solid.

www.solidDB.com

solidDB
High Availability User Guide

Version 6.0 | June 2007

solidDB 6.0

solidDB High Availability User Guide

Copyright © 2007 Solid Information Technology Ltd.
Document number: SHS-6.00
Product version: 06.00.1018
Date: 15.06.2007

All rights reserved. No portion of this product may be used in any way except as expressly authorized in writing by Solid Information Technology.

Solid logo with the text "SOLID" or "Solid", or "solidDB" is a registered trademark of Solid Information Technology Inc.

Solid AcceleratorLib™, Solid Availability™, Solid Bonsai Tree™, Solid BoostEngine™, Solid CarrierGrade Option™, Solid Database Engine™, Solid Diskless™, Solid EmbeddedEngine™, Solid FlowControl™, Solid FlowEngine™, Solid High Availability™, Solid HotStandby™, Solid Information Technology™, Solid Intelligent Transaction™, Solid Remote Control™, Solid SmartFlow™, Solid SQL Editor™, Solid SynchroNet™, and Built Solid™ are trademarks of Solid Information Technology Inc. All other products, services, companies and publications are trademarks or registered trademarks of their respective owners.

This product is protected by U.S. patents 6144941, 6970876, and 6978386.

This product contains the skeleton output parser for bison ("Bison"). Copyright (c) 1984, 1989, 1990 Bob Corbett and Richard Stallman.

> Bison is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 1, or (at your option) any later version. Bison is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
>
> For a period of three (3) years from the date of this license, Solid Information Technology Inc. will provide you, the licensee, with a copy of the Bison source code upon receipt of your written request and the payment of Solid's reasonable costs for providing such copy.

This product contains lexical analyzer Flex. Copyright (c) 1990 The Regents of the University of California. All rights reserved.

> This code is derived from software contributed to Berkeley by Vern Paxson. The United States Government has rights in this work pursuant to contract no. DE-AC03-76SF00098 between the United States Department of Energy and the University of California. Redistribution and use in source and binary forms are permitted provided that: (1) source distributions retain this entire copyright notice and comment, and (2) distributions including binaries display the following acknowledgement: "This product includes software developed by the University of California, Berkeley and its contributors'" in the documentation or other materials provided with the distribution and in all advertising materials mentioning features or use of this software. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANT-ABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

This product contains zlib general purpose compression library version 1.1.4, March 11th, 2002. Copyright (C) 1995-2002 Jean-loup Gailly and Mark Adler.

> This software is provided "as-is", without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software. Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following

restrictions: 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required. 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software. 3. This notice may not be removed or altered from any source distribution.

This product is assigned the U.S. Export Control Classification Number ECCN=5D992a.

Table of Contents

1 Welcome 1
1.1 About This Guide 1
1.1.1 Organization 1
1.1.2 Audience 2
1.2 Conventions 2
1.2.1 About solidDB 2
1.2.2 Typographic Conventions 2
1.2.3 Syntax Notation 3
1.3 Solid Documentation 4
2 Introducing the solidDB CarrierGrade Option 7
2.1 How HotStandby Works 7
2.1.1 The Transaction Log and HotStandby 7
2.1.2 Server HotStandby States 9
2.1.3 Server Diagrams 9
2.1.4 Combining HotStandby and SmartFlow 10
2.1.5 Switching the Secondary to Be the New Primary 11
2.1.6 Server CatchUp 12
2.1.7 The "Watchdog" Application 12
2.1.8 Replication Modes in HotStandby 13
2.2 Description of Server States 16
2.3 How Does HotStandby Affect Performance 18
2.3.1 Adaptive Durability 19
2.4 HotStandby Configuration 22
2.5 Implementing the CarrierGrade Option 22
2.5.1 HotStandby Configuration and System Design Issues 22
2.5.2 Watchdog Configuration 23
3 Getting Started with HotStandby 25
3.1 Before You Begin 25
3.2 HotStandby Demonstration 25
3.3 HotStandby Setup and Configuration Procedure 25
3.3.1 Section 1: Setup and Configuration Preparations 26
3.3.2 Section 2: Step-by-Step Procedure 26
4 Administering the CarrierGrade Option 29
4.1 What You Should Know 29
4.1.1 HotStandby and the Solid Configuration File 29
4.1.2 HotStandby and Access Rights 30
4.1.3 Solid Tools and the CarrierGrade Option 31
4.1.4 Database Migration (disk-based servers only) 31
4.1.5 Interoperability 32

4.2 Limitations with HotStandby 32
4.2.1 In-Memory Tables 32
4.3 Warnings 32
4.3.1 Network Partitions and Dual Primaries 32
4.3.2 Running Out of Space for Transaction Logs 33
4.4 Overview of Administration Tasks 34
4.5 Performing HotStandby Recovery and Maintenance 35
4.6 Switching Server States 35
4.6.1 Switchover and Failover 35
4.6.2 Performing Switchovers 35
4.6.3 Important Notes on Switching Servers 37
4.6.4 Verifying the Switch 38
4.6.5 Performing Failovers 38
4.6.6 Running the New Primary in PRIMARY ALONE State 39
4.6.7 Bringing the Secondary Server Back Online 40
4.7 Shutting Off HotStandby Operations 41
4.8 Synchronizing Primary and Secondary Servers 41
4.8.1 Catchup 42
4.8.2 Full Copy 43
4.8.3 Verifying the Copy 46
4.8.4 Using a Watchdog to Synchronize Servers 46
4.8.5 Copying a Primary Database to a Secondary Over the Network 46
4.8.6 Creating a New Database for the Secondary Server 47
4.8.7 Replacing an Existing Database on the Secondary Server 49
4.8.8 Verifying Netcopy Status 50
4.8.9 Copying a Database File from Primary Server to a Specified Directory 50
4.9 Connecting HotStandby Servers 52
4.10 Checking HotStandby Status 52
4.10.1 Displaying Switch Status Information 53
4.10.2 Displaying Connect Status Information 54
4.10.3 Displaying Communication Information 54
4.10.4 Displaying Role Start Time 55
4.11 Verifying HotStandby Server States 56
4.11.1 Server States Overview 57
4.12 Choosing Which Server to Make Primary 58
4.13 Changing a HotStandby Server to a Non-HotStandby Server 60
4.14 Special Configurations: Lower Cost vs. Higher Safety 60
4.14.1 Reducing Cost: N + 1 Spare and N + M Spares Scenarios 61
4.14.2 Increasing Reliability: 2N + 1 Spare and 2N + M SpareScenarios 61
4.14.3 How Solid HSB Supports The N+1 (N+M) and 2N+1 (2N+M) Approaches 62
5 Using HotStandby with Applications 65
5.1 Two Ways to Connect to HotStandby Servers 65

5.1.1 Transparent Connectivity 65
5.1.2 Basic Connectivity 65
5.1.3 Choosing the Connectivity Type 66
5.2 Using the Transparent Connectivity 66
5.2.1 Failure Transparency in TC 66
5.2.2 Load Balancing in TC 67
5.2.3 Syntax of the TC Info 70
5.2.4 TC Info Attribute Combinations 72
5.2.5 Handling TC Info Contradictions 73
5.2.6 Enacting Transparent Connectivity in JDBC 73
5.2.7 Programming for Connection Switch 76
5.3 Using the Basic Connectivity 80
5.3.1 Reconnecting to Primary Servers from Applications 80
5.3.2 Re-Connecting to Secondary Servers 86
5.3.3 SmartFlow Data Distribution Requirements 86
6 Configuring HotStandby 89
6.1 Configuring solidDB for HotStandby 89
6.1.1 Defining Secondary and Primary Node Configuration (Com Section) 90
6.1.2 Defining Timeouts Between Applications and Servers (Com Section) 90
6.1.3 Transaction Durability 92
6.2 Configuring HotStandby-Specific Parameters 93
6.2.1 Defining Primary and Secondary HotStandby Configuration 93
6.2.2 Setting HotStandby Server Wait Time to Help Detect Broken or Unavailable Connections 94
6.2.3 Defining a Name and Location for HotStandby Database Copy Operation 96
6.2.4 Defining Primary Server Behavior During a Secondary Failure 97
6.3 Performance Tuning 98
6.3.1 Tuning Replication Performance with Safeness and Durability Levels 98
6.3.2 Tuning Netcopy Performance (General Section) 98
6.4 Configuring Parameters for a Watchdog 99
6.4.1 Watchdog Section 100
6.5 Configuration File Examples 103
7 Monitoring HotStandby Server Pairs with a Watchdog Application 105
7.1 How the Watchdog Application Works 105
7.1.1 Failure Mode 106
7.1.2 Coding a Watchdog for Multiple Failures 107
7.2 Using the Sample Watchdog Application 107
7.3 HotStandby Failure Scenarios and Watchdog Actions 108
7.3.1 Primary is Down 108
7.3.2 Secondary is Down 110
7.3.3 Watchdog is Down 113
7.3.4 Communication Link Between Primary and Secondary Is Down 115

7.3.5 Communication Link Between Watchdog and Primary Is Down 117
7.3.6 Communication Link Between Watchdog and Secondary Is Down 119
7.3.7 Communication Links Between Watchdog and Primary, and Between Primary and Secondary, Are Down 121
7.3.8 Communication Links Between Watchdog and Secondary, and Between Primary and Secondary, Are Down 124
8 Upgrading Your Server by Using HotStandby 127
8.1 Cold and Hot Migration 127
8.2 Migration between HSB-Compatible Versions 127
8.2.1 Cold Migration 127
8.2.2 Hot Migration 127
8.3 Migration between HSB-Incompatible Versions 128
8.3.1 Preparation Steps 128
8.3.2 After the Upgrade 132
A Configuration Parameters 133
A.1 Ensuring that Primary and Secondary Parameter Values Are Coordinated 133
A.2 Determining Whether the Primary's Settings Take Precedence Over the Secondary's 135
A.3 Querying HotStandby Configuration Parameters 135
A.4 Modifying HotStandby Configuration Parameters 136
A.5 Access Mode 136
A.5.1 Access Mode Values 137
A.5.2 Saving Parameter Changes 137
A.6 Cluster Section 137
A.7 HotStandby Section 138
A.8 Watchdog Section 143
B Error Codes 149
B.1 HotStandby Errors and Status Codes 149
B.2 Solid Database Errors 158
B.3 Solid Table Errors 160
B.4 Solid Communication Errors 161
C Summary of HotStandby Administrative Commands 163
D Server State Transitions 175
D.1 HotStandby State Transition Diagram 175
E HSB System Events 181
Glossary 183
Index 187

List of Figures

2.1 HotStandby Server Scheme 9
2.2 Illustration Key 10
2.3 HotStandby with Master and Replica Server Scheme 11
2.4 HotStandby Switchover to New Primary (old Secondary) 11
2.5 Server Failover and Catchup Example 12
2.6 Synchronous HotStandby Configuration 14
2.7 Heterogeneous HotStandby Configuration with Watchdog 23
4.1 State Switch 36
4.2 Manual Full Copy Procedure 45
5.1 Master failover 87
5.2 Replica failover 88
7.1 Primary is Down Scenario and Remedy 109
7.2 Secondary is Down Scenario and Remedy 112
7.3 Watchdog is Down Scenario and Remedy 114
7.4 Broken Link Between Primary and Secondary Scenario and Remedy 116
7.5 Broken Link Between Watchdog and Primary Scenario and Remedy 118
7.6 Broken Link Between Watchdog and Secondary Scenario and Remedy 120
7.7 Broken Link Between Watchdog and Primary, and between Primary and Secondary, Scenario and Remedy 122
7.8 Broken Link between Watchdog and Secondary and between Primary and Secondary Scenario and Remedy 125
D.1 HotStandby Server State Transitions 177

List of Tables

1.1 Typographic Conventions 2
1.2 Syntax Notation Conventions 3
2.1 Description of Server States 16
4.1 Administration Tasks 34
4.2 ADMIN COMMAND 'hotstandby status' Options 53
4.3 Connect Status Return Values 54
4.4 HotStandby Server States 56
4.5 Server States 57
5.1 Choosing the Connectivity Type 66
5.2 TC Info Abbreviations 70
5.3 Possible Combinations of TC Info Attributes 72
5.4 Connect Request Errors 75
5.5 Warnings 75
5.6 Connection Switch Request 76
5.7 Communication Link Failure 77
5.8 Session State Preservation 78
5.9 HOTSTANDBY_CONNECTSTATUS Status Values 83
8.1 Hot Migration 129
A.1 Cluster Parameters 137
A.2 HotStandby Parameters 138
A.3 Watchdog Parameters 144
B.1 HotStandby Errors and Status Codes 149
B.2 Solid Database Errors 158
B.3 Solid Table Errors 160
B.4 Solid Communication Errors 161
C.1 HotStandby Commands 163
D.1 Server State Transition Table 178
E.1 HotStandby Events 181

List of Examples

5.1 Client-side INI File 72
5.2 Connect String in ODBC 72
5.3 Using Transparent Connectivity in JDBC 74
5.4 Basic Connection without Transparency 80
5.5 Sample Pseudo-Code 84
6.1 Entry in the `solid.ini` File 90
6.2 Partial `solid.ini` Files 93
7.1 Dual Primaries 107

Chapter 1. Welcome

The Solid CarrierGrade option increases the reliability of your database system, reducing downtime. The CarrierGrade option uses a "hot standby" approach, in which a second database server runs in parallel with the primary server and keeps an exact up-to-date copy of the data. If the primary database server fails, then your application can switch over to the secondary, with no loss of committed transactions, and with minimal performance impact. Switchover times can be quite fast — as short as a couple of hundred milliseconds, depending upon the characteristics of your hardware and software environment.

1.1 About This Guide

This guide is intended only for those who have purchased the CarrierGrade option as part of their solidDB package. It contains information specific to the CarrierGrade option only.

For general administration and maintenance information on Solid databases, see *solidDB Administration Guide*.

1.1.1 Organization

This guide includes the following information:

- Chapter 2, *Introducing the solidDB CarrierGrade Option*, describes the concepts, components, and physical configuration options of the CarrierGrade option.
- Chapter 3, *Getting Started with HotStandby*, provides step-by-step instructions to start two Solid HotStandby servers (a Primary server and a Secondary server).
- Chapter 4, *Administering the CarrierGrade Option*, describes typical operations, such as synchronizing Primary and Secondary servers, and using administrative commands to control how the servers interact.
- Chapter 5, *Using HotStandby with Applications* describes guidelines for coding applications that use the CarrierGrade option.
- Chapter 6, *Configuring HotStandby*, describes the parameter settings in the Solid Configuration file (`solid.ini`) for implementing and maintaining HotStandby.
- Chapter 7, *Monitoring HotStandby Server Pairs with a Watchdog Application*, describes how a separate "watchdog" application monitors HotStandby server pairs. This chapter also shows various possible failure scenarios (e.g. the Primary fails, or the Secondary fails) and what actions the watchdog program (or a human administrator) should take to recover from them.

- Chapter 8, *Upgrading Your Server by Using HotStandby*, describes how you can use the HotStandby functionality to upgrade your server without temporarily shutting down your entire system.
- Appendix A, *Configuration Parameters*, provides a reference of `solid.ini` configuration parameters that are used with the Solid CarrierGrade option.
- Appendix B, *Error Codes*, provides detailed information about error messages related to the CarrierGrade option.
- Appendix C, *Summary of HotStandby Administrative Commands*, provides a reference of commands used to administer the Solid CarrierGrade option.
- Appendix D, *Server State Transitions*, lists possible server states (Primary, Secondary, etc.) and shows the ways that HotStandby commands affect the server state.
- Appendix E, *HSB System Events*, lists the system events related to HotStandby.

1.1.2 Audience

This guide assumes the reader has general DBMS knowledge, and familiarity with SQL and solidDB.

1.2 Conventions

1.2.1 About solidDB

solidDB from Solid Information Technology (Solid) represents a family of advanced database solutions for mission-critical applications.

This documentation assumes that all options of solidDB are licensed for use. In some cases, however, a customer may choose not to license certain options. These include in-memory engine, disk-based engine, CarrierGrade Option (also known as "HotStandby" in previous releases), and SmartFlow Option. Please refer to your organization's contract with Solid, or contact your Solid account representative.

1.2.2 Typographic Conventions

This manual uses the following typographic conventions:

Table 1.1. Typographic Conventions

Format	Used for
Database table	This font is used for all ordinary text.

Format	Used for
NOT NULL	Uppercase letters on this font indicate SQL keywords and macro names.
`solid.ini`	These fonts indicate file names and path expressions.
`SET SYNC MASTER YES;` `COMMIT WORK;`	This font is used for program code and program output. Example SQL statements also use this font.
run.sh	This font is used for sample command lines.
`TRIG_COUNT()`	This font is used for function names.
`java.sql.Connection`	This font is used for interface names.
`LockHashSize`	This font is used for parameter names, function arguments, and Windows registry entries.
`argument`	Words emphasised like this indicate information that the user or the application must provide.
solidDB Administration Guide	This style is used for references to other documents, or chapters in the same document. New terms and emphasised issues are also written like this.
File path presentation	File paths are presented in the Unix format. The slash (/) character represents the installation root directory.
Operating systems	If documentation contains differences between operating systems, the Unix format is mentioned first. The Microsoft Windows format is mentioned in parentheses after the Unix format. Other operating systems are separately mentioned.

1.2.3 Syntax Notation

This manual uses the following syntax notation conventions:

Table 1.2. Syntax Notation Conventions

Format	Used for
`INSERT INTO` *`table_name`*	Syntax descriptions are on this font. Replaceable sections are on *`this`* font.

Format	Used for
`solid.ini`	This font indicates file names and path expressions.
[ ]	Square 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.
{ }	Curly brackets 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.

1.3 Solid Documentation

Below is a complete list of documents available for solidDB. Solid documentation is distributed in an electronic format, usually PDF files and web pages.

Electronic Documentation

- *Release Notes*. This file contains installation instructions and the most up-to-date information about the specific product version. This file (`releasenotes.txt`) is copied onto your system when you install the software.
- *solidDB Getting Started Guide*. This manual gives you an introduction to the solidDB.
- *solidDB SQL Guide*. This manual describes the SQL commands that solidDB supports. This manual also describes some of the system tables, system views, system stored procedures, etc. that the engine makes available to you. This manual contains some basic tutorial material on SQL for those readers who are not already familiar with SQL. Note that some specialized material is covered in other manuals. For example, the Solid "administrative commands" related to the High Availability (HotStandby) Option are described in the *solidDB High Availability User Guide*, not the solidDB SQL Guide.

- *solidDB Administration Guide*. This guide describes administrative procedures for solidDB servers. This manual includes configuration information. Note that some administrative commands use an SQL-like syntax and are documented in the *solidDB SQL Guide*.

- *solidDB Programmer Guide*. This guide explains in detail how to use features such as Solid Stored Procedure Language, triggers, events, and sequences. It also describes the interfaces (APIs and drivers) available for accessing solidDB and how to use them with a solidDB database.

- *solidDB In-Memory Database User Guide*. This manual describes how to use the in-memory database of solidDB In-memory Engine.

- *solidDB SmartFlow Data Replication Guide*. This guide describes how to use the Solid SmartFlow technology to synchronize data across multiple database servers.

- *solidDB AcceleratorLib User Guide*. Linking the client application directly to the server improves performance by eliminating network communication overhead. This guide describes how to use the AcceleratorLib library, a database engine library that can be linked directly to the client application.

  This manual also explains how to use two proprietary Application Programming Interfaces (APIs). The first API is the Solid SA interface, a low-level C-language interface that allows you to perform simple single-table operations (such as inserting a row in a table) quickly. The second API is SSC API, which allows your C-language program can control the behavior of the embedded (linked) database server

  This manual also explains how to set up a solidDB to run without a disk drive.

- *solidDB High Availability User Guide*. Solid CarrierGrade Option (formerly called the HotStandby Option) allows your system to maintain an identical copy of the database in a backup server or "secondary server". This secondary database server can continue working if the primary database server fails.

Chapter 2. Introducing the solidDB CarrierGrade Option

This chapter describes the solidDB CarrierGrade option. The CarrierGrade option enables a secondary server (a "hot standby server") to run in parallel with the primary server and keep an up-to-date copy of the data in the primary server. This allows implementation of systems that have increased reliability. A failed database server no longer brings your site to a complete halt. In as little as a few hundred milliseconds, in any engine configuration supported by solidDB (such as solidDB master or replica), the CarrierGrade option allows the secondary database to replace the failed one.

Note

The term "hot standby" (two words, all lower case) refers to the general technique of having a second server ready to take over if the first server fails. "HotStandby" (one word, capitalized as shown) refers to Solid's specific implementation of this general technique.

2.1 How HotStandby Works

HSB performs synchronous transaction replication between two nodes:

- a *primary server* node (Primary) that contains the active database, and
- a *secondary server* node (Secondary), which contains an exact, up-to-date copy of the active database, and which can replace the Primary if the Primary fails.

The Secondary receives updates from the primary server, and is ready to take over as the Primary if the original Primary fails. An additional benefit of having the Secondary is that the Secondary can also respond to read-only requests (e.g. SELECT statements) from clients. This allows you to spread some of your workload over two servers rather than one.

