a forced roll-forward recovery.	
-x hide	Hides server icon.	
-x ignoreerrors	Ignores index errors.	
-x testblocks	Tests database blocks.	
-x testindex	Tests database index.	
-?	Help = Usage.	

Glossary

This glossary gives you a description of the terminology used in SOLID documentation.

Binary Large Object (BLOb)

A BLOb is a large block of binary information such as a picture, video clip, sound excerpt, or a formatted text document. BLObs can be saved to and retrieved from SOLID *Embedded Engine*.

Checkpoint

Checkpoints are used to store a consistent state of the database quickly onto the disk. After a system crash, the database will start recovering transactions from the latest checkpoint. The more frequently checkpoints are made, the fewer transactions need to be recovered from the log file.

Client/server computing

Client/server computing divides a large piece of software into modules that need not all be executed within the same memory space nor on the same processor. The calling module becomes the ‘client’ that requests services, and the called module becomes the ‘server’ that provides services. Client and server processes exchange information by sending messages through a computer network. They may run on different hardware and software platforms as appropriate for their special functions.

Two basic client/server architecture types are called two-tier and three-tier application architectures.

Communication protocol

A communication protocol is a set of rules and conventions used in the communication between servers and clients. The server and client have to use the same communication protocol in order to establish a connection.

Database administrator

The database administrator is a person responsible for tasks such as:

- managing users, tables, and indices
- backing up data
- allocating disk space for the database files

Database management system (DBMS)

A DBMS is a system that stores information in and retrieves information from a database. A DBMS typically consists of a database server, administration utilities, an application interface, and development tools.

Database procedures (Stored procedures)

Database procedures allow programmers to split the application logic between the client and the server. These procedures are stored in the database, and they accept parameters in the activation call from the client application. This arrangement is beneficial especially in the case of heavy updates that first require extensive queries and that can be initiated with a small amount of parameter information. In these cases, the network traffic is significantly reduced, and much better performance can be achieved.

Event Alerts

Events are objects with a name and parameters. Event alerts are used to signal an event in the database. The signal is sent from an application using the `POST EVENT` command. The signal is received by one or more client applications waiting for the event. The use of event alerts removes resource consuming database polling from applications.

Log file (Transaction log)

This file holds a log of all committed operations executed by the database server. If a system crash occurs, the database server uses this log to recover all data inserted or modified after the latest checkpoint.

Network name

The network name of a server consists of a communication protocol and a server name. This combination identifies the server in the network.

SOLID Clients support Logical Data Source Names. These names can be used to give a database a descriptive name. This name is mapped to a network name using either parame-

ter settings in the clients `solid.ini` file or in Windows operating systems' registry settings.

Open Database Connectivity (ODBC)

ODBC is a programming interface standard for SQL database programs. SOLID *Embedded Engine* offers a native ODBC programming interface.

Relational database management system (RDBMS)

SOLID *Embedded Engine* is an RDBMS, which stores and retrieves information that is organized into two-dimensional tables. This name derives from the relational theory that formalizes the data manipulation requests as set operations and allows mathematical analysis of these sets. RDBMSs typically support the SQL language for data manipulation requests.

SQL Access Group's Call Level Interface (SAG CLI)

SAG CLI is a programming interface standard that defines the functions that are used to submit dynamic SQL clauses to a database server for execution. The ODBC interface is also based on SAG CLI. The SOLID *SQL API* conforms to the SAG CLI standard.

Schema

All tables are contained in a higher level construct called schema. It is a place where tables and related objects are gathered together under one qualifying name. For each schema there are zero or more tables, and for each table, there is exactly one schema to which it belongs. The relationship between a schema and its tables is similar to that of an operating system directory and the files contained within that directory.

Table schemas allow several logical databases to reside in the same physical database. A typical use could be to have a similar table structure for each customer in the database of an accounting firm. All the data would still be stored in a single physical database, which allows sharing the common parameter information.

SOLID directory

The default directory for storing SOLID DBMS database files. This is the server program's working directory.

Structured Query Language (SQL)

SQL is a standardized query language designed for handling database requests and administration. The SQL syntax used in SOLID *Embedded Engine* is based on the ANSI X3H2-1989 Level 2 standard including important ANSI X3H2-1992 (SQL2) extensions. For a more formal definition of the syntax, refer to *Appendix D SOLID SQL Syntax* of **SOLID Administrator Guide.**

Three-tier client/server architecture model

Compared to the two-tier architecture the three-tier architecture has an additional layer or layers of application servers. This allows splitting the application logic between client processes to a specialized application server process handling the resources management, other I/O, or calculation intensive tasks.

Instead of sending small SQL statements the client application sends whole procedures for the application server to be processed. This reduces the number of messages thus minimizing the network load. The application logic is often more easily managed because several applications use centrally maintained procedures.

Two-tier client/server architecture model

Generally, the two-tier architecture refers to a client/server system, where a client application containing all the business logic is running on a workstation and a database server is taking care of data management.