2.1.1 The Transaction Log and HotStandby

HotStandby uses the Primary server's transaction log, which contains a copy of the transactions committed on the server. In a non-HotStandby server, this transaction log is used to recover data if the server shuts down abnormally. In a HotStandby Primary server, the log data is also sent to the Secondary server to let it know what data to update. The Secondary database runs a continuous roll-forward process that receives the log data and keeps the Secondary's copy of the data synchronized with the Primary's copy.

Typically, if the Primary server goes down, then an independent high-availability manager application tells the Secondary to become the Primary. For brevity, we will call such a manger a "watchdog". After a new Primary is in operation, the clients can then connect to the new Primary and continue working. Clients will see all data that was committed before the Primary went down. (Clients need to re-start any transactions that were started but not finished when the original Primary server went down.)

A special type of client connectivity called *Transparent Connectivity (TC)* is available for clients to operate in the HSB environment, in the presence of failovers and switchovers See Chapter 5, *Using HotStandby with Applications* for more information.

If the Secondary server goes down, the Primary can continue to operate. It continues writing data to the transaction log and keeps that transaction log until the Primary and Secondary are re-connected to each other and the Primary has sent the log to the Secondary. [1]

Once the failed database server becomes available again, it can be configured to become the new Secondary database server (the server that did not fail is already acting as the current Primary, of course).

If the Primary server is the server that fails, then the servers will reverse their responsibilities, with the original Secondary taking over as the Primary, and the original Primary coming back into the system as the new Secondary after it is repaired. These reversals may happen each time there is a failure. The fact that either server can be the Primary allows the system to survive multiple failures over time, and continue operating virtually indefinitely.

Caution

If the Primary server is unable to contact the Secondary server for a long period of time, then the transaction log may fill all the available disk space. This may be avoided with appropriate configuration parameter settings. See Section 4.3.2, “Running Out of Space for Transaction Logs”

You can even use HSB to reduce downtime during hardware and software upgrades. You can leave one server running as Primary while you upgrade the other.

[1] The exact length of time that the Primary keeps the log depends upon the settings of the `solid.ini` configuration parameters `CheckpointDeleteLog` and `BackupDeleteLog`.

1. If `CheckpointDeleteLog=Y`, then the Primary keeps all transaction logs since the time that the Secondary went down or since the most recent checkpoint, whichever is less recent. (For a detailed explanation of checkpoints, see *solidDB Administration Guide*.)
2. If `CheckpointDeleteLog=N` and `BackupDeleteLog=Y`, then the Primary keeps all transaction logs since the time that the Secondary went down or since the most recent backup, whichever is less recent.
3. If `CheckpointDeleteLog=N` and `BackupDeleteLog=N`, then the server keeps the logs indefinitely.

HotStandby can also be used to help choose a customized balance of speed and safety. The HSB parameters *SafenessLevel* and *2SafeAckPolicy* control the way the Secondary server acknowledges the transactions. This parameter, in combination with the logging-related *DurabilityLevel* parameter, lets you specify a combination of speed and safety. Some parameter settings actually increase performance over non-HSB servers. (For more details, see the discussion of durability level and safeness parameters in the chapter on Performance Tuning.)

2.1.2 Server HotStandby States

In a HotStandby system, each server is in one of several possible "states" that describes that server's current behavior. For example, when the Primary and Secondary are communicating and synchronizing, they are in the PRIMARY ACTIVE and SECONDARY ACTIVE states, respectively. As another example, if the Primary loses contact with the Secondary, then the Primary automatically switches to the PRIMARY UNCERTAIN state. In that state, it will not accept new transactions. The user or more typically, the watchdog application may switch the server to the PRIMARY ALONE state, in which the server acts as an independent server — it accepts new transactions and stores them to send to the Secondary later. We describe all the states later.

2.1.3 Server Diagrams

Figure 2.1, “HotStandby Server Scheme” illustrates the basic HotStandby server scheme. In this scheme, there are two database servers — the Primary and the Secondary, each of which has its own disk drive on which it stores the database, and each of which has its own transaction log ("Txn Log"). The Primary writes to its transaction log and forwards it to the Secondary so that the Secondary can make the same changes to its copy of the database. The Secondary's transaction log is not actively involved in HSB, but it is maintained so that the Secondary can recover data that was committed but not yet written to the main data tables. (See *solidDB Administration Guide* for a more detailed explanation of logging and recovery.)

Figure 2.1. HotStandby Server Scheme

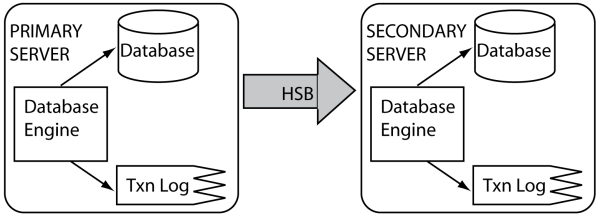
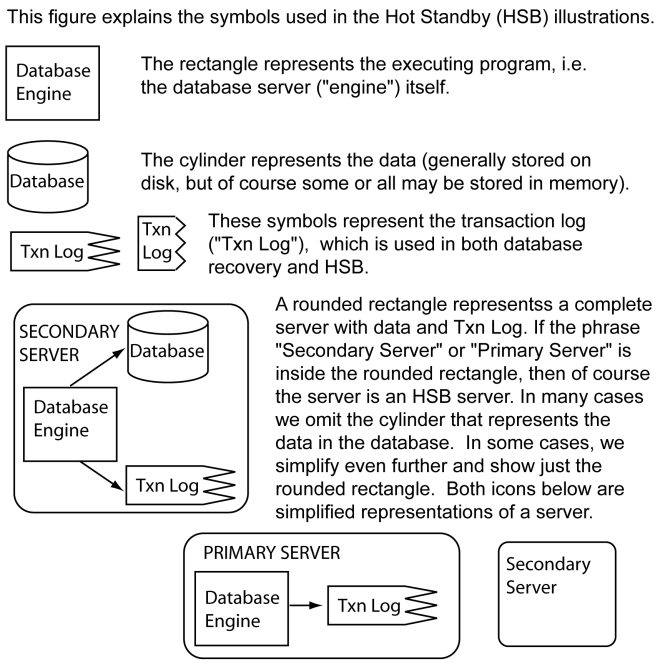


Figure 2.2. Illustration Key

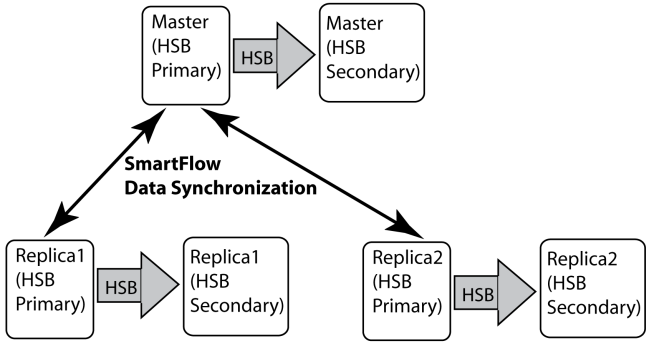


2.1.4 Combining HotStandby and SmartFlow

The Solid CarrierGrade option can be used in combination with Solid SmartFlow. SmartFlow provides bi-directional, periodically occurring data synchronization that allows you to create a distributed system that contains "master" and "replica" servers. With the CarrierGrade option, you can make any of the database servers of the distributed system highly available.

Figure 2.3, “HotStandby with Master and Replica Server Scheme” shows a simple distributed system that contains a master database and two replica databases. Each replica contains at least a subset of the data of the master database. Each of the database servers has been made fault-tolerant with HotStandby replication. The SmartFlow data synchronization occurs between the Primary servers of the database server hierarchy. In case of a problem with any of the Primary database servers, the failed node can do a HotStandby failover making the Secondary server of that node the new Primary. SmartFlow data synchronization can now continue with the new Primary server.

Figure 2.3. HotStandby with Master and Replica Server Scheme



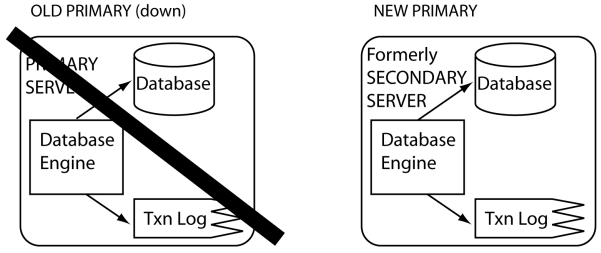
2.1.5 Switching the Secondary to Be the New Primary

When the Primary server fails, the administrator (or "watchdog" software) can use a special HotStandby administrative command to switch the Secondary server to be the new Primary. The command is:

ADMIN COMMAND 'hotstandby set primary alone';

This command must be issued by a client program, such as the watchdog application, that is connected to the Secondary, of course. The new Primary contains the up-to-date committed data from the old Primary database. Everything that was committed in the Primary database, is guaranteed to be found from the Secondary database. Applications can connect to the new Primary and resume their operations. The new Primary can operate alone and continue to write transactions and data to its database and transaction log.

Figure 2.4. HotStandby Switchover to New Primary (old Secondary)

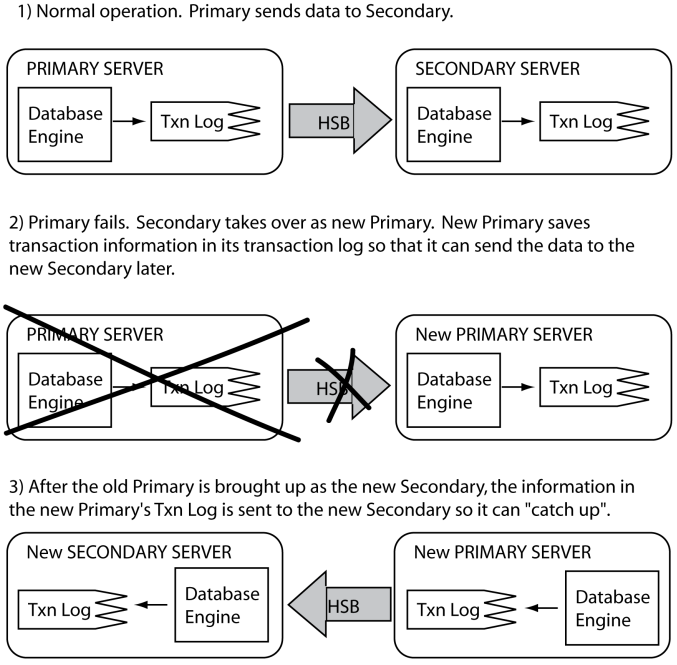


The server that was originally the Secondary becomes the new Primary after the old Primary server fails.

2.1.6 Server CatchUp

Once the old Primary is back online, it may be told to become the new Secondary. However, the information in the new Secondary lags behind that of the new Primary as new transactions have been committed to new Primary database. To bring the new Secondary up to date, the new Primary's transaction log data is sent to the new Secondary automatically after the servers are connected. All pending changes are written from the transaction log to the new Secondary so that the Secondary can keep in sync with the Primary. Server catchup is illustrated in Figure 2.5, “Server Failover and Catchup Example”. For more details on maintaining the synchronization of Primary and Secondary, read Section 4.8, “Synchronizing Primary and Secondary Servers”.

Figure 2.5. Server Failover and Catchup Example



2.1.7 The "Watchdog" Application

When the servers lose contact with each other, they switch to a state such as PRIMARY UNCERTAIN or SECONDARY ALONE. Once in this state, the servers do not normally change state again until they are commanded to do so with a command such as

```
ADMIN COMMAND 'hsb set primary alone';
```

Such a command must be issued by an entity other than the server(s) themselves. The entity may be:

- A human administrator.
- A watchdog application, which is an external user-defined program. The product includes a simple program that can be used as a template.

  (The job of the watchdog may be taken by a high-availability manager (HA Manager) designed to manage redundant components. Such a manager may, in turn, be integrated with a possible high-availability framework if such is provided for the equipment in question.)

We expect that most customers will eventually build or buy watchdog software that meets their needs. Throughout the rest of this manual, we will usually refer to the "watchdog" as though it were a piece of software, but you should keep in mind that the watchdog functionality may be performed by a human. The ADMIN COMMANDs are the same, whether they are issued by a human or by software.

Note also that the "watchdog" (human or software) is not the only factor that controls what happens when servers lose contact with each other.

2.1.8 Replication Modes in HotStandby

1-Safe and 2-Safe Replication

Solid offers various choices to tune the system to the required balance between performance and endurance. One such choice is the choice of replication protocol used. A system parameter called Safeness Level determines whether the replication protocol is synchronous (2-safe) or asynchronous (1-safe).

- *1-safe*: the transaction is first committed at Primary and then transmitted to Secondary
- *2-safe*: the transaction is not committed before it has been acknowledged by Secondary (default).

The safeness level may be controlled at three levels: global (server), session and transaction.

Synchronous HotStandby with 2-Safe Replication

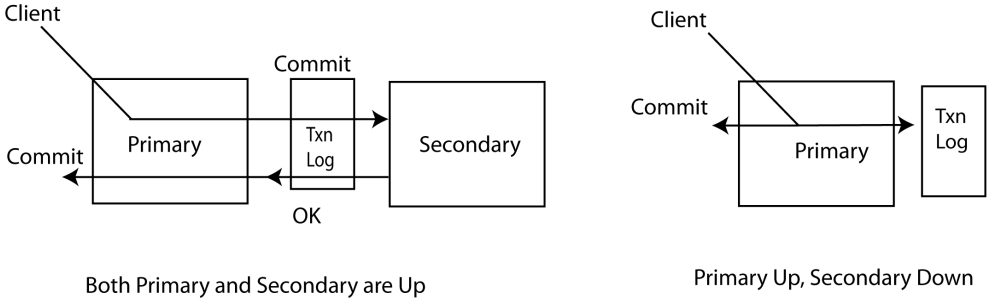
To ensure that the Primary and Secondary have exactly the same data, Solid uses, primarily, a *Synchronous HotStandby* model. This means that the Primary server does not tell the user that the transaction has been committed when the Primary has written the data; instead, the Primary waits until the Secondary server has also committed (or at least received) the data, and only then does the Primary tell the user that the data was

committed. (This is called a "2-safe" replication method; the data is written in two places before the user is told that the data has been committed.) [2]

Before committing changes to a transaction in the Primary database, the Primary server sends the transaction data to the Secondary server. The Secondary server must send acknowledgement to the Primary that it has committed (or at least received) the data. Otherwise, the Primary server times out and changes its state from PRIMARY ACTIVE to PRIMARY UNCERTAIN (these states are discussed in more detail later). The Primary server, in this case, can neither roll back nor commit the transaction. The watchdog may set the Primary server to PRIMARY ALONE state, which allows the Primary to continue to receive transactions and operate independently of the Secondary. It commits the pending transaction(s) that were sent to the Secondary and resumes accepting new transactions

Note that the Secondary server sends acknowledgement as soon as it has committed (or at least received) the transaction log entries. This configuration prevents lost transactions when there is a single failure. Additionally, a file-based transaction log is optionally retained to facilitate database recovery in case a total system failure occurs.

Figure 2.6. Synchronous HotStandby Configuration



Basic Steps in Sending Data

Below are the steps in sending data with synchronous replication:

1. The Primary server writes data (in record level format) to the transaction log at the Primary node.

2. If the Primary server encounters a commit statement, then all changed data is sent to the Secondary server. (Note that if the Secondary server fails after the transaction starts and before the Primary sends the data, then the Primary will roll back the transaction.)

[2]For more about 2-safe vs. 1-safe algorithms, see *Transaction Processing: Concepts and Techniques*, by Jim Gray and Andreas Reuter, Morgan Kauffman, 1993.

3. The Secondary acknowledges the commit message. The timing of the acknowledgement depends upon the setting of the *2SafeAckPolicy* configuration parameter. In the fastest alternative, called 2-safe received, the Secondary sends acknowledgement to the Primary immediately upon receiving the commit message. In the safest alternative, called 2-safe durable, the Secondary sends acknowledgement after executing and durably writing the transaction to the Secondary's own transaction log.

   When the Primary receives the Secondary's acknowledgement, the Primary notifies the user that the data has been committed.

4. If the Primary does not receive acknowledgement from the Secondary (for example, due to network failure or node failure), then the Primary server times out and switches to the PRIMARY UNCERTAIN state. The Primary is unable to roll back or commit the transaction itself because it does not know the state of recent transactions in the Secondary. The Primary does not know which of the following happened:

   - the Secondary was down before the transaction was committed
   - the Secondary already committed the transaction, but the Primary server did not receive acknowledgement, for example because of network failure.

   While the server is in PRIMARY UNCERTAIN state, the current transaction as well as new transactions that a user tries to commit are blocked and the user may perceive that the server is unresponsive.

5. If the Watchdog detects that the Secondary is down or the network failed, then the watchdog can switch the Primary server to the PRIMARY ALONE state. Once the Primary server is set to PRIMARY ALONE, it commits the pending transaction(s) that were sent to the Secondary and resumes accepting new transactions.

6. Changes are accumulated to the transaction log file until the Secondary server is back in operation or until the Primary server is out of disk space. (If the server runs out of disk space for the transaction log, then the Primary changes to read-only mode.)

7. If you think that the Secondary server will be out of operation for a long time and the server is likely to run out of disk space for the transaction log, then you may want to switch the Primary server from PRIMARY ALONE to STANDALONE state. This means that the transaction log will not store all transactions since contact was lost with the Secondary, and therefore the Secondary cannot "catch up" merely by reading the transaction logs from the Primary. If the Secondary can"t be brought up to date with the transaction logs, the only way to synchronize the Secondary with the Primary is to copy the Primary's database file(s) to the Secondary. This can be done with either the HotStandby copy or HotStandby netcopy command. For details on copy and netcopy, read Section 4.8.5, “Copying a Primary Database to a Secondary Over the Network” and Section 4.8.9, “Copying a Database File from Primary Server to a Specified Directory”.

8. To execute either copy or netcopy, the Primary must be in the PRIMARY ALONE state. After a copy/netcopy, the Primary server remains in the PRIMARY ALONE state, regardless of whether the command succeeds or fails.

9. In order for the Primary to again start sending its transactions to the Secondary, the Primary server must be explicitly connected to the Secondary server by using the command hotstandby connect described in Section 4.9, “Connecting HotStandby Servers”. After the Primary server is connected to the Secondary server, the Primary operates in the PRIMARY ACTIVE state.

   Once the servers are connected, they will start performing catchup (when all pending changes are automatically written from the transaction log to the Secondary to keep in sync with the Primary). Note that before server catchup, the Primary and Secondary exchange information and determine where to begin the catchup so that a transaction is not committed twice on the Secondary.

Asynchronous HotStandby with 1-Safe Replication

Optionally, asynchronous replication from Primary to Secondary may be used. This is called 1-safe replication. With 1-safe replication, the transactions are acknowledged immediately once they are committed at the Primary. This offers significant performance gains. After the commit, the transactions are sent to the Secondary, in an asynchronous way. The trade-off is that, when a failure occurs at Primary, a few transactions, that were in transfer, may be lost.

Either of the two replication methods may be chosen dynamically, or even per session, or transaction. The replication delay involved with 1-safe replication may be controlled, too.

2.2 Description of Server States

Both servers in an HSB (HotStandby) pair expose certain states that me be queried and manipulated. The full state diagram is depicted in Appendix D, *Server State Transitions*. The states and their meanings are listed below.

Table 2.1. Description of Server States

STATE	DESCRIPTION
PRIMARY ACTIVE	The servers are connected, and this server (the Primary server) is accepting read-write transactions and sending the data to the Secondary server. The Secondary server must be in SECONDARY ACTIVE state.
PRIMARY ALONE	The peer servers are not interconnected. The peer may be up, but it is not connected and therefore is not accepting any transactions (that is, it may be in the SECONDARY ALONE state).

STATE	DESCRIPTION
	This server (the Primary) is actively accepting and executing read-write transactions and collecting them to be sent to the Secondary later.
PRIMARY UNCERTAIN	The servers have disconnected abnormally and the *AutoPrimaryAlone* configuration parameter is set to "No". In the PRIMARY UNCERTAIN state, any unacknowledged transactions remain in a pending status, which means that the server will not commit or roll back the transaction until a watchdog changes the server to another state. The operator has three possible actions: re-connect the Primary to the Secondary, set the Primary server to PRIMARY ALONE state, or set the Primary server to SECONDARY ALONE state. 1. If the server is re-connected to the Secondary, then the transactions are committed on the Primary. 2. If the state is changed to PRIMARY ALONE, then the open transactions are committed on the Primary. 3. If the state is changed to SECONDARY ALONE, then the open transactions remain pending. They are finally resolved after the server changes to another state. For example, if the server is moved to the SECONDARY ACTIVE state, the blocked transactions are aborted or committed, depending on the catchup outcome. If the server state is changed to STANDALONE or PRIMARY ALONE, then the blocked transactions are committed. If you want the Primary server to automatically go to PRIMARY ALONE rather than PRIMARY UNCERTAIN when it loses contact with the Secondary, then read the description of the *AutoPrimaryAlone* configuration parameter. Note: a watchdog can maximize safety by always switching the server from PRIMARY UNCERTAIN to SECONDARY ALONE. This prevents the possibility of dual primaries. However, it also prevents users from updating data on the server. (See Section 4.3.1, “Network Partitions and Dual Primaries”.)
SECONDARY ACTIVE	The peer servers are interconnected, and this server is accepting incoming transaction log data from the Primary. These transactions are executed on the Secondary so that it has the same data as the Primary (the transactions are also written to the Secondary's own transaction log so that the Secondary itself can recover the data if the Secondary fails). Additionally, clients may perform read-only transactions on a server in the SECONDARY ACTIVE state. When a

STATE	DESCRIPTION
	server is in the SECONDARY ACTIVE state, the server's peer must be in PRIMARY ACTIVE state.
SECONDARY ALONE	The Secondary is disconnected from its peer server. Only read requests are accepted. The server may be connected to the peer by issuing the command "HotStandby connect" on either the Secondary or the Primary.
STANDALONE	The server has no HSB state (Primary or Secondary) and operates in the way a regular standalone server operates. Transaction logs are processed and removed in the normal way, too; they are not saved for the Secondary. To resume HSB operation, the server must be set to either PRIMARY ALONE or SECONDARY ALONE, and the Primary will have to do a netcopy or copy operation to send a complete copy of the database to the Secondary.
OFFLINE	The server was started in "netcopy listen mode" (also called "backupserver mode"). In this mode, the server is waiting for an incoming netcopy from a server that is in PRIMARY ALONE state. When the server successfully completes netcopy, the server moves to the state SECONDARY ALONE. You cannot directly observe the OFFLINE state because a server in OFFLINE state does not accept client connections. A server in the OFFLINE state will respond only to a netcopy operation (described later).

2.3 How Does HotStandby Affect Performance

Although you might expect that HotStandby (HSB) would reduce performance, this is not always the case. In fact, in some configurations HSB can even increase performance significantly. The key factors in HSB performance include:

- the use of adaptive durability when preserving transactions over single failures is needed
- the use of 1-Safe replication protocol when minor transaction loss over failures is acceptable
- the 2-Safe Acknowledgement Policy (when the 2-safe replication is used) — i.e. whether the Secondary acknowledges receipt of transactions as soon as it receives them from the Primary, or whether the Secondary waits until it has committed the transactions
- the possibility of performing read-only transactions on the Secondary server
- the server's internal parallelism

2.3.1 Adaptive Durability

The `solid.ini` configuration file allows you to specify whether you want relaxed durability (fast), strict durability (safe), or a third option, called "adaptive durability".

- Strict Durability: If a transaction is written to the transaction logs as soon as the transaction is committed, we call that "strict durability". This maximizes safety.
- Relaxed Durability: If the server is permitted to defer the transaction write until the server is less busy, or until it can write multiple transactions together, we call that "relaxed durability" (or "relaxed logging"). In a server that is not part of an HSB pair, using relaxed durability means that you risk losing the most recent few transactions if the server terminates abnormally. If the server is part of an HSB pair, however, then a copy of the transaction is on the other server (the Secondary), and even if the Primary server fails before logging the transaction, the transaction is not lost. Thus, when relaxed durability is used with HSB, relaxed durability causes very little reduction in safety. On the other hand, relaxed durability can improve the performance of the system, especially in situations where the server load consists of a large number of small write transactions.
- Adaptive Durability: Adaptive durability applies only to HotStandby Primary servers (and it is the default). Adaptive Durability means simply that if the server is in Primary Active state (sending transactions to the Secondary), then it will use relaxed durability, but in any other state it will use strict durability. This gives you high performance (with little loss of safety) when HSB is active, yet maintains high safety if only one server is operating. Adaptive Durability may effectively enacted only when the 2-Safe replication is used (default).

Adaptive durability can significantly increase performance while still guaranteeing high degree of data safety in failure situations. It can increase overall system throughput and it can reduce latency, that is, the time the user must wait before being told that the transaction has committed.

For more details about relaxed logging and the *`DurabilityLevel`* parameter, see *solidDB Administration Guide*.

1-Safe Replication

With 1-safe replication, the commit statement is acknowledged immediately once the commit processing is completed at the Primary. The committed transaction is transmitted to the secondary asynchronously, after the control has been returned to the application. The delay involved in transmitting the transaction may range from few milliseconds to a few hundred milliseconds. 1-safe replication offers significant performance gains because the latencies are reduced dramatically at Primary. The downside of 1-safe is that, in the case of a failure, a few transactions may be lost in a failover.

The 1-safe replication may be set, for the server, with the parameter:

```
[HotStandby]
SafenessLevel=1safe ;values: 1safe, 2safe, auto; default is 2safe
```

It is also possible to control the safeness level dynamically with the SET commands:

```
SET SAFENESS {1SAFE| 2SAFE| DEFAULT}
```

sets the safeness level for the current session, until it is changed.

```
SET TRANSACTION SAFENESS {1SAFE| 2SAFE| DEFAULT}
```

sets the safeness level for the current transaction. After commit, the safeness level returns to the value set for the session, or the startup value, or the system default (which is 2-safe).

The option DEFAULT denotes the current setting for the session.

It is also possible to control the safeness level with the programmatic durability controls (like SET DURABILITY RELAXED) when the *SafenessLevel* parameter has a special value "auto" (i.e. "automatic"). In this case, "strict" corresponds to "2-safe" and "relaxed" to "1-safe".

2-Safe Acknowledgement Policy

When the 2-safe replication is enabled (default), the Primary server does not tell the client that the transaction has been successfully committed until the Primary receives acknowledgement that the Secondary has the transaction. Solid currently allows three different acknowledgement policies:

- 2-safe received. The Secondary server sends acknowledgement when it receives the data (default).
- 2-safe visible. The Secondary has updated its copy of the data, and the change is now "visible". In other words, a client application that has connected to the Secondary server will be able to see the update.
- 2-safe durable. The Secondary server acknowledges when it has made the data durable, that is, when it has committed the data and written the data to the disk.

2-safe received is faster. 2-safe durable is safer. Note that since these acknowledgement policies apply only when the Primary and Secondary server are both active (i.e. both are applying the transactions), even 2-safe received is considered safe. You risk losing transactions only if both servers fail practically simultaneously (within a second of each other).

Using 2-safe received reduces latency, i.e. the amount of time between the start of the commit and the time that the user receives confirmation of the commit. The *2SafeAckPolicy* has little impact on overall throughput.

For more details about the *2SafeAckPolicy* parameter, see Appendix A, *Configuration Parameters*.

Internal Parallelism

When you use the HotStandby (HSB) feature, every transaction that contains a write operation is executed twice — once on the Primary, and once on the Secondary. In some situations, this may mean that a single transaction takes approximately twice as long with HSB as without HSB. However, this does not mean that overall throughput will decrease 50%. The servers have a high degree of parallelism, and while the Secondary is working on one transaction, the Primary will work on another transaction.

To ensure that your system takes advantage of parallelism, we suggest that you spread your transactions across several connections rather than submitting all transactions through the same connection. Note, however, that the more queries you run in parallel, the more memory the server needs, so adding connections and running queries in parallel does not always increase throughput, especially in systems that do not have a large amount of memory. You may need to experiment to find the optimal number of queries to run at a time.

Performing Read-Only Transactions on the Secondary

Clients are allowed to connect to the Secondary and perform read-only operations.

In some situations, you can "spread the load" and improve your system's overall performance by having read-only clients connect to the Secondary and perform their reads there. This is particularly useful for work such as report-generation or "data warehousing" queries, where you want to read a lot of records and don"t want to change any of them.

Other Performance Factors

Not surprisingly, actual throughput and response times depend on many factors, including (but not limited to) the speed of the network, the amount of other traffic on the network, the complexity of the SQL statements, and the number of SQL statements per transaction. The usual factors, such as amount of memory and disk speed, also affect performance, of course.

Other Safety Factors

The hot standby approach is designed to protect you against the failure of a single part of your system. However, it won't protect you if both servers can be affected by the same problem, such as a power failure. If you set the *DurabilityLevel* to "relaxed" or "adaptive", and if your acknowledgement policy is 2-safe received, then you may lose transactions if both servers go down nearly simultaneously. At the very least, each server should be connected to an Uninterruptible Power Supply (UPS) to protect against power failure. Furthermore,

as with any database system, important data should be backed up and probably should be archived at a separate site. HotStandby is not a substitute for backing up your data. Note that you can run the backup (using ADMIN COMMAND "backup" command) on either of the servers of the HSB pair. Often it is the secondary server that has more resources available for creating the backup.

Summary

If you have read-only queries, for example queries that generate summary reports, you may want to run those on the Secondary to spread the workload over both machines.

Unless you need the highest possible level of safety, you can increase performance by doing the following:

- Use adaptive logging (set *DurabilityLevel=2*)
- Use "2-safe received" mode (set *2SafeAckPolicy=1*).

Note that even if you use the "less safe" settings specified here (AdaptiveDurability and 2-safe received mode), you are still protected by HSB unless there are at least 2 failures. You sacrifice very little safety for much higher performance. Solid HotStandby gives you high performance and safety at the same time.

2.4 HotStandby Configuration

A HotStandby configuration allows for a Primary server, Secondary server, and watchdog to reside in different machines and use different operating systems and APIs as shown in the example in Figure 2.7, “Heterogeneous HotStandby Configuration with Watchdog”. For details on implementing heterogeneous configurations, read Section 2.5.1, “HotStandby Configuration and System Design Issues”.

All communication between the Primary and Secondary database (including putting a failed system back in service and re-synchronizing Primary and Secondary databases) occurs within existing communication layers, such as TCP/IP. HotStandby requires no auxiliary storage or transfer methods, such as shared disks or ftp transfers.

2.5 Implementing the CarrierGrade Option

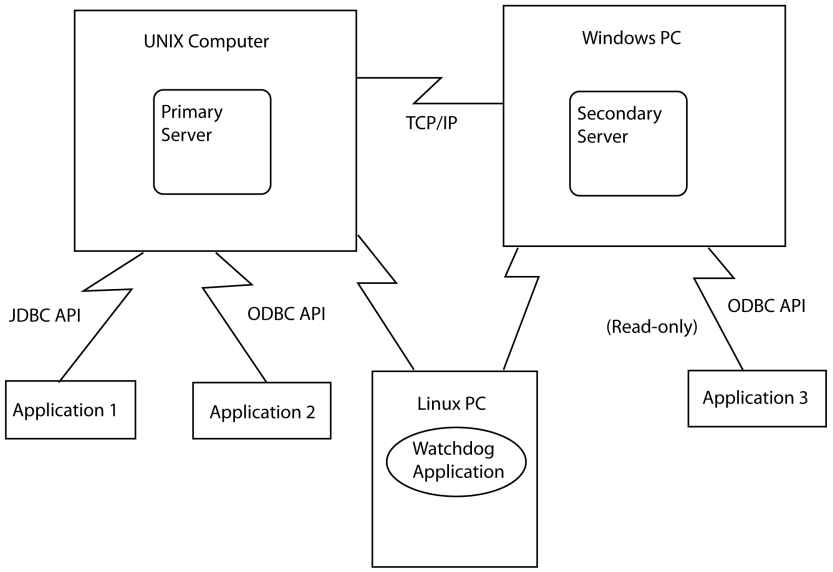
This section describes how to configure a system to use the HotStandby functionality provided by the CarrierGrade option. This system also discusses some important design issues.

2.5.1 HotStandby Configuration and System Design Issues

How you configure HotStandby (locally or remotely, at one or more different locations, over the Internet, and with a watchdog program) can affect the reliability and efficiency of your system. This section addresses these issues.

The illustration below shows one example of a heterogeneous system, in which the Primary and Secondary servers do not even use the same type of hardware and operating system.

Figure 2.7. Heterogeneous HotStandby Configuration with Watchdog



2.5.2 Watchdog Configuration

For better efficiency and more precision in monitoring the state of the servers, a watchdog is recommended as a separate component of any HotStandby configuration.

If only two machines are available, making it impossible to run a watchdog program in a separate machine, run the watchdog on the same machine where the Secondary server resides and set the parameter *AutoPrimaryAlone* to no in the configuration file (`solid.ini`) of both the Primary and Secondary server. Note that setting this parameter to "no" is extremely important, as it prevents the potential error of having two Primary servers.

Caution

If both servers are in a state that allows writing (PRIMARY ALONE or STANDALONE), and if the databases of both servers are independently updated, then it will not be possible to re-synchronize the two databases. Make sure that your watchdog does not allow both servers to be put in the PRIMARY

ALONE or STANDALONE state at the same time. See Section 4.3.1, “Network Partitions and Dual Primaries”.

If the Primary server does fail, then the watchdog is able to switch the Secondary to become the new Primary.

There are some disadvantages to putting the watchdog in the same machine as the Secondary. The disadvantages include:

- If only the communication link between the watchdog and the Primary is down, this configuration may result in a false switchover between the Primary and the Secondary.
- The communication link becomes a "single point of failure", i.e. a single failure that may disable the entire system. (In most HotStandby configurations, the entire system is not disabled unless there are at least 2 failures.)
- If there is a network failure and the Secondary machine cannot communicate with the Primary machine, the users and applications are still able to access the Primary server and theoretically could continue operating with the Primary server. However, the Primary server stops accepting transactions because the watchdog can"t notify the Primary server to continue operating, for example by switching to PRIMARY ALONE state.

Note

In some operating systems or HA (High Availability) frameworks, a reliable "heartbeat" between two machines provides a way of detecting if two machines are running. If you have such a system, using the "heartbeat" is the preferred way to implement the watchdog. The heartbeat mechanism significantly improves the reliability of the HotStandby system and watchdog operations.

Chapter 3. Getting Started with HotStandby

This chapter provides step-by-step instructions for setting up two Solid HotStandby servers (a Primary server and a Secondary server).

This chapter assumes you have already installed solidDB with the CarrierGrade option. Be sure to follow the installation instructions that came with the product.

3.1 Before You Begin

Before you set up the CarrierGrade option, please note the following information:

1. Read Section 4.1, “What You Should Know”. This section contains important information about using the Solid CarrierGrade option.

2. To learn about HotStandby features, run the demonstration contained in the solidDB package. For details, read the section below.

3. Refer to Section 3.3, “HotStandby Setup and Configuration Procedure” for step-by-step instructions on setting up and configuring HotStandby.

4. Get acquainted with the Solid command line SQL editor solsql or the multipurpose graphical tool SolidConsole. SolidConsole is available for download, at www.solidDB.com/downloads.

3.2 HotStandby Demonstration

The Solid software package that you installed includes all the files that you need to run a quick demonstration of the HotStandby feature. This demonstration is simplified, but will increase your understanding of how to use the feature.

Detailed instructions for the demonstration are in the file:

- `samples\hsb\readme.txt`

3.3 HotStandby Setup and Configuration Procedure

After you have completed the demonstration and are ready to configure a HotStandby system of your own, please follow these instructions. This section describes how to:

1. Prepare to set up and configure the server

2. Start the servers and connect them as a HotStandby pair

3.3.1 Section 1: Setup and Configuration Preparations

This section prepares you for using the HotStandby configuration procedures provided in Section 2. It provides a brief overview of the setup and configuration prerequisites for using the CarrierGrade option.

Hardware

Make sure you have the appropriate hardware to support your HotStandby physical configuration. For example, if you want to use the CarrierGrade (HotStandby) option, at minimum, you will need two computers, one for the Primary server and one for the Secondary server. You may need a third computer for the watchdog software, if you choose to use a watchdog to monitor the Primary and Secondary servers.

3.3.2 Section 2: Step-by-Step Procedure

This section provides initial set up and configuration instructions. Solid CarrierGrade (HotStandby) option requires that you initially configure one solidDB server to run as a Primary and another solidDB server to run as a Secondary.

To set up your HotStandby servers (without any other Solid options), follow the steps below.

Important

When executing ADMIN COMMANDs in the instructions below, make sure that you use single quotes when single quotes are shown. Double quotes and single quotes are not interchangeable.

1. Configure the Primary and Secondary nodes

   At minimum, HotStandby requires that you configure the following parameters in the *`[HotStandby]`* section of the `solid.ini` configuration file:

   - *`HSBEnabled`*
   - *`Connect`*

   If you omit the *`HSBEnabled`* parameter in a server that you intend for HotStandby, then the server will be a non-HotStandby server when it is started. If you omit the *`Connect`* parameter, then the server will start as a HotStandby server, but you will have to provide the connection string via an ADMIN COMMAND before the servers can connect.

The transaction logging of the HotStandby servers for local recovery purposes may be either enabled or disabled. If the logging is disabled, the primary server keeps the necessary part of the transaction log information in memory for replication. Disabling the transaction logging improves performance of write transactions but reduces the degree of data safety. By default, transaction logging is enabled. You can disable it by setting the following in your `solid.ini` configuration file:

```
[Logging]
LogEnabled=no
```

For more details about these parameters, read Section 6.2.1, “Defining Primary and Secondary HotStandby Configuration”.

If you are using the TCP/IP protocol, you may want to adjust the timeout interval between your applications and the HotStandby servers. For details, read Section 6.1.2, “Defining Timeouts Between Applications and Servers (Com Section)”.

Optionally, you may change the default settings for HotStandby-specific configuration parameter options. For details on these parameters read, Section 6.2, “Configuring HotStandby-Specific Parameters”.

2. Configure the watchdog application (optional)

   The CarrierGrade option includes a sample watchdog application for monitoring Primary and Secondary servers. You can create a custom watchdog program based on the simple sample that Solid provides.

   If you are using the watchdog application, you need to provide configuration settings for the watchdog. For details, read Section 6.4, “Configuring Parameters for a Watchdog”.

3. Start the server that will become the Primary

   Start the server the way you would start any solidDB server. The server will read the HotStandby configuration information from the `solid.ini` file.

4. Switch the state of the database server to "PRIMARY ALONE"

   After the Primary database server is started, switch the Primary server's state to PRIMARY ALONE (if the state is not already PRIMARY ALONE) by issuing the following command:

   ADMIN COMMAND 'hotstandby set primary alone';

   NOTE: In admin commands, you may use the abbreviation "hsb" in place of "hotstandby", for example:

   ADMIN COMMAND 'hsb set primary alone';

NOTE: In SolidConsole, you can execute any of the above commands by direct manipulation of the GUI interface (in the Administration/HSB pane).

5. Copy the Primary database to the Secondary node

   a. Start the Secondary server with the command-line parameter **-x backupserver**.

   b. Issue the following command on the Primary:

      ADMIN COMMAND 'hotstandby netcopy';

   c. To verify that the netcopy was completed, issue the following command on the Primary:

      ADMIN COMMAND 'hotstandby status copy';

      Note that you use the keyword "copy" even though you did a "netcopy" operation.

   For more details about copying a database from the Primary to the Secondary, refer to the instructions in Section 4.8.9, “Copying a Database File from Primary Server to a Specified Directory”.

6. Connect the Primary server to the Secondary server

7. At this point, the Secondary server should be up and running.

   To tell the Primary server to connect to the Secondary server, issue the following command to the Primary server:

   ```
   ADMIN COMMAND 'hotstandby connect';
   ```

   To verify that the connection was successful, type the following command:

   ADMIN COMMAND 'hotstandby state';

   The Primary server should respond that its state is "PRIMARY ACTIVE".

8. Start using the applications and the watchdog

   This completes the set up and configuration instructions.

Chapter 4. Administering the CarrierGrade Option

This chapter describes how to maintain your CarrierGrade (HotStandby) installation. The administration topics included in this chapter are:

- Overview of HotStandby administration operations
- Performing HotStandby recovery and maintenance
- Shutting off HotStandby operations
- Synchronizing Primary and Secondary servers
- Connecting HotStandby servers
- Checking HotStandby status
- Verifying HotStandby server states
- Using HotStandby with applications
- Special Configurations: Higher Safety vs. Lower Cost

4.1 What You Should Know

This section describes what you need to know about HotStandby before you begin administration and maintenance.

4.1.1 HotStandby and the Solid Configuration File

To enable HotStandby functionality, you must provide a special *`[HotStandby]`* section in the solidDB configuration file (`solid.ini`), and your license file must allow use of the Solid CarrierGrade (HotStandby) option.

The minimum configuration information required in the *`[HotStandby]`* section is:

- Set *`HSBEnabled`* to "yes".

- Set the *Connect* parameter setting for the server. This setting defines the network name used to connect to the other server (either Primary or Secondary). If you do not set this parameter in the `solid.ini` file, then the server will not start up as a HotStandby server. Note that after the server has started, you may set or change this parameter by using an ADMIN COMMAND.

For example, on your Primary server, your solid.ini file might look similar to the following:

```
[Com]
Listen="tcp 1301"
[HotStandby]
HSBEnabled=yes
Connect="tcp 188.177.166.12 1302"
[Logging]
LogEnabled=yes
[... other sections]
```

And your Secondary's `solid.ini` file might look like:

```
[Com]
Listen="tcp 1302"
[HotStandby]
HSBEnabled=yes
Connect="tcp 188.177.166.11 1301"
[Logging]
LogEnabled=yes
[... other sections]
```

Note that each server's "connect" string must match with the other server's "Listen" string.