Index

A

arguments for timed commands, 3-7
automating administrative tasks, 3-6

B

backup directory, 7-3
backups
  automating, 3-6
  failed, 3-2
  making manually, 3-1
  online, 3-2

C

cache
  database, 8-4
changing database location, 3-5
checkpoints, 3-4
  automatic daemon, 3-4
  automating, 3-6
  erasing automatically, 3-5
  frequency, 3-4
closing SOLID Server, 2-4
cluster, 5-8
columns
  adding to a table, 5-6
  deleting from a table, 5-6
committing work
  after altering table, 5-6, 5-8
  after altering users and roles, 5-5
communication
  between client and server, 6-1
  selecting a protocol, 6-4
  tracing problems, 9-8
communication protocols, 6-4
  DECnet, 6-7
  IPX/SPX, 6-8
  Named Pipes, 6-7
  NetBIOS, 6-6
  selecting, 6-4
  Shared Memory, 6-4
  summary, 6-9
  TCP/IP, 6-5
  UNIX Pipes, 6-5
concatenated indexes, 8-2
configuration file, B-1
connecting to SOLID *Enbedded Engine*, 2-3
control file
  SOLID *SpeedLoader*, 4-2
control file syntax
  SOLID *SpeedLoader*, 4-4
creating reports
  automating, 3-6

D

data source name, 6-4
Data Sources, 6-11
  defining in solid.ini, 6-11
database
  changing location, 3-5
  closing, 3-5
    automating, 3-6
  creating, 2-2
  opening
    automating, 3-6
  recovery, 3-4
  several databases on one computer, 3-6

database. See also index file, 7-2
DECnet, 6-7
documentation
  electronic, xi

E

executing system commands
  automating, 3-6
EXPLAIN PLAN statement, 9-3
external sorting, 8-5

F

FileSpec, 7-2

I

import file
  SOLID *SpeedLoader*, 4-2
index file
  changing block size, 7-6
  location, 7-2
  maximum size, 7-2
  splitting to multiple disks, 7-2
indexes, 8-2
  creating, 5-7
  creating a unique index, 5-7
  deleting, 5-7
  foreign key, 5-8
  managing, 5-7
ini file
  SOLID *SpeedLoader*, 4-3
installing SOLID *Embedded Engine*, 2-1
IPX/SPX, 6-8

L

listen name, 6-1, 6-3
log file, 3-4
  SOLID *SpeedLoader*, 4-3
logon. See connecting to SOLID *Embedded Engine*, 2-3

M

multi-column indexes, 8-2

N

Named Pipes, 6-7
NetBIOS, 6-6
network names, 6-1, 6-3
  activating modifications, 6-3
  adding, 6-2
  clients, 6-3
  DECnet, 6-8
  modifying, 6-2
  Named Pipes, 6-7
  NetBIOS, 6-6
  removing, 6-3
  Shared Memory, 6-4
  TCP/IP, 6-5
  UNIX Pipes, 6-6
network names IPX/SPX, 6-8
Network trace facility, 9-9
non-graphical user interfaces
  creating new database, 2-2

O

ODBC
  data source name, 6-4

P

parameters, B-1
  BackupCopyLog, 3-4
  BackupDeleteLog, 3-4
  BackupDirectory, 7-3
  CacheSize, 7-4
  CheckpointInterval, 3-5
  Connect, 7-2
  default settings, 7-1
  FileNameTemplate, 7-4
  FileSpec_]1...N], 7-2
  Info, 7-5
  Listen, 7-2
  managing, 7-5
  Threads, 7-4
  TmpFile, 7-4
  Trace, 7-5
  TraceFile, 7-5
  with constant values, 7-6
passwords

criteria, 2-2
entering, 5-3
performance
indexes, 8-2
Ping facility, 9-10

R

recovery, 3-4
referential integrity, 5-8
running several servers, 3-6

S

server names. See network names, 6-1
Shared Memory, 6-4
shutting down SOLID *Embedded Engine*, 2-4
automating, 3-6
SOLDD, 4-12
options, 4-12
SOLEXP, 4-11
SOLID *Remote Control*, 1-5
SOLID *Remote Control* (Teletype)
commands, 4-14
options, 4-13
starting, 4-13
SOLID *Embedded Engine*
background, 1-1
closing, 2-4
connecting to, 2-3
features, 1-1
installing, 2-1
introduction, ix
starting, 2-1
SOLID *SpeedLoader*
control file, 4-2
control file syntax, 4-4
import file, 4-2
ini file, 4-3
log file, 4-3
SOLID *SQL Editor*, 1-5
SOLID *SQL Editor* (Teletype), 4-16
commands, 4-17
options, 4-16
SOLLOAD, 4-4
options, 4-4
sorting, 8-5
SQL Info facility, 9-2
SQL scripts, 5-2
sample.sql, 5-5
users.sql, 5-2
SQL statements, 5-1
examples for administering indexes, 5-7
examples for managing indexes, 5-7
examples for managing tables, 5-5
examples for managing users and roles, 5-3
tuning, 8-1
starting SOLID *Remote Control* (Teletype), 4-13
starting SOLID *Embedded Engine*, 2-1

T

tables
adding columns to, 5-6
committing work after altering, 5-6, 5-8
creating, 5-6
deleting columns from, 5-6
managing, 5-5
removing, 5-6
TCP/IP, 6-5
throwing out users
automating, 3-6
timed commands, 3-6
tracing communication, 9-8
tuning SQL statements, 8-1

U

UNIX Pipes, 6-5
user and roles
committing work after altering, 5-5
user names
reserved names, 5-2
user privileges, 5-2
granting, 5-4
granting administrator privileges, 5-5
revoking, 5-4
user roles, 5-2
administrator role, 5-3, 5-5
creating, 5-4
deleting, 5-4
giving a user a role, 5-4

granting privileges to, 5-4
reserved role names, 5-2
revoking privileges from, 5-4
revoking the role of a user, 5-5
system console role, 5-3
usernames
criteria, 2-2
default, 2-2
users
creating, 5-3
deleting, 5-3
throwing out, 3-6

V

viewing Message Log, 2-4

W

Windows registry
data sources, 6-12