Remember that the parameters in the `solid.ini` file are checked only when the server starts. If you want to change the setting(s) of a running server, you may use ADMIN COMMANDs to do that. Refer to Appendix A, *Configuration Parameters*, for more details on parameter manipulation.

Read Chapter 6, *Configuring HotStandby*, for details on the *Connect* parameter and other parameter settings. Refer to Chapter 3, *Getting Started with HotStandby*, if you are setting up the Solid CarrierGrade option for the first time.

4.1.2 HotStandby and Access Rights

Administrators require no special access rights to run HotStandby. Normal access rights apply in both the Primary and Secondary servers. For administration purposes, SYS_ADMIN_ROLE or SYS_CONSOLE_ROLE

are required to execute the HotStandby administrative commands. These commands are executed with the Solid SQL command:

ADMIN COMMAND 'hotstandby command_string';

For example, if you use Solid SQL Editor (teletype):

ADMIN COMMAND 'hotstandby status connect';

When the HotStandby command is entered using Solid Remote Control (teletype), enter the hotstandby command string only (without the quotes), for example:

hotstandby status connect

4.1.3 Solid Tools and the CarrierGrade Option

All tools available for performing administration with solidDB apply also to the CarrierGrade option. You can issue HotStandby-specific administrative commands in Solid SQL Editor solsql (or Solid Remote Control solcon). In addition, Solid Speedloader (SOLLOAD), Solid Export (SOLEXP), and Solid Data Dictionary (SOLDD) can be used with the CarrierGrade option. Also, SolidConsole offers various possibilities to monitor and manage an HSB configuration. For a description of these tools, read "Using Solid Data Management Tools" in *solidDB Administration Guide*.

4.1.4 Database Migration (disk-based servers only)

solidDB 4.x databases, FlowEngine 3.x databases and Solid Embedded Engine 3.5x databases can be converted to the latest solidDB format by using one of the following command-line parameters:

-xconvert (to convert the database file to the new structure and shut down the server), or

-xautoconvert (to convert the database file and continue running)

All required system tables, including those for the HotStandby functionality, are created. After the conversion, the converted databases can no longer be used with the older product versions. Therefore, you are urged to back up your databases and files before migrating to the new release.

Note

When Solid databases are no longer in use by HotStandby, they remain compatible with solidDB.

4.1.5 Interoperability

The Primary and Secondary should be of HSB-compatible versions. Typically, adjacent versions are HSB-compatible. See the Release Notes for information on HSB-compatibility with previous versions.

4.2 Limitations with HotStandby

4.2.1 In-Memory Tables

If you are connected to the Secondary and you are reading data from in-memory tables, the transaction isolation level is automatically set to READ COMMITTED, even if you specified REPEATABLE READ. (In-memory tables do not support SERIALIZABLE on either the Primary or the Secondary.)

4.3 Warnings

To use HotStandby safely, you need to avoid "dual primaries" and you need to avoid running out of transaction log space. Both these issues are explained below.

4.3.1 Network Partitions and Dual Primaries

In some circumstances, it is possible to have both servers acting in PRIMARY ALONE state. This is a serious, unrecoverable error because if each server commits any transactions that the other does not, then you cannot re-synchronize the servers because there is no way to "merge" the databases to create a single database that has correct information. In practice, the transactions committed in the "wrong primary" database during the dual primary situation, will be lost. Having dual primaries can also lead to other errors.

This paragraph describes one way that you can accidentally wind up with two Primary servers. Suppose that you start with a Primary and a Secondary that are operating properly. Suppose that the network fails, and suddenly neither server is able to communicate with the other. You are the System Administrator, and you think that the Secondary server has failed, so you switch the Primary server to PRIMARY ALONE state. Meanwhile, your assistant, who is monitoring the Secondary server, thinks that the Primary server has failed, and therefore switches the original Secondary into PRIMARY ALONE state. Your system now has two servers running in PRIMARY ALONE state. Suppose further that at least one client that was connected to the original Primary has lost contact with the original Primary and now tries to connect to the original Secondary (which is now also in PRIMARY ALONE state). If that client performs a transaction on the original Secondary, and another client performs a transaction on the original Primary, then neither server's data is a superset of the other, and neither server can be said to have the "correct" version of the data. Furthermore, there is no way to "merge" the two servers" data to get a set of data that is guaranteed to be consistent.

We refer to this as the "dual primaries" problem. It is most likely to be caused by a "network partition" — i.e. a situation in which some but not all network connections are lost and your single network effectively becomes divided into separate sub-pieces, each of which allows communication within the piece but not with other pieces. Thus both servers lose connections with each other, but are still up and running, and in some cases may still be able to communicate with some clients.

The dual primary situation is uncommon. Even if you do wind up with dual primaries, you won't actually have inconsistent data unless someone is able to perform a write operation on the original Secondary (after it has switched to PRIMARY ALONE). If the original Secondary is completely cut off from the rest of the network, then no one can write to it, and the original Primary will be a superset of the Secondary, and you will still be able to get a single consistent set of data (after you reconnect the servers and allow the Secondary to catch up with the changes made on the original Primary).

Although dual primaries are rare, they are extremely dangerous when they do occur, and you must use extreme caution to prevent your data from becoming inconsistent.

The possibility of dual primaries is higher when you have multiple watchdogs (or human administrators), each of which also is not in contact with the "other" watchdog.

The chances of dual primaries are also higher if you have set the configuration parameter *`AutoPrimaryAlone=Yes`* in the `solid.ini` files of one or both servers. Using *`AutoPrimaryAlone=Yes`* means that your system may respond more quickly to failures, and does not need to rely on a watchdog, but it also means that the system no longer has any independent observer (watchdog or human) to prevent dual primaries.

4.3.2 Running Out of Space for Transaction Logs

When you use HotStandby, if you put a server in PRIMARY ALONE state, you must be careful that it doesn"t run out of disk space for transaction logs.

In a non-HotStandby server, if you checkpoint frequently, then the transaction log doesn"t grow very large because after each checkpoint the server deletes the "old" transaction log(s) — i.e. the logs with the data changes that occurred before the checkpoint. (For more information about checkpointing, see *solidDB Administration Guide*.)

However, in a HotStandby server that is operating in PRIMARY ALONE state, the server must keep the transaction logs that have accumulated since the time that the Primary lost contact with the Secondary. If the Secondary is down for a long time, the server may keep a large amount of transaction log data that it would otherwise throw away after each checkpoint. In a worst-case situation, if the Secondary cannot be brought back up in a reasonable time and there is not enough disk space to store all the transactions that occur, then the Primary's transaction logs may fill up all of the available disk space. This will cause the server to switch to read-only mode.

You can prevent this from happening by setting the appropriate value of the parameter *MaxLogSize* in the *[HotStandby]* section. After reaching the specified total log size, the server will automatically switch to the STANDALONE state, at the next checkpoint. (In a diskless server, the state will remain PRIMARY ALONE, though, as there is no disk writing at all)

If the server is set to the STANDALONE state, it will not keep all transactions logs since the time that the Primary lost contact with the Secondary. Without complete transaction logs, of course, you cannot synchronize your system merely by connecting the Primary to the Secondary and allowing the Secondary to "catch up" by reading old logs. You will have to copy the entire database from the Primary to the Secondary by using the "copy" or "netcopy" command. These commands are described later.

4.4 Overview of Administration Tasks

This chapter describes administration tasks you may need to perform when using HotStandby. Topics included in this section are:

Table 4.1. Administration Tasks

Topic	Description	Page
Performing HotStandby Recovery and Maintenance Tasks	Describes HotStandby tasks in the case of a system failure (resulting from either a broken communication link or an inoperable hot-standby server). These tasks include:	Section 4.5, “Performing HotStandby Recovery and Maintenance”
Copying a Primary database to a new Secondary over the network.	Describes how to create a remote (network) copy of a database when the remote server is a new addition to the HotStandby configuration (i.e. is a new Secondary), or the remote server's data becomes corrupted and must be replaced.	Section 4.8.5, “Copying a Primary Database to a Secondary Over the Network”
Checking HotStandby Status	Describes how to check HotStandby status information for the Primary and Secondary servers.	Section 4.10, “Checking HotStandby Status”
Verifying HotStandby Server States	Describes how to check the state (Primary, Secondary, or standalone) of a HotStandby server.	Section 4.11, “Verifying HotStandby Server States”

Topic	Description	Page
Changing a HotStandby Server to a non-HotStandby Server	Describes how to set a server configured for HotStandby to a normal, non-HotStandby server.	Section 4.13, “Changing a HotStandby Server to a Non-HotStandby Server”

4.5 Performing HotStandby Recovery and Maintenance

In case of a system failure or the need for server maintenance, you may be required to perform some or all of the following operations:

- Switch the server state. This includes setting the Primary server to PRIMARY ALONE state, which continues accumulating transactions in the transaction log so that they can be sent to the Secondary later, or

  Shut down HotStandby.

- Synchronize the servers to be sure the Primary and Secondary databases are identical.

- Connect the Primary server to the Secondary server if the communication link is broken for some reason.

These topics are described in following sections. For details on re-connecting applications to Secondary or Primary databases, read Section 5.3.1, “Reconnecting to Primary Servers from Applications”.

4.6 Switching Server States

The CarrierGrade option requires that a user or an external application switch the server state when necessary. You can set up a watchdog program (an external monitoring program) to provide state switches in case of a Primary database failure.

4.6.1 Switchover and Failover

By switchover we mean reversing the roles of the Primary and Secondary when they are running. This may be needed for various maintenance purposes.

On the other hand, failover is an action of taking up the role of the Primary, by the Secondary, if Primary fails.

4.6.2 Performing Switchovers

A watchdog program may reverse the roles of the servers by issuing the following command at the Secondary:

ADMIN COMMAND 'hotstandby switch primary';

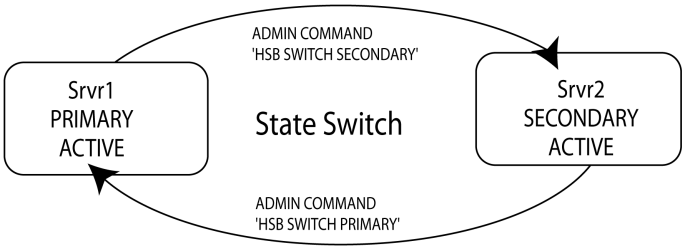
or, at the Primary:

ADMIN COMMAND 'hotstandby switch secondary';

This command can be used whether or not the two servers are connected. If the servers are connected, the states are simply reversed; the old Secondary becomes the new Primary, and the old Primary becomes the new Secondary. If the servers are not connected, the old Secondary becomes the new Primary, and the other server's state is unchanged.

The diagram below shows what happens if you issue the command "switch secondary" or "switch primary" when the servers are connected. Note that the command "switch primary" is only issued on a server that is in a SECONDARY state (e.g. SECONDARY ACTIVE), while the command "switch secondary" is only used on a server that is in a PRIMARY state (e.g. PRIMARY ACTIVE).

Figure 4.1. State Switch



When executing the command hotstandby switch primary to switch the Secondary server (Srvr2) to Primary, if the Secondary server (Srvr2) is not connected to the other server (Srvr1), then an error is returned.

If the two servers are connected, they switch states. In other words, the old Primary (Srvr1) becomes the new Secondary and old Secondary (Srvr2) becomes the new Primary.

If the old Secondary (Srvr2) cannot connect to the other server (Srvr1), then both servers switch to SECONDARY ALONE. (Note that even if the *AutoPrimaryAlone* configuration parameter (described later) is set to yes, the new Primary will switch to SECONDARY ALONE, not PRIMARY ALONE.)

When the hotstandby switch primary command is executed, it starts a process to switch the state. If the switch process started successfully, the following message is displayed:

```
Started the process of switching the role to primary
```

During the switch, all active write transactions are aborted. You can monitor the status of the switch using the command hotstandby status switch. For details, read Section 4.6.4, “Verifying the Switch”.

4.6.2.1 Switching Primary to Secondary

You can switch a Primary server to a SECONDARY state by issuing the command:

ADMIN COMMAND 'hotstandby switch secondary';

This is particularly useful if two servers have switched states and you want to switch them back to their original states. For example, continuing the example above, when the new Secondary comes back in service, you can then switch its state back to Primary and switch the new Primary back to Secondary.

When executing hotstandby switch secondary, if the servers are not already connected to each other, then the old Primary tries to connect to the old Secondary.

If the two servers are connected, they switch states. In other words, the old Primary becomes the new Secondary and the old Secondary becomes the new Primary.

When the hotstandby switch secondary command is executed, it starts a process to switch the state. If the switch process started successfully, the following message is displayed:

```
Started the process of switching the role to secondary
```

You can check the switch status of any HotStandby server to verify if a switch was performed successfully. For details, read Section 4.10.1, “Displaying Switch Status Information”.

4.6.3 Important Notes on Switching Servers

- If you issue a COMMIT after a SWITCH command, the COMMIT fails with an error:

  ```
  'replicated transaction is aborted'.
  ```

  All transactions are terminated during the switch. Note, however, that ADMIN COMMANDs (administrative commands), such as the "HSB switch" command, are not transactional commands and cannot be rolled back. (NOTE: Administrative commands do force the start of a new transaction if one is not already open, however. To avoid leaving an open transaction, or having a transaction's start time be different than you expected, you may want to execute COMMIT WORK after administrative commands.)

  In the event of a configuration error that causes both servers to have the state of PRIMARY (e.g. both are PRIMARY ALONE), you can use the command hotstandby switch secondary to switch one of the servers back to a SECONDARY state. If the servers have the same data, then normal operations on both servers

are resumed. However, if the servers do not have the same data, then the Primary server rejects the connect operation from the Secondary and issues the following message:

```
14525: HotStandby databases are not properly synchronized.
```

HotStandby replication is not started. In this case, a full copy of the Primary database is required at the Secondary server. You will first need to decide which database is correct. Note that if a 14525 error occurs, the database states do not change; both servers remain in the same state they were in before the command was issued.

- It is recommended that you use a watchdog to alert you of a Primary node failure. The watchdog can also switch server states in case of a Primary failure. Read Chapter Chapter 7, *Monitoring HotStandby Server Pairs with a Watchdog Application* for details on using a watchdog for monitoring and switching server states.

- Client applications can continue their operations with the new Primary or Secondary (read-only) after they have reconnected to the new server(s). The application should be written to connect to a new Primary in the event of a database failure. For details, read Section 5.3.1, “Reconnecting to Primary Servers from Applications”.

4.6.4 Verifying the Switch

You can check the status of the switch process at the Primary or Secondary with the following command:

ADMIN COMMAND 'hotstandby status switch';

The command displays a status message that tells you if the switch has never occurred between the two servers, is successful, still in progress, or if the switch has failed. Refer to Appendix B, *Error Codes*, for the meaning of error messages.

4.6.5 Performing Failovers

A failover is performed by executing, at the Secondary, the command:

ADMIN COMMAND 'hotstandby set primary alone';

The server gains the new state once all the pending transactions received before, from the Primary, are processed. This will guarantee that no transactions are lost, and the database state reflects the state at the Primary just before the failure. However, if the safeness level used is 1-safe, some transactions may be lost in failover.

4.6.6 Running the New Primary in PRIMARY ALONE State

Although the connection to the Secondary server is broken, this state lets you run a Primary server with continuous updates to the transaction log. After the Secondary server comes back up, the server in PRIMARY ALONE state can resume sending transactions to the Secondary server.

There are three ways to set a server to PRIMARY ALONE state:

- By issuing the following command:

  ADMIN COMMAND 'hotstandby set primary alone';

  or

- By doing a controlled disconnect:

  ADMIN COMMAND 'hotstandby disconnect';

  at either the Primary or the Secondary. Note that if you do a controlled shutdown by executing

  ADMIN COMMAND 'shutdown';

  on the Secondary, then the Secondary will implicitly disconnect before shutting down, and the Primary will safely switch to the PRIMARY ALONE state.

- By setting the configuration parameter *AutoPrimaryAlone* in the *[HotStandby]* section of the configuration file (`solid.ini`) to "yes", to default to the PRIMARY ALONE state.

  If the PRIMARY ALONE state is the default, then the server is automatically put in PRIMARY ALONE state when the connection to the Secondary is broken. Otherwise, after a server fails, the server's state remains PRIMARY UNCERTAIN unless the command ADMIN COMMAND 'hotstandby set primary alone' is issued by the administrator or the watchdog program. By default, the *AutoPrimaryAlone* parameter in the *[HotStandby]* section of the `solid.ini` file is set to "no", which specifies that the Primary server operating in its PRIMARY ACTIVE state is switched to PRIMARY UNCERTAIN automatically if the Secondary server fails.

The PRIMARY ALONE state persists until one of the following occurs:

- A connection is successfully made to the Secondary server.
- The server runs out of space for the transaction log.
- The log size limit (*MaxLogSize*) is reached.

- Another command switches the server to another state, such as STANDALONE.
- The Primary server is shut down.

Caution

One should be careful not to perform shutdown of the Primary simultaneously with commanding Secondary to the PRIMARY ALONE state. The two operations are conflicting and may result in the Secondary gaining the SECONDARY ALONE state, instead. The coincidence hardly will happen in a real operation. However, one may be tempted to simulate the Primary failure with a shutdown, while testing the system. This should not be done, as shutdown is no substitute for failure. It is a complex distributed operation involving both Primary and Secondary. Another reason for not doing that is that a Primary server, after being shut down, and consequently started up as a new Secondary, will not be able to catchup with the new Primary. If there is a real need to shutdown Primary, the correct sequence is: (1) perform the switchover, and (2) shutdown the new Secondary. The new Primary will automatically switch to the PRIMARY ALONE state.

4.6.7 Bringing the Secondary Server Back Online

To bring the Secondary server back online, connect the Primary with the Secondary server. For details, read Section 4.9, “Connecting HotStandby Servers”.

Once you bring a Secondary node online, it may require catchup. Changes in the Primary have accumulated over a period of time. While the Primary was set to PRIMARY ALONE state, the Primary wrote transactions and data to the transaction log.

When the Secondary is connected again to the Primary, the Primary's pending changes are written from the transaction log to the Secondary server for synchronization. While the changes are written to the Secondary, the Secondary is in SECONDARY ALONE state and the Primary is in PRIMARY ALONE state. (If you issue the command ADMIN COMMAND "hsb status connect', you will get a message telling you whether the servers are performing catchup.) After the Secondary has successfully finished processing these pending changes, the Primary and Secondary servers" states are automatically changed to PRIMARY ACTIVE and SECONDARY ACTIVE, respectively.

Note

If the Primary server was set to the STANDALONE state using the command **hotstandby set standalone**, the full database must be copied from the Primary to the Secondary before the Secondary can be put in SECONDARY ACTIVE state. Read Section 4.8, “Synchronizing Primary and Secondary Servers”.

4.7 Shutting Off HotStandby Operations

You may occasionally need to temporarily shut off HotStandby operations in the Primary server — for example, if you are taking the Secondary server out of service to upgrade it and the Primary does not have enough disk space to store the transaction logs that will accumulate while the Secondary is out of service. (See Section 4.3.2, “Running Out of Space for Transaction Logs” for more details.)

To shut off HotStandby at the Primary server, you must disconnect the servers (if they are currently connected) and then set the Primary server to STANDALONE state, using the following sequence of commands.

ADMIN COMMAND 'hotstandby disconnect'; -- if servers are connected

ADMIN COMMAND 'hotstandby set standalone';

This allows the Primary server to continue operating as though it were a non-HotStandby server.

Note

Once you have stopped storing transaction logs to send to the Secondary, you can no longer have the Primary and Secondary servers catch up merely by connecting them again. Instead, you will need to manually synchronize the servers when you resume HotStandby operations. For details, read the following section.

If you want to permanently stop using this server as a HotStandby server, then see Section 4.13, “Changing a HotStandby Server to a Non-HotStandby Server”.

4.8 Synchronizing Primary and Secondary Servers

In order for the servers to start HSB replication, the servers' databases must be identical. In other words, the secondary database must be an exact copy of the primary database. The process of making the databases of a HotStandby system identical is called HotStandby synchronization.

Situations where the Primary and Secondary need to be synchronized include:

- the Secondary is new and does not yet have a copy of the Primary's database to start with.
- the Secondary was not running for awhile and its copy of the data is not up-to-date.
- both the "Primary" and the "Secondary" were running in Primary Alone state at the same time, and thus have conflicting data.
- the Secondary's disk drive crashed, or the file was corrupted and must be replaced.

There are two main ways of synchronizing the data on the servers: "full copy" and "catchup".

4.8.1 Catchup

Catchup can be used if and only if the Primary server has stored a copy of all of the transactions that the Secondary server "missed" while the servers were disconnected. If the Primary has stored all those transactions, then when it is reconnected to the Secondary, it will automatically forward those transactions to the Secondary so that the Secondary can "catch up" to the Primary.

A solidDB server stores transactions (to forward to the Secondary) only while it is in the PRIMARY ALONE state, not while it is in the STANDALONE state or is operating as a non-HotStandby server. Therefore, if the server has been in STANDALONE state or has been operating as a non-HotStandby server since it last was connected with the Secondary, then it does not have all the transactions and cannot do catchup. Instead, you will have to do a full copy (described later).

There is no explicit "catchup" command. The servers will automatically try to catch up when you connect them using

ADMIN COMMAND 'hotstandby connect';

When the Primary and Secondary are connected, they automatically check to see whether the Primary server has data in its transaction logs to send to the Secondary. If the data is there, the servers automatically attempt to catch up.

During the catchup process, the Primary and Secondary servers stay in PRIMARY ALONE and SECONDARY ALONE states. Clients may continue to submit queries and commit transactions. The catchup process is transparent to the client applications.

If the servers recognize that the Primary and Secondary databases are not identical even after copying transactions from the Primary to the Secondary, you will get an error message.

If catchup fails (or if you know ahead of time that it will not work because the Primary server was in STANDALONE state, for example), then you will need to do a full copy.

Catchup applies only when the Secondary has already been running in SECONDARY ACTIVE state at some point. If you have a brand new Secondary server, then even if the Primary was running in PRIMARY ALONE state and has stored all transactions since the time that the Primary itself started, you will need to do a full copy to give the Secondary its initial copy of the database.

There is additional information about the catchup process in Section 4.6.7, “Bringing the Secondary Server Back Online”.

4.8.2 Full Copy

A full copy is just what its name implies: copying all the data from the Primary to the Secondary. This is done by copying the database file(s) themselves.

Full copy is used in the following situations:

- The Secondary server is brand new and is getting its initial copy of the Primary's database.
- The Primary server has written transactions when it was not in the PRIMARY ALONE state, and therefore catchup is not possible.
- The Secondary's database is corrupted or missing.
- The Secondary is diskless and has experienced a failure. When a Secondary diskless server is started after a failure, the diskless server requires a complete copy of the database using the **hotstandby netcopy** command. Unlike a disk-based Secondary, the Secondary diskless server cannot read the transaction log and apply the changes that occurred while it was inoperable.
- The Primary server has all of the data needed for catchup, but catchup is expected to take longer than simply copying the current data files.

Caution

If the Secondary server has old database files, a full copy will write over those old files. If for any reason the files on the Secondary contain data that was not in the Primary (for example, if both servers were operating in PRIMARY ALONE state at the same time), then that data will be lost.

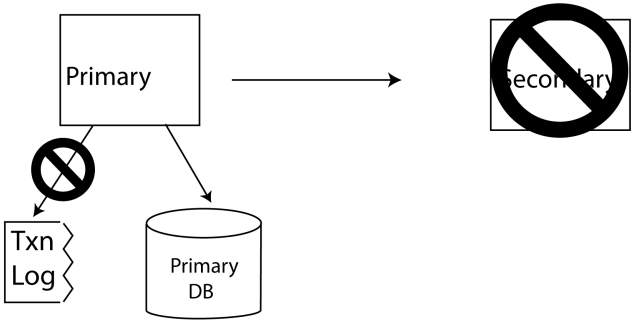
There are two HotStandby commands that can do a full copy - i.e. that copy the database file(s) from the Primary to the Secondary. You may use either of the following:

```
ADMIN COMMAND 'hotstandby netcopy';
ADMIN COMMAND 'hotstandby copy [<directory_name>]';
```

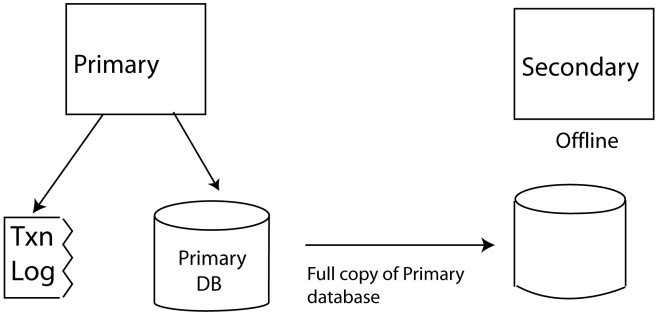
The **netcopy** operation copies the database over the network to a Secondary server that is running and can receive the file(s) over the network. The **copy** operation copies the database files to a specified disk drive directory that is visible to the Primary server. The secondary server must not be running during the **copy** operation. The **netcopy** command is usually preferable to the **copy** command, so most of our examples will show only **netcopy**, not "copy".

The **copy** and **netcopy** commands are described in Section 4.8.9, “Copying a Database File from Primary Server to a Specified Directory” and Section 4.8.5, “Copying a Primary Database to a Secondary Over the Network”.

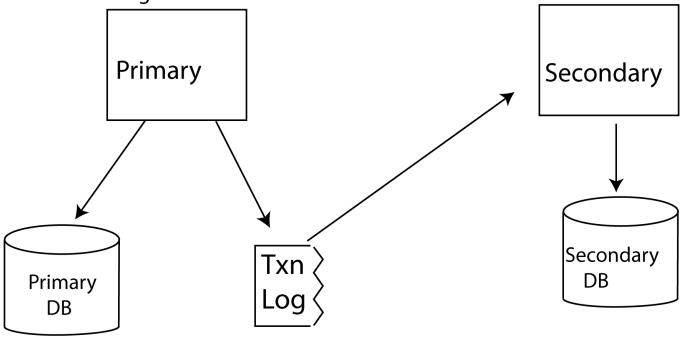
Figure 4.2. Manual Full Copy Procedure



Secondary is down for a long time so the primary stops using the transaction log to store data for the secondary. (The log is still used for local recovery.)



Primary database is copied to the secondary node and the primary starts writing to the transaction log.



Database is now copied and the primary sends transaction log file changes to the secondary.

Note

The preceding diagram over-simplifies the usage of the transaction log. In the first part of the diagram, when the Primary and Secondary are not connected, the Primary actually continues to write data to the transaction log, but keeps only enough data to perform recovery, not enough to allow the Secondary to catch up with all the changes since the connection was broken.

4.8.3 Verifying the Copy

You can verify the status of the copy or netcopy operation by issuing the following command at the Primary server:

ADMIN COMMAND 'hotstandby status copy';

Note that you use the keyword "copy" (not "netcopy") even if the operation was a netcopy.

The command displays a status message that tells you whether the copy operation was successful, is still in progress, or has failed, indicated by an error code and error message. Refer to Appendix B, *Error Codes*, for the meaning of error messages.

4.8.4 Using a Watchdog to Synchronize Servers

The commands that allow you to synchronize servers manually can also be used by a watchdog program to synchronize servers automatically. If catchup is sufficient, then all that the watchdog needs to do is monitor the Secondary to see when it comes up, and then execute the command to connect the Primary to the Secondary. If full copy is required, then the watchdog can instruct the Primary server to do a netcopy (or copy) operation. Remember that a full copy will write over any data on the Secondary.

4.8.5 Copying a Primary Database to a Secondary Over the Network

The netcopy command sends a copy of the database file from the Primary server to the Secondary server. The Secondary server must already be running. The command to perform a netcopy is:

ADMIN COMMAND 'hotstandby netcopy';

When the Primary does a netcopy, the Primary uses the connect string that is specified in the *`[HotStandby]`* section of `solid.ini`.

For details on the *`Connect`* parameter, which defines the connect string, see Section 6.2.1, “Defining Primary and Secondary HotStandby Configuration”.

When you execute the **hotstandby netcopy** command, it performs a database checkpoint, before it sends a copy of the Primary database.

To execute the command, the Primary server should be in PRIMARY ALONE state.

The Primary continues accepting transactions and connections during the netcopy (however, any ADMIN COMMAND that changes the server state will be rejected.) The Secondary does not continue accepting transactions and connections. When the **netcopy** starts, if the Secondary has any open connections or transactions, it will roll back the open transactions and disconnect from its clients, then it will start receiving the **netcopy**. While the Secondary receives the **netcopy**, the Secondary will communicate only with the Primary server.

When the **netcopy** is completed successfully, the Secondary's state changes to SECONDARY ALONE (if it wasn"t already in that state).

The Primary server stays in the PRIMARY ALONE state during the netcopy operation. After the netcopy has successfully completed, the Primary server continues to stay in the same state. Before you can resume full hot standby operations, you must connect the Primary and Secondary servers, which will set the Primary server to PRIMARY ACTIVE state. For information about connecting the two servers, see Section 4.9, “Connecting HotStandby Servers”.

There are two major situations in which you use netcopy to create a copy of the database for the Secondary server:

- When creating a database for a brand-new Secondary that has never had one before. (This method is also used when copying a database to a diskless Secondary, since after a failure it loses its entire database and must be treated as a brand new Secondary.)
- When replacing an existing Secondary database (e.g. one that has been corrupted)

Both of these are discussed below.

4.8.6 Creating a New Database for the Secondary Server

Normally, when you start a solidDB server, it asks you if you"d like to create a new database (if there isn"t already a database). However, if the server is a Secondary server, it should use a copy of the Primary's database rather than create its own database. Therefore, when you start a Secondary server that does not have an existing database, you must give it a command-line parameter to tell it to wait to receive a copy of the database from the Primary. The command-line parameter is **-x backupserver**. For example, you would start the Secondary server with the command:

solid -x backupserver

The space between the "-x" and "backupserver" is optional. The following is equivalent:

solid -xbackupserver

The **-x backupserver** command line parameter tells the server to go into "netcopy listening mode" (also called "backup listening mode"). In this mode the only possible operation for the Secondary server is to receive a database copy from the Primary server. The Secondary will not respond to any other command, and in fact will not even accept a connection request from a client program such as solsql, your application, or a watchdog program.

If there exists a Secondary database, you can start the server in a normal way that will result in the server being in the SECONDARY ALONE state.

Once the Secondary has been started with **-x backupserver**, or is in the SECONDARY ALONE state, you can execute the netcopy command on the Primary.

First, make sure that the Primary is in PRIMARY ALONE state. Then issue the following command on the Primary:

ADMIN COMMAND 'hsb netcopy';

On the Primary, the **hotstandby netcopy** command uses the connect string defined with the *`connect`* parameter in the `solid.ini` configuration file to connect to the Secondary server. Once connected, it copies the database files through the network link.

In netcopy listening mode, the Secondary server only attempts to open the Secondary database after it has received a new database copy through the **hotstandby netcopy** command at the Primary.

Following is the procedure to create the Secondary database copy:

1. Be sure you have configured the `solid.ini` file so that it is valid for the HotStandby configuration. For details on the *`Connect`* parameter, which defines the connect string, see Section 6.2.1, “Defining Primary and Secondary HotStandby Configuration”.

   This connect string will be used to connect to the Secondary server from the Primary and to copy the database files over the network.

2. Start the Primary server.

3. Start the Secondary server in netcopy listening mode by executing the following command:

   solid -x backupserver

   Or, alternatively, start the Secondary server with an existing database.

4. Set the Primary server to PRIMARY ALONE state if it isn"t already in that state.

   ADMIN COMMAND 'hotstandby set primary alone';

5. Issue the following command at the Primary server:

   ADMIN COMMAND 'hotstandby netcopy';

6. After the netcopy has completed, you can connect the two servers and start (or resume) full hot standby operation by issuing the command:

   ADMIN COMMAND 'hotstandby connect';

When the Secondary server receives a new copy of the database through the network link, it opens the Secondary database. After the servers are connected (with the "hsb connect" command), the Secondary server runs in its normal SECONDARY ACTIVE state and is ready to accept user transactions from the Primary.

Note

If netcopy is sent to a server that is in the SECONDARY ALONE state, the existing database is overwritten with the copied database. This option is handy if there is a need to resynchronize databases, or to repair a corrupted Secondary database.

4.8.7 Replacing an Existing Database on the Secondary Server

Although **netcopy** is used primarily to send a database to a Secondary that has never had a database before, **netcopy** can be used in other situations as well. For example, if the Secondary's copy of the database has been corrupted (due to a disk drive crash, for example), then you may send the Secondary a copy of the Primary's database by using the **netcopy** command.

If you are replacing an existing database, then the Secondary server does not need to be in "netcopy listening mode"; in other words, you do not need to start the Secondary server with **-x backupserver**. Simply make sure that the Primary is in PRIMARY ALONE state and the Secondary is in SECONDARY ALONE state, then issue the following command to the Primary:

ADMIN COMMAND 'hotstandby netcopy';

Note that after the netcopy completes, the Primary server will still be in PRIMARY ALONE state and the Secondary server will automatically be put in SECONDARY ALONE state (if it wasn"t already in that state). The servers will not automatically connect; you will still need to issue the command:

ADMIN COMMAND 'hotstandby connect';

If you do a netcopy while the Secondary is in SECONDARY ALONE state, and if any clients are connected to the Secondary (to do read-only queries), then the Secondary server rolls back any open transactions and breaks any client connections. Once the netcopy is completed, the Secondary server will remain in the SECONDARY ALONE state.

4.8.8 Verifying Netcopy Status

When you start a netcopy command, it runs asynchronously in the background. The servers do not display a message when the netcopy completes. In fact, the servers do not even display a message if the netcopy fails due to a problem such as a network error. To see whether the netcopy completed successfully, you should always verify the status of the netcopy by using the following command at the Primary server:

ADMIN COMMAND 'hotstandby status copy';

(Note that the command uses the keyword "copy", not "netcopy". The same command is used for both the copy and netcopy operations.)

The command displays a status message that tells you if the netcopy was successful, is still in progress, or has failed, indicated by an error code and error message. Refer to Appendix B, *Error Codes*, for the meaning of error messages.

4.8.9 Copying a Database File from Primary Server to a Specified Directory

In some cases, the Primary and Secondary servers may be able to see some of the same disk drive(s) and therefore can read and write some of the same directories. If the directory that the Secondary will use for the database is visible to the Primary, then you can use the "hotstandby copy" command to copy the database from the Primary's directory to the Secondary's directory.

To copy the file using "hotstandby copy", issue the following command at the Primary server:

ADMIN COMMAND 'hotstandby copy[*`directory_name`*]';

where:

`directory_name` is the name of the directory that you want to copy the file to. The format of the directory name is operating system dependent.

The directory name is optional. If you do not specify a directory name, then the server will use the value specified by the *`CopyDirectory`* parameter in the `solid.ini` configuration file.

One key difference between the **hotstandby copy** command and the **hotstandby netcopy** command is that the **netcopy** command can be used only when the Secondary is running, while the **copy** command should be used only when the Secondary server is NOT running. Performance-wise, there is no significant difference between the two database copy methods.

Caution

Before using the **hotstandby copy** command, be sure to shut down the Secondary server. The Secondary server must not try to access the database file while the Primary is writing that file.

When you execute the **hotstandby copy** command, it creates a checkpoint to the database, and then makes a copy of the Primary database before sending that copy to the Secondary.

After a copy operation, the Secondary is still down. You must bring it back up and then issue the "hotstandby connect" command to connect the two servers.

The Primary server should be in PRIMARY ALONE state when you issue the command, and the Primary server will remain in that state during (and after) the command. Since the server is in PRIMARY ALONE state, transaction processing on the Primary continues normally during the copy command, and the Primary will store the transactions in the transaction log so that they can be forwarded to the Secondary later. When the Primary database is connected to the Secondary using the administrative command **hotstandby connect**, the Primary and Secondary servers automatically perform "catchup" to bring the Secondary up-to-date.

4.8.9.1 Starting the Secondary Server and Catching up

When the copy is completed, you must start the Secondary server with the newly copied database. To do this, you start the server the normal way, that is, by issuing the command "solid" at your operating system prompt:

solid

After you re-start the Secondary server, use the **hotstandby connect** command at the Primary server to connect the Primary server to the Secondary server.

ADMIN COMMAND 'hotstandby connect';

The **hotstandby connect** command is discussed in more detail in Section 4.9, “Connecting HotStandby Servers”.

After the Primary is connected to the Secondary, the Primary server and Secondary server automatically start performing catchup. This means that the Primary server brings the Secondary database up-to-date by copying the Primary's transaction log to the Secondary, and then the Secondary rolls forward the transaction log and updates its copy of the database.

4.9 Connecting HotStandby Servers

If the connection between the Primary and Secondary servers is broken or not yet established, you need to issue the following command at the Primary or Secondary node:

ADMIN COMMAND 'hotstandby connect';

For example, after performing a netcopy, you normally connect the servers.

Since there is no automatic connect mechanism in the HotStandby servers, you should have the watchdog application perform this command when the connection between the servers is broken.

After issuing this command, a confirmation message is displayed if the connection between the Primary and Secondary servers is successful. Note that if the Primary and Secondary are connected, but the transaction log is not yet fully copied at the Secondary, you will receive the following message from the Primary server:

```
Started the process of connecting the servers
```

If the state of the Primary server was PRIMARY UNCERTAIN or PRIMARY ALONE when you executed the command and if the connection is successful, then the state of the Primary server changes to PRIMARY ACTIVE. If unsuccessful, the state remains PRIMARY UNCERTAIN or PRIMARY ALONE. If you see an error message, refer to Appendix B, *Error Codes*, for the meaning of the error message.

The connect string that the Primary uses to connect to the Secondary server is specified using the *Connect* parameter in the *[hotstandby]* section of the `solid.ini` configuration file. You can view current connect settings in the Primary and Secondary nodes by issuing the command:

ADMIN COMMAND 'hotstandby cominfo';

For details on querying connect status at Primary and Secondary servers, read Section 4.10.2, “Displaying Connect Status Information”. For details on re-connecting an application to the Primary server, read Section 5.3.1, “Reconnecting to Primary Servers from Applications”.

4.10 Checking HotStandby Status

This section describes the HotStandby status information that you can request from both the Primary and Secondary servers.

To check status, issue the following command in the Primary or Secondary server:

ADMIN COMMAND 'hotstandby status *option*';

where *option* can be one of the following:

Table 4.2. ADMIN COMMAND 'hotstandby status' Options

Option	Description
catchup	Indicates whether or not the server is doing catchup. Catchup occurs after the Primary server connects to the Secondary. During catchup, the Primary sends accumulated transaction logs so that the Secondary can apply the changes. Possible values are: 'ACTIVE' and 'NOT ACTIVE'. For more details, see *hsb status catchup* in Appendix C, *Summary of HotStandby Administrative Commands*.
connect	Shows whether the last attempt to connect the servers was successful. For more details, see *hsb status connect* explanation in Appendix C, *Summary of HotStandby Administrative Commands*.
copy	Shows whether the last attempt to copy/netcopy was successful. For more details, see *hsb status copy* in Appendix C, *Summary of HotStandby Administrative Commands*.
switch	Shows whether the last attempt to switch the server into PRIMARY ACTIVE or SECONDARY ACTIVE state was successful. For more details, see *hsb status switch* in Appendix C, *Summary of HotStandby Administrative Commands*.

In addition, the next two sections contain more details about the status of "switch" and "connect" operations.

Example

```
ADMIN COMMAND 'hotstandby status catchup';
```

4.10.1 Displaying Switch Status Information

You may need to verify if a state switch occurred between two HotStandby servers. To check HotStandby switch status information, issue the following command in the Primary or Secondary server:

ADMIN COMMAND 'hotstandby status switch';

- When no prior switch has occurred between the two servers, the following message is displayed: NO SERVER SWITCH OCCURRED BEFORE
- When the switch process is still active, the following message is displayed: ACTIVE
- When the most recent prior switch process has completed successfully, the following message is displayed: SUCCESS

- When the most recent attempt to switch has failed, the following message is displayed: ERROR *number*, where *number* identifies the type of error that occurred during the switch. Refer to Appendix B, *Error Codes*, for the meaning of the error message.

4.10.2 Displaying Connect Status Information

You can query connect status information between the Primary and Secondary servers. This capability is equivalent to the SQL function HOTSTANDBY_CONNECTSTATUS, which you can use in the application code.

To check connect status, issue the following command in the Primary or Secondary server:

ADMIN COMMAND 'hotstandby status connect';

where the possible return values are:

Table 4.3. Connect Status Return Values

Error Code	Text	Description
0	CONNECTED	Connect active. Returned from both the Primary and Secondary server.
14007	CONNECTING	Primary server connecting to the Secondary server. Returned from both the Primary and Secondary servers.
14008	CATCHUP	Primary server is connected to the Secondary server, but the transaction log is not yet fully copied. Returned from both the Primary and Secondary server.
14010	DISCONNECTING	The servers are in the process of disconnecting.
14537	BROKEN	Connection is broken. Returned from both the Primary and Secondary servers.

4.10.3 Displaying Communication Information

You can query communication information used to connect to the other servers. This is the value of the *Connect* parameter setting in the *[HotStandby]* section of the solidDB configuration file (`solid.ini`). You can use this information in the client applications to connect to other servers.

To display communication information, issue the following command in the Primary or Secondary server:

ADMIN COMMAND 'hotstandby cominfo';

4.10.4 Displaying Role Start Time

Sometimes it is important to know when the server entered the current role of Primary or Secondary. This information may be retrieved by using two corresponding options of the **ADMIN COMMAND 'info'**:

```
admin command 'info primarystarttime';
       RC TEXT
       -- ----
        0 2005-06-09 14:22:18
admin command 'info secondarystarttime';
       RC TEXT
       -- ----
        0 2005-06-09 18:24:44
```

The reported time is the time the role has become Primary or Secondary. The STANDALONE state is considered to be a state of the Primary role.

Specifically, the primary starttime is set when the following transitions occur:

* SECONDARY ALONE => PRIMARY ALONE

* SECONDARY ALONE => STANDALONE

* SECONDARY ACTIVE => PRIMARY ACTIVE

The secondary starttime is set when:

* The server is started in the SECONDARY ALONE state

* OFFLINE (started with **-x backupserver**) => SECONDARY ALONE

* PRIMARY ALONE => SECONDARY ALONE

* STANDALONE => SECONDARY ALONE

* PRIMARY ACTIVE => SECONDARY ACTIVE

If the current role contradicts the query, the query returns an empty string. For example, if the role is SECONDARY and the command 'info primarystarttime' is issued, it returns an empty string.

4.11 Verifying HotStandby Server States

When administering and maintaining HotStandby, it is often necessary to check the state of HotStandby servers. Appendix D, "Server State Transitions," provides a summary of HotStandby state transitions that occur while performing administrative and troubleshooting operations. To check the current state of a HotStandby server, issue the following HotStandby command in the server:

ADMIN COMMAND 'hotstandby state';

The command returns one of the following states:

Table 4.4. HotStandby Server States

State	Description
PRIMARY ACTIVE	The connected server is a normal Primary server. Changes to the Primary server are sent to the Secondary server.
STANDALONE	The Primary server accepts transactions and stores them in the database; however, it is not connected to the Secondary, and it also does not store the transactions so that they could later be sent to the Secondary.
PRIMARY UNCERTAIN	The Primary server has a broken connection to the Secondary server. The servers have disconnected abnormally and *AutoPrimaryAlone* is set to "No". In the PRIMARY UNCERTAIN state, any open transactions are pending, i.e. the server will not commit or roll back the transaction until a watchdog changes the server to another state.
PRIMARY ALONE	The Primary server is working by itself. The connection to the Secondary is broken, but transactions are accepted. The transactions are stored so that they can later be sent to the Secondary.
SECONDARY ACTIVE	The connected server is a normal Secondary server. The server acknowledges transactions that are sent from the Primary server.
SECONDARY ALONE	The Secondary server has a broken connection to the Primary server. The Secondary will still accept read-only requests.

Note

1. If a server's `solid.ini` configuration file is configured to make the server a HotStandby server, then when the server is started the server will start in the SECONDARY ALONE state.

2. Note that the OFFLINE state is not listed in the preceding table; the server cannot return the state name "OFFLINE" because when the server is in the OFFLINE state you cannot connect to it and issue any query (such as ADMIN COMMAND 'hotstandby state').

3. If ADMIN COMMAND 'hotstandby state' is issued on a server that is not configured for Hot-Standby, the following error message is returned:

```
14527: This is a non-HotStandby Server
```

4.11.1 Server States Overview

Not all combinations of server states are possible. For example, the Secondary can only be in SECONDARY ACTIVE state if the Primary is in PRIMARY ACTIVE state. The following table shows possible server states of a HotStandby server when its associated server is in a particular state.

Table 4.5. Server States

State of the Server	Possible state(s) of the Associated Server
PRIMARY ACTIVE	SECONDARY ACTIVE
PRIMARY ALONE	PRIMARY ALONE * PRIMARY UNCERTAIN SECONDARY ALONE STANDALONE *
PRIMARY UNCERTAIN	PRIMARY ALONE PRIMARY UNCERTAIN SECONDARY ALONE STANDALONE
SECONDARY ACTIVE	PRIMARY ACTIVE
SECONDARY ALONE	PRIMARY ALONE PRIMARY UNCERTAIN

State of the Server	Possible state(s) of the Associated Server
	SECONDARY ALONE STANDALONE
STANDALONE	PRIMARY ALONE * PRIMARY UNCERTAIN SECONDARY ALONE STANDALONE *

* If one server is in the PRIMARY ALONE state or STANDALONE state, the other server should not be in the PRIMARY ALONE or STANDALONE state. This is because if changes are made to both servers independently, there is no way to merge the two databases into one.

4.12 Choosing Which Server to Make Primary

In some situations, when you are trying to recover from a failure where both databases have failed, you may not know which server should be made the Primary. The server that was the Primary before the servers lost contact with each other is not necessarily the server that should become the Primary now.

To determine which server should become the Primary, you can use the following command on each server:

ADMIN COMMAND 'hsb logpos';

This function returns a value as a string or binary value. The server that has the "greater" value (the one which has accepted more transactions) is the server that should become the Primary.

To use the command, follow the instructions below:

1. Both servers should be up and running and in SECONDARY ALONE state.

2. Connect to both servers.

3. In each server, execute:

   ADMIN COMMAND 'hsb logpos';

   Successful admin commands will return error code 0, a string, and the server's previous role. For more information and an example on the command output, refer to Appendix C, *Summary of HotStandby Ad-*

ministrative Commands, table row logpos. (Note: The application should regard the string as an opaque value, which has no defined structure.)

4. Compare the string values. For example, in C, use the `strcmp()` function. The server that returned the string that was "greater" should be chosen to be the new Primary. If the strings are equal, then either server may be switched to Primary.

5. Select the Primary by using the command below on the server that will become Primary:

   ADMIN COMMAND 'hsb set primary alone';

6. Connect the HotStandby servers with each other by using the command below:

   ADMIN COMMAND 'hsb connect';

7. If the previous command succeeds, the Secondary catches up with the Primary, and the HotStandby pair is functional again. If the command fails, you must separately synchronise the nodes by issuing the command below on the Primary server:

   ADMIN COMMAND 'hsb netcopy';

   The netcopy command does not give a return value when it has finished. Instead, you must observe it actively. This can be done with the command below: ADMIN COMMAND 'hsb status copy'; The possible return values are ACTIVE, SUCCESS or FAILED. In the case of a failure, the reason for the failure is also output. After the synchronisation is done, issue the command:

   ADMIN COMMAND 'hsb connect';

   The HotStandby pair is functional again.

Caution

This procedure does not guarantee that the server with the higher string value is a superset of the other server. It is still possible that the two servers will each have accepted transactions that the other did not — e.g. both servers may have been running in PRIMARY ALONE state. To detect the possibility that neither server is a superset of the other, the servers compare information when executing the "connect" command. If neither server is a superset, then the *`Connect`* command will fail and give an appropriate error message.

4.13 Changing a HotStandby Server to a Non-HotStandby Server

You can change a Primary or Secondary server to become a normal, non-HotStandby server by editing the *[HotStandby]* section of the `solid.ini` file. remove the *HSBEnabled* parameter (or set it to "no"). We recommend that you also remove or comment out the *Connect* parameter. After changing the parameter settings in the `solid.ini` file, you must re-start the server for the changes to take effect.

If you want the server to temporarily stop acting as a HotStandby server, but you would like it to resume acting as a HotStandby server later, then you may want to leave the `solid.ini` file unchanged and instead simply change the state of the server to STANDALONE. See Section 4.7, “Shutting Off HotStandby Operations”.

4.14 Special Configurations: Lower Cost vs. Higher Safety

Solid's HotStandby solution uses pairs of Primary and Secondary servers to provide true hot standby capability. Using pairs of servers is not optimal for every customer, however. If near-instantaneous failover is not required, you may not be able to justify the expense of having a Secondary for every Primary server. At the other extreme, some customers may need extra reliability and may have the money to buy "spares for the spares", i.e. to purchase not only a Secondary for every Primary, but also 1 or more additional spare servers so that when a Primary goes down and its Secondary replaces it, a spare can be used as the "new Secondary" if the original Primary cannot be repaired quickly.

To allow customers to reduce costs or increase reliability, Solid HotStandby (HSB) supports some alternatives to the standard hot standby model. The standard model is sometimes called the "N+N" or "2N" model, because the number of Primary and Secondary servers is the same ("N"). The alternatives include:

- N + 1 Spare or N + M Spares: This is the Spare Node scenario for Standalone. There are N "primary" servers and 1 or more spares. There are no Secondary servers. A failed "primary" server is replaced with a spare. This is not a true "hot standby" scenario and is better called "warm standby", since the computer is available but it does not have a copy of the database.

- 2N + 1 Spare or 2N + M Spares: This is the Spare Node scenario for HotStandby. There are N HotStandby pairs, i.e. every Primary has a Secondary. In addition, there are M spares, where M is at least 1 and usually less than N. When a Primary or Secondary fails, a spare is brought in as the new Secondary. Thus a Primary server never operates alone for long, even if its original partner has failed.

Below, we explain the N+M (or N+1) and the 2N+M (or 2N+1) approaches and the Solid features that help you implement these.

4.14.1 Reducing Cost: N + 1 Spare and N + M Spares Scenarios

In these scenarios, there are N "primary" servers, each of which operates in Standalone state, that is, without being connected to a Secondary. In addition, there are M spare servers, where M is at least 1 and usually less than N. If a "primary" server goes down, one of the spares replaces it. Data is copied from the original server to the spare, then the original server is taken offline and the spare is configured to act as the original server. Note that any spare can replace any Primary server (no spare is dedicated to a particular Primary server). Note also that failover is not almost instantaneous.

We refer to this approach as the "N + 1" (single-spare) or "N + M" (multiple-spare) scenario.

Because this approach requires that you have a copy of the original server's data somewhere, this approach will not work if the original server's disk drive is damaged and there is no backup of the data. This N+M approach is most useful in the following situations:

1. You are using the spare node(s) to handle scheduled maintenance, not unexpected failures.

2. You have reliable backups that you can quickly copy to the spare server.

    a. You have backups on tape or on a RAID drive or some other safe location, or

    b. You are using Solid's SmartFlow technology, and you can copy or re-create enough of the data by reading from the SmartFlow "master" or SmartFlow "replica(s)" of the server that failed.

3. Individual pieces of data are not critical or are not unique.

    a. For example, if what you really need is the "computing horsepower" (load-spreading capability) rather than the specific data, then you may be able to meet your needs by copying a standard or "seed" database, or getting the data from clients, and then continuing to run.

    b. Similarly, if all the servers have approximately the same data and are responding almost entirely to "read" requests with few or no "write" requests — for example, if you are running a large number of servers that all use the same internet routing tables, or telephone directory information — then you can copy a useful database from any one of your computers.

4.14.2 Increasing Reliability: 2N + 1 Spare and 2N + M SpareScenarios

Normal Solid HotStandby operation is highly reliable. The odds of both the Primary and Secondary failing at nearly the same time are very low, provided that they use separate reliable power supplies. But suppose that you want to reduce even this risk, or suppose that the server that failed cannot be repaired rapidly? Ideally, when a Primary fails and you replace it with a Secondary (or when a Secondary fails), you'd like to have a

"new" Secondary that replaces the "old" Secondary so that you can continue to run with a complete pair of servers.

This situation is called the 2N + 1 Spare (or 2N + M Spares) scenario. You have N Primary servers, N Secondary servers, and at least 1 spare that will replace any Secondary that has failed or has been converted to a Primary. Spares are not dedicated to a particular server (or HSB pair of servers), and some configuration is required before the spare can replace the failed server.

4.14.3 How Solid HSB Supports The N+1 (N+M) and 2N+1 (2N+M) Approaches

You must make a spare server look like the server that it is replacing. Typically, this means:

1. You must copy data to the spare.
2. You must tell the spare to "listen" at the same network address as the server that it is replacing, or at another address that the client programs know to communicate through.
3. In addition, in the 2N+1 (2N+M) scenario, you must also tell the new Secondary server and the current Primary server how to communicate with each other, In other words, you must tell each of them the address to use to connect to the other.

Solid has two features to support these needs:

- Solid allows you to copy data to the spare server without shutting down the spare server.
- Solid allows you to dynamically set certain configuration parameters.

These are explained in more detail below:

1. Although Solid configuration parameters are normally set by shutting down the server, updating the `solid.ini` configuration file, and then re-starting the server, it is also possible to change some configuration parameters (such as the "com.listen" and "hotstandby.connect" parameters) by executing ADMIN commands similar to the following:

   ADMIN COMMAND 'parameter com.listen="tcp SpareServer1 1315"';

   ADMIN COMMAND 'hsb parameter connect "tcp srvr27 1316"';

   This means that a spare can be dynamically configured to take the place of another server without shutting down first. Similarly, a Primary can be told the Connect string of its new Secondary.

Caution

Executing these commands does NOT write the updated parameter values to the `solid.ini` file. Thus, to ensure that the server has the new values the next time it restarts, you should also update the `solid.ini` file, as well as execute the commands shown above.

Important

The spare server should be started with the **-x backupserver** command-line option so that it is ready to receive the netcopy from the Primary server. For more information about the **-x backupserver** option, see Section 4.8.6, “Creating a New Database for the Secondary Server”, and also see the explanation of command-line options in *solidDB Administration Guide*.

2. Solid's "netcopy" command allows you to copy a database to a server that is already up and running.

   a. Set the new value of the "connect" parameter:

      ADMIN COMMAND 'hsb parameter connect "tcp srvr27 1316"';

   b. Execute the netcopy command:

      ADMIN COMMAND 'hsb netcopy';

   c. Connect the current Primary with the new Secondary by executing the command:

      ADMIN COMMAND 'hsb connect';

Chapter 5. Using HotStandby with Applications

This chapter explains how applications should deal with failures and switchovers in HotStandby configurations.

5.1 Two Ways to Connect to HotStandby Servers

There are two ways to program applications in HotStandby environments. In addition to the Basic Connectivity, where the Client has to connect explicitly to each of the HSB servers, the Transparent Connectivity (TC) is offered whereby the Client enacts only one logical connection called the TC Connection. Both connectivity types are supported in the Solid ODBC and JDBC drivers. The connectivity type is selected by formulating a generalized connect string (Data Source Info) accordingly.

5.1.1 Transparent Connectivity

With this connectivity type, the application does not have to deal with connecting to any specific server, or to reconnect in the case of a failover. The application maintains a logical connection (handle) called a *TC Connection*. The connection handle is maintained over failovers and switchovers for as long as there is any server in the PRIMARY ACTIVE, PRIMARY ALONE or STANDALONE state, within the specified set of servers. At failovers and switchovers, the driver performs an internal operation called *connection switch*. The application is notified about the connection switch, because the application must reconstruct some of the session states (depending on the failure transparency level). With TC, read-only load can be balanced between the Primary and the Secondary server. Briefly, the Transparent Connectivity relieves the application from taking care of multiplicity of servers and their addresses.

Important

Solid tools, such as the solsql, do not support the TC connection.

5.1.2 Basic Connectivity

With Basic Connectivity, the application has to take care of connecting to each server of the HotStandby configuration separately, by using specific server addresses. If a failover happens, the active connection is lost, and the application has to reconnect to the new Primary server.

5.1.3 Choosing the Connectivity Type

The following compatibility matrix helps you choosing the connectivity type by indicating the supported feature against the selected connect info:

Table 5.1. Choosing the Connectivity Type

Feature	Standalone config.	HSB configuration
Basic Connectivity	Yes (BC Info)	Yes (BC Info)
Transparent Failover	No	Yes (TC Info)
Load Balancing	No	Yes (TC Info)

The principle is that TC info can be used in all configurations. The provided functionality is the highest the server configuration supports. On the other hand, if the goal is to use explicit Basic Connectivity, it can be used in all configurations as well.

5.2 Using the Transparent Connectivity

When using Solid Transparent Connectivity, the driver hides the existence of two HSB servers, to some extent, from the application. The driver offers a single logical TC Connection that is mapped to the internal active connection. In an ideal case, when both Primary and Secondary servers are running in the active state, the driver also maintains the *standby connection*, that is, the connection to Secondary. This connection will be set to the event wait mode, where it is ready to receive HSB state change events. Those events are the primary source of information on failovers and switchovers that the driver will use. In some cases (such as the Primary Alone operation), the standby connection will be missing, but the driver will try to enact it whenever possible. The standby connection is handled totally transparently to the Client. On the other hand, any occurrence of a connection switch, that is, changing the mapping of the TC Connection to an internal active connection, will be notified to the Client by way of a special error code.

5.2.1 Failure Transparency in TC

Failure transparency handles the masking of failures. Three levels are available:

1. NONE, which disables failure transparency. This is the default value.

2. CONNECTION, which preserves the server connection, that is, makes it unnecessary to reconnect in the case of failover or switchover.

3. SESSION, which preserves most of the session attributes having non-default values. Additionally, prepared statements are preserved. However, open cursors are closed, and on-going transactions are aborted. For the list of preserved session attributes see Section 5.2.7, “Programming for Connection Switch”.

The failure transparency level is set with the TF_LEVEL attribute of the TC Info.

5.2.2 Load Balancing in TC

The Transparent Connectivity driver uses two methods to direct the transaction load; one to handle read intensive load and the other to handle write intensive load. For the sake of load balancing, the logical TC Connection is mapped to a lower level server connection called Workload Connection. The workload connection may change over time and it is, normally, of no concern to the application. However, if this is necessary, there is a way to find out what is the current workload connection.

Static Load Balancing Configuration

The load balancing methods are:

1. PREFERRED_ACCESS=READ_MOSTLY. This method is for handling read intensive load. The read-only transactions can be executed at both the Secondary and Primary. If the parameter *`Cluster.ReadMostlyLoadPercentAtPrimary`* is set to zero, the read-only load is fully executed at the Secondary server.

2. PREFERRED_ACCESS=WRITE_MOSTLY. This is the default value. This method is for handling write intensive load. All the transactions are executed at the Primary server. This corresponds to the typical HotStandby operation.

With setting PREFERRED_ACCESS=READ_MOSTLY, the Primary server tells the driver which server to connect to for the workload connection. If the load is directed to Secondary, and a write operation is issued, a hand-over happens to Primary and the transaction is executed in the Primary server. After the transaction commit, the load is directed back to Secondary. If Secondary fails, the connection fails over from Secondary to Primary.

Additionally, a new load balancing configuration parameter is introduced. It allows to direct a certain amount of read-only load to Primary.

```
[Cluster]
ReadMostlyLoadPercentAtPrimary=<n>
```

where n = [0 ... 100]. The default is 50.

This parameter defines the percentage of the total read-mostly load directed at the Primary. Based on this value, an *Assigned Workload Server* is selected. By default, half of the connections use the Primary and half the Secondary. This is a preferable value for most mixed loads. If the value is set to zero, all the load is directed at the Secondary. This is suitable in cases where very read-intensive (or read-only) applications use PREFERRED_ACCESS=READ_MOSTLY and (in the same time) write intensive applications use PREFERRED_ACCESS=WRITE_MOSTLY.

Note

Load balancing operates only at the isolation level READ COMMITTED. If the server's isolation level (startup) default is set to a different value, the setting PREFERRED_ACCESS=READ_MOSTLY forces the isolation level of this session to READ COMMITTED. The Isolation level may be dynamically reset to a higher one, say REPEATABLE READ but then the load balancing is disabled.

Note

Load balancing is disabled if the session is set to Autocommit mode.

Dynamic Control of Load Balancing

If the assigned workload server is Secondary, it can be changed programmatically to Primary. At the session level, the workload connection server can be changed to Primary with the statements below:

- `SET WRITE`
- `SET ISOLATION LEVEL REPEATABLE READ`
- `SET ISOLATION LEVEL SERIALIZABLE`

The statement takes effect immediately, if it is the first statement of a transaction, or from the next transaction, otherwise.

At the transaction level, the following statements change the workload connection server to Primary for the time of one transaction:

- `SET TRANSACTION WRITE`
- `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ`

- `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE`

The affected transaction is the one that is started by using the statement, or the next one, in other cases. After the transaction has been executed at the Primary, the workload connection server is reverted to the assigned one for the session.

The effect of the SET [TRANSACTION] WRITE statement may be reverted with the statement SET [TRANSACTION] READ WRITE (SQL:1999). Also, the isolation level statements have the same effect:

- `SET ISOLATION LEVEL READ COMMITTED`

- `SET TRANSACTION ISOLATION LEVEL READ COMMITTED`

Failover Transparency with Load Balancing

When both Transparent Failover is set (*TF_LEVEL* is other than NONE) and load balancing is enabled (*PREFERRED_ACCESS=READ_MOSTLY*), the applied failover policy is the following:

1. Primary failure: all the load is directed to the new Primary being in the PRIMARY ALONE state.

2. Secondary failure: all the load is directed to the Primary (PRIMARY ALONE)

3. Connection break between the servers; the servers are in the PRIMARY ALONE and SECONDARY ALONE states: if there is an ongoing read-only transaction executing in the Secondary, it is also successfully committed in the Secondary. All the subsequent transactions are directed to the Primary (in PRIMARY ALONE).

When the normal hot-standby operation is resumed (with servers being in PRIMARY ACTIVE and SECONDARY ACTIVE states) the load is rebalanced between the Primary and the Secondary.

Note

Even with *TF_LEVEL=NONE* (no failure transparency), some rudimentary failover capability is available: failover from Secondary to Primary when the Secondary fails. All other failures result in a communication link failure. So, with *TF_LEVEL=NONE*, in most failure cases it is required that the application reconnects (with the same TC Info). To avoid reconnection, it is recommended that failure transparency is always enabled when load balancing is used.

5.2.3 Syntax of the TC Info

When using Solid Transparent Connectivity, the Client enacts only one logical connection called the TC Connection. This connection is specified in the TC Info explained in this chapter. TC Info enacts transparent failover and load balancing both HSB configurations.

The full syntax of the Solid TC Info is the following:

```
<Solid TC Info>::= {[<failure transparency level attribute>]
[<preferred access attribute>] <connect target list>} | <cluster info>

<failure transparency level attribute> ::= TF_LEVEL={NONE |
CONNECTION | SESSION}

<preferred access attribute> ::= PREFERRED_ACCESS={WRITE_MOSTLY |
READ_MOSTLY}

<connect target list>::= [SERVERS=]<connection string>[, <connection string > ...

<cluster info>::= CLUSTER <connect string>[, <connect string>...]
```

Additionally, the following abbreviations can be used.

Table 5.2. TC Info Abbreviations

Abbreviation	Corresponding Syntax
TF	TF_LEVEL
CON	CONNECTION
SES	SESSION
PA	PREFERRED_ACCESS
RM	READ_MOSTLY
WM	WRITE_MOSTLY
S	SERVERS

Failure transparency attribute

Failure transparency, represented by the TF_LEVEL attribute, takes care of masking of failures. Three levels are available:

1. NONE, which disables failure transparency. This is the default value.

2. CONNECTION, which preserves the server connection, that is, makes it unnecessary to reconnect in the case of failover or switchover.

3. SESSION, which preserves certain session attributes having non-default values. Additionally, prepared statements are preserved. However, open cursors are closed, and on-going transactions are aborted.

Load balancing attribute

The preferred access attribute (PREFERRED_ACCESS) indicates whether the load balancing is applied or not. Two levels are available:

1. WRITE_MOSTLY, where the workload is fully directed to Primary. This is the default value.

2. READ_MOSTLY, where the workload is directed (by default) to Secondary. The write transactions are handed over to the Primary

Finally, the Solid TC Info includes a list of server addresses. The driver will scan the list from left to right and try to find the Primary and Secondary servers. Therefore, the preferable configuration must be put at the beginning of the list. The rest of the list may contain some spare addresses that might be activated at some time, during the system lifetime. Keep the list short because, in error situations, it can take a long time before the error is returned to the application. The addresses are tried one by one, involving the login timeouts specified. The number of addresses in the list is unlimited.

If none of the attributes TF_LEVEL nor PREFERRED_ACCESS is specified (or TF_LEVEL=NONE), the connection behavior falls back to Basic Connectivity. If more than one connection string is given, the connection is established to the first server on the list that accepts the connection request.

The CLUSTER keyword can be used as a synonym for:

```
TF_LEVEL=SESSION PREFERRED_ACCESS=READ_MOSTLY SERVERS=
```

For example, the following TC Info:

```
    F_LEVEL=SESSION PREFERRED_ACCESS=READ_MOSTLY
    SERVERS=tcp srv1.acme.com 1315, tcp srv2.acme.com 1315
```

may be replaced with:

```
    CLUSTER tcp srv1.acme.com 1315, tcp srv2.acme.com 1315
```

Example 5.1. Client-side INI File

```
[Data Sources]
Cluster1=
   TF_LEVEL=SESSION
   PREFERRED_ACCESS=READ_MOSTLY
   SERVERS=
      tcp -c 1000 srv1.dom.acme.com 1315,
      tcp srv2.dom.acme.com 1315,
      tcp srv3.dom.acme.com 1316
```

Example 5.2. Connect String in ODBC

```
rc = SQLConnect(comHandle, "CLUSTER
      tcp -c 1000 srv1.dom.acme.com 1315,
      tcp srv2.dom.acme.com 1315,
      tcp srv3.dom.acme.com 1316", ...
```

5.2.4 TC Info Attribute Combinations

The following table summarizes the possible combinations of the TC Info attributes and presents the resulting connection capabilities:

Table 5.3. Possible Combinations of TC Info Attributes

PREFERRED_ ACCESS:	TF_LEVEL: Not specified or NONE	TF_LEVEL: CONNECTION	TF_LEVEL: SESSION
Not specified	• No failover or switchover support • No load balancing (Basic Connectivity)	• Transparent Failover (session state not preserved) • Transparent Switchover • Workload in Primary only • No load balancing	• Transparent Failover (session state preserved) • Transparent Switchover • Workload in Primary only • No load balancing

PREFERRED_ ACCESS:	TF_LEVEL: Not specified or NONE	TF_LEVEL: CONNECTION	TF_LEVEL: SESSION
WRITE_MOSTLY (default)	• No transparent failover support • Transparent Switchover • Workload in Primary only • No load balancing	• Transparent Failover (session state not preserved) • Transparent Switchover • Workload in Primary only • No load balancing	• Transparent Failover (session state preserved) • Transparent Switchover • Workload in Primary only • No load balancing
READ_MOSTLY	• No Transparent Failover support • Transparent Switchover • Workload in Secondary and Primary • Load balancing	• Transparent Failover (session state not preserved) • Transparent Switchover • Workload in Secondary and Primary • Load balancing	• Transparent Failover (session state preserved) • Transparent Switchover • Workload in Secondary and Primary • Load balancing

5.2.5 Handling TC Info Contradictions

The attributes of the TC Info may contradict the actual service made available. In those situations, the connection is granted, but the SUCCESS_WITH_INFO warning is issued. This is done in the following cases:

- PREFERRED_ACCESS is specified, but HSB is not enabled. Basic connectivity is enabled.
- TF_LEVEL is specified, but HSB is not enabled. Basic connectivity is enabled.

5.2.6 Enacting Transparent Connectivity in JDBC

In JDBC, Transparent Connectivity is enabled with two non-standard connection properties.

Failure transparency handles the masking of failures. It applies equally to both the HotStandby and Cluster Transparency modes. Failure transparency is enabled with the solid_tf_level connection property. Three levels are available:

1. NONE, which disables failure transparency. This is the default value.

2. CONNECTION, which preserves the server connection, that is, makes it unnecessary to reconnect in the case of failover or switchover.

3. SESSION, which preserves all the session attributes having non-default values. Additionally, prepared statements are preserved. However, open cursors are closed, and on-going transactions are aborted.

The preferred access attribute indicates the type node where the workload connection is made to. The preferred access attribute is enabled with the solid_preferred_access connection property. Two levels are available:

1. WRITE_MOSTLY, where the workload connection is made to the PRIMARY.

2. READ_MOSTLY, where the workload connection is made (by default) to the SECONDARY.

The list of server addresses is given as a part of the extended JDBC connect string:

```
conStr= "jdbc:solid://host_name:port [,host_name:port].../user_name/password";
```

The number of addresses in the address list is limited to 20.

Caution

When using Transparent Connectivity in JDBC, you have to take care of dropping the statement objects explicitly. The garbage collector will not detect unreferenced statement objects.

Example 5.3. Using Transparent Connectivity in JDBC

```
...
String conStr = "jdbc:solid://srv1.acme.com:1323,srv2-acme.com:1423/dba/dba";
Properties prop = new Properties();
prop.add("solid_tf_level", "1");
...
Connection c = DriverManager.getConnection(conStr, prop);
...
```

Connect Error Processing

When a connect request is issued for a TC Connection, it is considered successful if at least one applicable server is found and connected to. The server may be in one of the states: PRIMARY ACTIVE, PRIMARY ALONE, or STANDALONE. Otherwise, the connect effort is considered failed. The address list is scanned once.

There may be various reasons for the connect request to fail. Most of them are masked by the following error cases:

Table 5.4. Connect Request Errors

SQLSTATE	Native code	Message text and description
08001	25217	Client unable to establish a connection Description: The driver has used the TC connect info to find an applicable server and connect to it. The effort has failed due to one of he following reasons: • No host listed in the address list was found • A host was found but the login timed out • A host was found but the login was rejected • Hosts found but not in the PRIMARY/STANDALONE state
HY000	21307	Invalid connect info... Description: a syntax error is found in an elementary connect string or in the TC connect info (data source info).
HY000	21300	Protocol ... not supported. Description: the string "TC" in the beginning of the TC connect info is misspelled (or, an incorrect protocol name is given in the elementary connect string).

There are cases when the connection is accepted with a warning.

Table 5.5. Warnings

SQLSTATE	Native code	Message text and description
0100	25218	Connected to Standalone or Primary Alone server. An effort has been made to set any non-default value of TF_LEVEL or PREFERRED_ACCESS, and there

SQLSTATE	Native code	Message text and description
		is only one server available. On this case, neither failure transparency nor load balancing is available.

5.2.7 Programming for Connection Switch

Principles of Connection Switch Handling

A connection switch refers to a situation where the driver changes the active server connection. Generally, the reason for a connection switch is a failover to the Secondary server or a switchover between the servers. More specifically, a need for a connection switch is detected from one of the following events:

- Event from the Secondary server about the state change to PRIMARY ALONE (failover) or PRIMARY ACTIVE (switchover). This is the main (and the fastest) mode of performing the connection switch.
- Indication of the state change at Primary.
- Link failure on the active connection.
- Connection timeout on the active connection.

The driver executes the connection switch in two steps:

1. The need for the connection switch is detected. The driver returns the following connection switch error on a pending request, or the following request:

Table 5.6. Connection Switch Request

SQLSTATE	Native code	Message text and description
HY000	25216	Connection switch, some session context may be lost Description: The driver has discovered the need of the connection switch. The client is expected to issue a transaction rollback call, to finalize the connection switch. This error code and message will be received at each consecutive network request call until the rollback call is issued

2. The Client program issues a rollback command (ODBC: `SQLEndTran()` with SQL_ROLLBACK; JDBC: `Connection.rollback()`). If the rollback is successful, a new active connection has been mapped to the TC connection that may be used.

Note

The connection switch error may be returned on a few consecutive ODBC calls. Therefore, a provision must be made to always respond with a rollback to this error, on any ODBC network request. If this happens in the middle of a transaction, the transaction must be re-executed.

On the other hand, if a new active connection cannot be established, the following error code is returned:

Table 5.7. Communication Link Failure

SQLSTATE	Native code	Message text and description
08S01	14503	Communication link failure Description: The driver has failed to establish a new active connection. The TF connection is lost and the Client has to reconnect (using a Data Source Info) in order to continue.

After receiving the rollback call, the driver will use a few alternative ways of finding the new active connection. In the simplest case, it will use the standby connection for the purpose. If that connection is not in the right state, the driver will wait for two seconds for the proper event to arrive. If the event does not arrive, and in other cases, the driver will fall back to the address list in the TC connect info and will repeat the connect sequence iteratively for a maximum time of 10 seconds. If all the efforts fail, the driver returns the above error.

The effect of the error is that the connection is lost, as seen by the application. Any further request issued on that connection will result in the same error.

Preservation of Session State

When the connection switch is executed by the driver, some of the session context can be lost and the Client must reconstruct it. The amount of the preserved state is dictated by the Transparent Failover level, expressed with the TC Info attribute TF_LEVEL. Essentially, with the TF level CONNECTION, no state is preserved while, at the SESSION level, most of the session state is preserved. The preservation of the session state is implemented by caching the necessary data in the driver. The higher transparency level is achieved at the expense of the response time of the requests requiring caching, and increased memory usage in the driver.

Regardless of the TF level, the following holds in the case of failovers:

- The updates of the current transactions are lost (because of the transaction rollback)
- Open cursors and their positions are lost.

The following table summarizes the session state preservation.

Table 5.8. Session State Preservation

TF_LEVEL	Preserved state
CONNECTION	No session state is preserved.
SESSION	Prepared statements • The prepared states are preserved. The effects of the following statements are preserved: • SET CATALOG • SET SQL INFO • SET SQL SORTARRAYSIZE • SET SQL CONVERTORSTOUNIONS • SET SQL SORTEDGROUPBY • SET SQL { OPTIMIZEROWS | OPTIMISEROWS } • SET SIMPLEOPTIMIZERRULES • SET LOCK TIMEOUT <seconds> • SET OPTIMISTIC LOCK TIMEOUT • LOCK_TIMEOUT • SET IDLE TIMEOUT • SET STATEMENT MAXTIME • SET ISOLATION LEVEL • SET DURABILITY

TF_LEVEL	Preserved state
	• SET SAFENESS • SET SCHEMA • SET SQL JOINPATHSPAN • SET SYNC USER • SET SYNC MODE The following standard ODBC attributes are preserved • SQL_ATTR_ACCESS_MODE (SET READ ONLY, SET READ WRITE) • SQL_ATTR_CURRENT_CATALOG (duplicates SET CATALOG above) • SQL_ATTR_AUTOCOMMIT

Additional Proprietary ODBC attributes

The following read-only ODBC attributes are available for application programmers for any format of the TC Info.

- SQL_ATTR_PA_LEVEL

  (integer, Preferred Access level: 0=WRITE_MOSTLY, 1=READ_MOSTLY)

  The attribute indicating whether the load balancing is used or not.

- SQL_ATTR_TC_WORKLOAD_CONNECTION

  (string, server name of the workload connection)

  The current workload connection server; if queried before the Commit, the value indicates the server the transaction will be committed on.

- SQL_ATTR_TF_LEVEL

(integer, TF level: 0=NONE, 1=CONNECTION, 3=SESSION)

The failure transparency level)

- SQL_ATTR_TC_PRIMARY

  (string, Primary server connection string)

  There is always a value indicating the current Primary server.

- SQL_ATTR_TC_SECONDARY

  (string, Secondary server connection string)

  The value indicates the assigned workload sever if (i) PA=READ_MOSTLY and (ii) the Secondary is the designated workload server (this is the default). Otherwise, an empty string is returned.

Note

The proprietary ODBC attributes cannot be used with the Windows ODBC driver manager. If you need to use proprietary ODBC attributes in Windows, Solid ODBC driver import library (`solidimpodbca.lib` or `solidimpodbcu.lib`) has to be linked directly with the application.

5.3 Using the Basic Connectivity

With Basic Connectivity, the application has to take care of connecting to each server of the HotStandby or Cluster configuration separately, by using specific server addresses. If a failover happens, the active connection is lost, and the application has to reconnect to the new Primary server.

Example 5.4. Basic Connection without Transparency

```
Connect=tcp srv1.dom.acme.com 1315
```

5.3.1 Reconnecting to Primary Servers from Applications

Preparing Client Applications for HotStandby

Client programs that have lost their connection to the Primary must be able to reconnect to the new Primary server (the old Secondary). You must code client applications to be able to:

1. Recognize that Primary is not available for write transactions any more.

2. Connect to the other server or switch to using previously created connection.

3. Take into account whether the current (interrupted) transaction was lost/aborted and must be re-executed on the new Primary server.

Getting the Secondary Server Address

The easiest way to get the connection information for the Secondary database server is to use the **ADMIN COMMAND 'hotstandby cominfo'** command, which gives the connection information for the other server in the HSB pair. When your application first connects to Primary, the application can execute this command and store the result. Later, if Primary goes down, the application can use the stored information to connect to Secondary (new Primary).

Note that when the **cominfo** command returns a value, it does NOT imply that Primary and Secondary are currently connected. The "cominfo" command simply returns the value specified in the *Connect* parameter of the `solid.ini` configuration file, or the value most recently specified with the **hsb parameter connect** command. If you need to check the connect status between Primary and Secondary servers, you can use **ADMIN COMMAND 'hotstandby status connect'**.

Detecting HotStandby Server Failure in Client Applications

To use the HotStandby feature, applications must know when to switch from the failed Primary to the Secondary (new Primary) server.

There are a couple of possible ways to do this. The best way is to simply check the return codes from the functions that you call to see if you have received an error that indicates you should switch to the other server.

You may also monitor the states of the servers (for example, check the Primary server to see whether its state has changed to PRIMARY UNCERTAIN).

The errors that indicate you should try switching to another server include:

10013 == transaction is read only

10041 == database is read only

10047 == replication transaction aborted

11002 == disk full

11003 == configuration exceeded

14501 == operation failed

14502 == invalid rpc parameter

14503 == communication error

14506 == server is closed (for example, because it is currently the target of an HSB copy/netcopy operation)

14510 == comm write failed

14511 == comm read failed

14518 == connection broken

14519 == user thrown out (for example, because of some administrative operation)

14529 == operation timed out

20009 == session error, write operation failed

21306 == server not found, connect failed

21308 == connection is broken (write failed with code ...)

21318 == operation failed (unusual return code)

ODBC Applications

The following error message is returned to ODBC applications that cannot establish a connection (for example, due to an inoperable database server):

- SQLState = 08001 - Client unable to establish connection

In addition, the following Solid communication error message is produced:

- 21306 - Server *'server_name'* not found, connection failed.

If a connection fails (for example, due to a network failure) in between operations, such as executing queries and fetching results, the following error message is returned:

- SQLState = 08S01 - Communication link failure

JDBC Applications

The following error message is returned to JDBC applications that cannot establish a connection (for example, due to an inoperable database):

- SQLState = 08001 - Unable to connect to data source.

If a connection fails (for example, due to a network failure) in between operations, such as executing queries and fetching results, the following error message is returned:

- SQLState = 08S01 - Communication link failure

Note

ODBC and JDBC use different error messages for the same error code (08001).

Switching the Application to the New Primary

After the application detects that it cannot send transactions to the "old Primary" server, the application must poll the old Primary and old Secondary servers until it finds a server that is in PRIMARY ACTIVE, PRIMARY ALONE, or STANDALONE state. Polling is accomplished by having the application attempt to connect to the servers and check the status of the servers when the connection is established.

When the connect is successful, the client can request the server state by using SQL function HOTSTANDBY_STATE, which is described in section Using Function HOTSTANDBY_STATE later in this chapter.

Caution

After the switch, all open database objects, such as prepared statements, open cursors and transactions, are no longer active. Thus, you must initialize these objects again. Also, if you were using Temporary Tables or Transient Tables (BoostEngine features), the tables will be empty on the new Primary.

Using Function HOTSTANDBY_CONNECTSTATUS

To verify connect status information when reconnecting to a Primary server from an application, you can use the `HOTSTANDBY_CONNECTSTATUS` function. This function is equivalent to the administrative command **hotstandby status connect**.

The function has no arguments and returns one of the following status values:

Table 5.9. HOTSTANDBY_CONNECTSTATUS Status Values

Status	Description
CONNECTED	The connection is active. This status is returned from both the Primary and Secondary servers.

Status	Description
CONNECTING	The Primary server is connecting to the Secondary server. This status is returned from both the Primary and Secondary servers.
CATCHUP	The Primary server is connected to the Secondary server, but the transaction log is not yet fully copied. This status is returned from both the Primary and Secondary server.
BROKEN	The connection is broken. This status is returned from both the Primary and Secondary servers.

Using Function HOTSTANDBY_STATE

To implement application polling of the Primary and Secondary servers, you can use the `HOTSTANDBY_STATE` function. This function is equivalent to the administrative command **hotstandby state**. It allows the application request the current HotStandby state when it is connected to the server.

Note

This function has no arguments. For a description of each possible state that this function may return, see Section 4.11, “Verifying HotStandby Server States”.

Example 5.5. Sample Pseudo-Code

An application, whether or not it is HSB-enabled, should have error handling that allows the application to replay a failed/aborted transaction.

In a non-HSB environment, a transaction may be aborted because of a concurrency conflict (optimistic tables) or deadlock (pessimistic tables). The application must catch these error situations and either automatically retry the transaction or ask interactive user to re-execute the transaction.

If your application already has code to handle failed/aborted transactions, then it is relatively easy to extend this code to make use of HSB.

In a very simplified example, the application pseudo-code with proper error handling for a non-HA-aware application handling looks something like this:

```
BEGIN TRANSACTION
EXECUTE APPLICATION LOGIC
PREPARE & EXECUTE STATEMENTS
COMMIT TRANSACTION
IF ERROR OCCURRED
```

```
      IF ERROR == concurrency conflict or deadlock
            GO TO BEGIN TRANSACTION
      END IF
      other error handling
END IF ;
```

Improving the above application to make it HA-aware is very simple. You must add code so that the application can:

- connect to either of the two servers instead of only one; and
- in the case of an error, find the server that is currently in one of the following states: PRIMARY ACTIVE, PRIMARY ALONE or STANDALONE.

The pseudo-code should look similar to the following:

```
BEGIN TRANSACTION
EXECUTE APPLICATION LOGIC
PREPARE & EXECUTE STATEMENTS
COMMIT TRANSACTION
IF ERROR OCCURRED
      IF ERROR == server unavailable for write transactions
            FIND CURRENT PRIMARY SERVER
            GO TO BEGIN TRANSACTION
      END IF
      IF ERROR == concurrency conflict or deadlock
            GO TO BEGIN TRANSACTION
      END IF
      IF ERROR == something else
            other error handling
      END IF
END IF
```

The logic to find the current primary server is also very simple. Just check the current state of both servers (try to re-connect if necessary) and if either of them is PRIMARY ACTIVE, PRIMARY ALONE or STANDALONE, set that server as the current primary. If neither server meets that criterion, wait awhile and retry checking the current server states.

5.3.2 Re-Connecting to Secondary Servers

In some cases, you may want to connect to the current Secondary (if it is up). Applications can submit read-only queries to the Secondary server; this can sometimes help you balance the workload across your servers.

An application can only connect to Secondary databases in the read-only mode. Note that a client can connect to the Secondary server (only in read-only mode) by using the following parameter values in the *HotStandby* section of the `solid.ini` configuration file in these servers:

- *Connect* parameter in the Primary server
- *Listen* parameter in the Secondary server

As mentioned earlier, you can also use the command

ADMIN COMMAND 'hotstandby cominfo';

to get the connection information for a server's partner. Thus, if you are connected to the current Primary server, you can get the address of the current Secondary by using the cominfo query.

5.3.3 SmartFlow Data Distribution Requirements

Any node of a SmartFlow data distribution system can be made highly available with the Solid CarrierGrade option.

When the master and replica databases of a SmartFlow system are synchronizing data, the synchronization occurs between the Primary servers of the database server pairs. In other words, the Primary of the Master communicates with the Primary of the Replica. See Figure 2.3, “HotStandby with Master and Replica Server Scheme”.

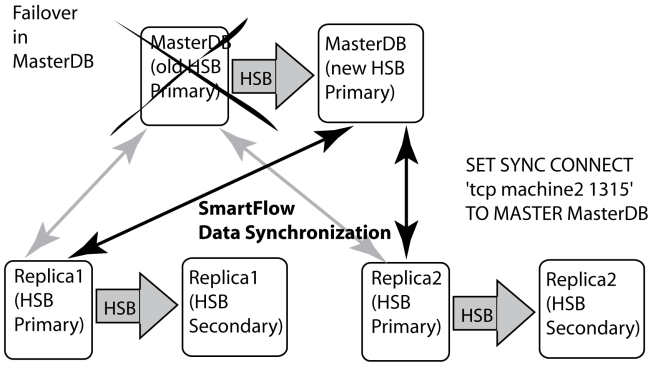
A database server may fail over to its Secondary server at any point of time, including when the database server is synchronizing data with another server using SmartFlow data synchronization. If the failover occurs during synchronization, executing the synchronization message stops and the process must be resumed after the failover. For details about how to resume synchronization after an error has occurred, please refer to *solidDB SmartFlow Data Replication Guide*.

If a server containing a SmartFlow master database is made fault tolerant with Solid CarrierGrade, the replicas of the master database must know the connect strings to both master servers. To do this, execute the following statement in each of the replica databases.

```
SET SYNC CONNECT 'connect_string_to_server_1, connect_string_to_server_2'
TO MASTER <master_nodename>
```

In the diagram below, the gray arrows represent the original connections to the original Primary server, while the black arrows represent the new connections to the new Primary (old Secondary) server. The alternate connection is used if the synchronization with the old primary server fails.

Figure 5.1. Master failover



If the server using Solid CarrierGrade is a server containing a replica database AND if the master server uses remote procedure calls (CALL procedure AT node_name) to run procedures at the replica, for example to initiate the synchronization, the master server must be informed about the connect strings to both of the replica servers. Typically a master server uses remote procedure calls to initiate synchronization with a replica database. To inform the master about the connect strings to the replica server pair, execute the following statement in the master database.

SET SYNC CONNECT 'connect_string_to_server_1,

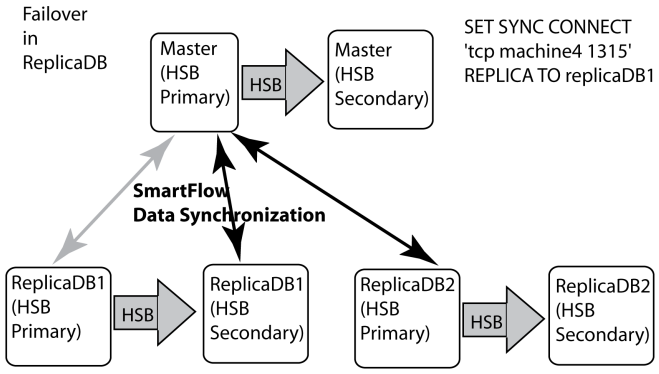
connect_string_to_server_2' TO REPLICA <replica_nodename>

Alternatively, you can save the statement in the replica server and propagate it to master the next time that you synchronize. In that case, use the following statement:

SAVE SET SYNC CONNECT 'connect_string_to_server_1,

connect_string_to_server_2' TO REPLICA <replica_nodename>

Figure 5.2. Replica failover



If the master server never executes remote procedure calls in the replica, then the above statement is not needed.

Chapter 6. Configuring HotStandby

This chapter describes the parameter settings for implementing and maintaining HotStandby functionality. This description supplements the information contained in the chapter "Configuring solidDB" of *solidDB Administration Guide*.

Parameters are grouped according to section categories in the `solid.ini` configuration file. When you are using the Solid CarrierGrade option, you are required to configure the *`[HotStandby]`* section of the `solid.ini` configuration file. If you are using a watchdog application to monitor your HotStandby servers, you may also include a *`[Watchdog]`* section. (The *`[Watchdog]`* section is optional.)

You can change configuration parameters in the following ways:

- Manually editing the configuration file `solid.ini`. Note that the server reads the `solid.ini` file only when it starts, and therefore any changes to the `solid.ini` file do not take effect until the next time that the server starts.
- Entering the following commands in Solid SQL Editor solsql:

```
ADMIN COMMAND 'parameter <section_name.param_name>=<value>';
```

- For example

```
ADMIN COMMAND 'parameter hotstandby.2SafeAckPolicy=2';
ADMIN COMMAND 'parameter com.listen="tcp sf_server 1315"';
```

Caution

Note that when you use an ADMIN COMMAND to change a parameter, the changes to some, but not all, parameters will take effect immediately. For more details, see Appendix A, *Configuration Parameters*.

6.1 Configuring solidDB for HotStandby

The `solid.ini` file (at both the Primary and Secondary nodes) contains the parameters that are necessary to set up a HotStandby system.

6.1.1 Defining Secondary and Primary Node Configuration (Com Section)

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

A Primary or Secondary server may use one or multiple network names. Note that all components of network names are case insensitive.

Example 6.1. Entry in the `solid.ini` File

Primary node:

```
[Com]
;The Primary server listens to the network with this name
Listen = tcp 1320
```

Secondary node:

```
[Com]
;The Secondary server listens to the network with this name
Listen = tcp 1321
```

For more information about the *Listen* parameter, see *solidDB Administration Guide*.

6.1.2 Defining Timeouts Between Applications and Servers (Com Section)

This section describes how to configure Application Read Timeout and Connect Timeout settings by using either the `solid.ini` *Connect* parameter or the connect string of the `SQLConnect` function for ODBC. These timeout values apply to the server's connections with client applications, including Solid SQL Editor (solsql), Solid Remote Control teletype (solcon), and the watchdog application.

Application Read TimeOut Option

This option detects failures in low level network RPC read operations. Its timeout setting applies to the read in the physical network, which works only for the TCP/IP protocol. This RPC read timeout (called connection timeout in ODBC and JDBC) value is configured in connect and listen strings using the following option:

```
TCP -rnumber_of_milliseconds [machine_name] port_number
```

To specify this in the `solid.ini` file, use the *`Connect`* parameter in the *`[Com]`* section of `solid.ini`. For example:

```
[Com]
;Set RPC read timeout to 1000 milliseconds (one second)
Connect=TCP -r1000 1313
```

The default value for RPC read timeout is 60000 milliseconds (60 seconds). Value zero (0) sets an indefinite timeout.

If you are using ODBC, you specify the timeout setting in the connect string of the `SQLConnect` function. For example:

```
SQLConnect (hdbc, "TCP -r1000 1313", SQL_NTS,
"dba", SQL_NTS, "dba", SQL_NTS);
```

In the example above, the constant SQL_NTS indicates that the previous string (servername, username, or password) was passed as a standard Null-Terminated String.

Note

For client applications, such as the watchdog, it is convenient to provide RPC read timeout (called also connection timeout) in the *`connect`* parameter using the **-r** option. Otherwise certain network failure types may cause indefinite waits.

Note

The *`Connect`* parameters in the *`[Com]`* section, *`[Watchdog]`* section, and *`[Hotstandby]`* sections are all for different purposes. Make sure that you edit the correct one.

Specifying -C Option in the Connect Parameters

You can specify the connect timeout (called also login timeout) value in the *Connect* parameter used in the *[Com]* and *[Watchdog]* sections of the solid.ini file. This connect timeout works only for the TCP/IP protocol. The syntax is:

```
Parameter = tcp -cnumber-of-milliseconds [machine name] port_number
```

where Parameter is *Connect* or *Listen*.

For example:

Application node:

```
[Com]
;The server listens to port 1320, and the Connection timeout is 1000 ms.
Listen = tcpip -c1000 1320
```

If no value is provided for the connect timeout, the server uses the operating system-specific default value.

Note

For client applications, such as the watchdog, it is convenient to provide the connect timeout value in the *Connect* parameter using the **-c** option. Otherwise certain network failure types may cause a long wait before the failure is detected.

6.1.3 Transaction Durability

DurabilityLevel

The parameter *DurabilityLevel* applies to both HotStandby and non-HotStandby servers. This parameter has three different values, which correspond to "relaxed", "adaptive", and "strict" durability.

Adaptive durability is used only with HotStandby. Adaptive durability means:

- If Primary and Secondary are connected and operating normally (if they are in PRIMARY ACTIVE and SECONDARY ACTIVE states, respectively), the server uses relaxed durability;
- In all other situations (e.g. PRIMARY ALONE, STANDALONE, etc.), the server uses strict durability.

For an explanation of the differences between "strict" and "relaxed" durability, or for more information about the *DurabilityLevel* parameter, see *solidDB Administration Guide*.

6.2 Configuring HotStandby-Specific Parameters

At both the Primary and Secondary nodes, the solid.ini file contains the *[HotStandby]* section to specify HotStandby-specific configuration parameters.

6.2.1 Defining Primary and Secondary HotStandby Configuration

The minimum set of solid.ini configuration parameters that you must set to enable HotStandby is:

- *HSBEnabled*. This parameter turns HSB on or off.
- *Connect*. This parameter defines the network name used to define either the Primary or Secondary server. The network name is the protocol and server name that the Primary server uses to connect to the Secondary server, or vice versa. (Strictly speaking, the *Connect* parameter is not required to be in the solid.ini file. You may start the server without this parameter and then use an ADMIN COMMAND to specify the Connect string. If the Connect string is not set, then the server can run only in the states that do not require a connection, i.e. PRIMARY ALONE, SECONDARY ALONE, and STANDALONE.)
- *LogEnabled*. If this parameter is set, it should be set to "yes". Note that this parameter is in the *[Logging]* section, not the *[HotStandby]* section, of the solid.ini file.

Example 6.2. Partial solid.ini Files

Primary Node

```
[HotStandby]
HSBEnabled = yes
;The Primary server connects to the Secondary server
;using the following connect string.
Connect = tcp machine2 1321
[Logging]
LogEnabled=yes
```

Secondary Node

```
[HotStandby]
HSBEnabled = yes
```

```
;The Secondary server connects to the Primary server
;using the following connect string.
Connect = tcp machine1 1320
[Logging]
LogEnabled=yes
```

Note

If the `solid.ini` file does not contain a *`[HotStandby]`* section, or does not contain *`HSBEnabled=Yes`* in the *`[HotStandby]`* section, then the server starts as a non-HotStandby server and HotStandby replication is not used.

6.2.2 Setting HotStandby Server Wait Time to Help Detect Broken or Unavailable Connections

A HotStandby server uses timeout parameters to control how long it will wait before concluding that an existing connection is broken or a new connection cannot be established.

These parameters are:

- *`HotStandby.PingTimeout`*
- *`HotStandby.PingInterval`*
- *`HotStandby.ConnectTimeout`*

A HotStandby server that is in the PRIMARY ACTIVE state or the SECONDARY ACTIVE state will change to PRIMARY UNCERTAIN, PRIMARY ALONE, or SECONDARY ALONE if it tries to contact the other server and receives no reply within a specified amount of time.

To control how long the server waits, you can:

- Set the *`PingTimeout`* parameter to specify the amount of time that the server should wait before changing to the PRIMARY UNCERTAIN state.
- Set the *`PingInterval`* parameter to specify the interval between the "pings" the server is sending to watch the health of the other server.
- Set the *`ConnectTimeout`* parameter to specify the amount of time that the server should wait when trying to establish a new connection to the other server (for example, in an **ADMIN COMMAND 'hot-standby connect'** operation).

Each of these parameters is described in more detail below.

PingTimeout and PingInterval Parameters [HotStandby]

A "ping" operation is essentially an "Are you there?" question sent by one database server to another. (Some networking software also has a "ping" operation, but the Solid *PingTimeout* configuration parameter in the *[HotStandby]* section applies only to solidDB server pings, not general network pings.) When this parameter is set, both the Primary and Secondary HotStandby servers "ping" each other at regular intervals.

The optional *PingTimeout* and *PingInterval* parameters in the *[HotStandby]* section have the purpose to control the ping operation:

- *PingTimeout* specifies how long a server should wait before concluding that the other server is down or inaccessible. Default is 4000 (4 sec.)
- *PingInterval* specifies the interval, in milliseconds, between two pings. Default is 1000 (1 sec.)

For example, if the *PingInterval* is 10 seconds, then the servers will ping each other after every 10 seconds. If *PingTimeout* is 20 seconds and one server (S1) does not hear from the other (S2) within 20 seconds, then S1 will conclude that S2 is down or inaccessible. Server S1 will then switch to another state, e.g. from "PRIMARY ACTIVE" to "PRIMARY UNCERTAIN".

If the values of the above parameters are different, the precedence take values set in Primary during execution of the "hsb connect" command. The values will not change during switchovers. However, they can be changed dynamically with the ADMIN COMMAND "parameter" command.

If *PingTimeout* is set to zero, pinging is disabled.

Ping requires little overhead and a solidDB server is set up to respond quickly to pings. The server does not need to wait until it finishes processing the current SQL query, for example, before responding to a ping. You can set the *PingInterval* value to a fairly short interval, such as a second, or even less.

If it is important that you quickly detect failover when a server goes down, then set the *PingTimeout* value to a relatively time. However, shorter values also mean a higher chance for "false alarms". If your network has a lot of traffic and thus causes delays before a ping response is received, then you may need to set the *PingTimeout* to a large value to avoid false alarms.

ConnectTimeout Parameter [HotStandby]

In some network implementations, a connect operation may not respond for an indefinite period of time. One possible reason is that the remote machine is a known node, but is unavailable during the connect attempt. By specifying a connect timeout value, you can set the maximum time in seconds that a HotStandby connect operation waits for a connection to a remote machine.

The *ConnectTimeout* parameter (which is useful only on certain platforms) is only used with certain administration commands. These are:

- **hotstandby connect**
- **hotstandby switch primary**
- **hotstandby switch secondary**

You set the connect timeout value in milliseconds using the *ConnectTimeout* parameter in the *[HotStandby]* section of the `solid.ini` file. The units are milliseconds. The default is 0, which means no timeout. You can set it to a different value, for example:

```
[HotStandby]
 ; Set ConnectTimeout to 20 seconds (20000 milliseconds).
ConnectTimeout=20000
```

6.2.3 Defining a Name and Location for HotStandby Database Copy Operation

The optional *CopyDirectory* parameter in the *[HotStandby}* section defines the name and location of the directory that the HotStandby copy operation copies to. The HotStandby copy operation is specified with the command:

```
ADMIN COMMAND 'hotstandby copy [directory_name]';
```

For example, on a Microsoft Windows system, the parameter and value might look like:

```
[HotStandby]
CopyDirectory=c:\Solid\DatabaseEngine4.1\secondary\dbfiles
```

This parameter has no default value, so if the directory is not specified in the `solid.ini` file, it must be provided in the copy command. If you provide a relative path for the *CopyDirectory* parameter, the path will be relative to the directory that holds the Primary server's `solid.ini` file.

This parameter is not needed if you do the HotStandby database copy operations using the **ADMIN COMMAND 'hotstandby netcopy'** command. Of these two alternatives, **netcopy** provides more flexible functionality and is thus the recommended command.

6.2.4 Defining Primary Server Behavior During a Secondary Failure

You can use the *AutoPrimaryAlone* parameter in the *[HotStandby]* section to control whether the Primary server automatically switches to PRIMARY ALONE state or stays in PRIMARY UNCERTAIN state after losing contact with the Secondary server.

If *AutoPrimaryAlone* is set to Yes, then when Primary loses contact with Secondary, Primary will automatically switch to the PRIMARY ALONE state, which allows it to continue accepting transactions. If *AutoPrimaryAlone* is set to No, then when Primary loses contact with Secondary, Primary will automatically switch to the PRIMARY UNCERTAIN state.

By default, *AutoPrimaryAlone* is set to No.

```
[HotStandby]
AutoPrimaryAlone =  No
```

The PRIMARY UNCERTAIN state prevents Primary from accepting new transactions or committing the currently active ones. Primary will not switch to PRIMARY ALONE state until the Watchdog or System Administrator tells it to.

If *AutoPrimaryAlone* is set to No, the server can be set to the PRIMARY ALONE state by executing the **ADMIN COMMAND 'hotstandby set primary alone'** command. Note that this command does not change the value of *AutoPrimaryAlone* in the configuration file.

If you change the default to Yes, the Primary server's state changes from PRIMARY ACTIVE to PRIMARY ALONE rather than to PRIMARY UNCERTAIN.

Important

If you are running a watchdog program on the same machine where the Secondary server resides, be sure to set the parameter *AutoPrimaryAlone* to no. In this situation, setting *AutoPrimaryAlone* to no is crucial because it prevents the potential error of having two primary servers. Primary may be in the PRIMARY ALONE state, and the watchdog at server failure could switch Secondary to a PRIMARY ALONE state. This same error can also occur if a user happens to set the old Secondary server to become the new Primary. For more information about dual primaries, see Section 4.3.1, “Network Partitions and Dual Primaries”.

6.3 Performance Tuning

6.3.1 Tuning Replication Performance with Safeness and Durability Levels

The performance of data replication during normal operation depends on the setting of the Durability Level and Safeness Level. Additionally, when 2-safe replication is used, the type of the 2-Safe Acknowledgment Policy affects the latency time, as perceived by the application. For more information, see Section 2.3, “How Does HotStandby Affect Performance”.

6.3.2 Tuning Netcopy Performance (General Section)

The hotstandby netcopy command allows the Primary's database to be copied to a remote Secondary. This command is also used to copy a database from a Primary server to a Secondary server when one or both servers are diskless. The connect string used to connect to the Secondary server for the netcopy is specified in the *`[HotStandby]`* section of `solid.ini`.

The Primary database files are copied through the network link. For more details on netcopy, read Section 4.8.5, “Copying a Primary Database to a Secondary Over the Network”.

The following parameters in the *`[General]`* section of the `solid.ini` file allow for improved netcopy performance.

BackupBlockSize Parameter [General]

The *`BackupBlockSize`* parameter in the *`[General]`* section of the `solid.ini` file is used to tune the performance of netcopy (and the performance of backup, of course) by increasing or decreasing its block size when it copies the Primary database files to the Secondary server. As a general rule, larger block size means faster netcopy/backup, but at the cost of possibly slowing down the server's response time to other requests while the netcopy/backup is being done.

By default, the *`BackupBlockSize`* parameter is set to 64K. You can set it to a different value, for example:

```
[General]
BackupBlockSize = 32K
```

or

```
[General]
BackupBlockSize = 32768
```

Note that the minimum value for *BackupBlockSize* is the server block size and the maximum value is currently 1GB ("M" and "K" suffixes are supported; for example, 32K and 1M). The value of *BackupBlock-Size* should be a multiple of the server's database block size.

Tuning Database Catchup Performance

When a failed Secondary server is back in service and connected to Primary, HotStandby continues sending the Primary's HotStandby transaction log file contents to the Secondary node in an automated process known as HotStandby database catchup. The *CatchupSpeedRate* parameter in the *[HotStandby]* section of the `solid.ini` file is used to tune the performance of the database catchup by adjusting how much of the server's time is spent on catchup vs. servicing current client database queries.

If *CatchupSpeedRate* is assigned a value of 90, this means that the server will spend approximately 90% of its time on catchup and about 10% of its time responding to user queries. For example:

```
[HotStandby]
CatchupSpeedRate = 50
```

The higher the number, the faster catchup will perform, but the more it will impact other activities, such as user queries. By default, *CatchupSpeedRate* is set to 70.

6.4 Configuring Parameters for a Watchdog

A watchdog is an separate program for monitoring and controlling Primary and Secondary servers. The watchdog monitors both HotStandby servers and switches their states when necessary. This alleviates the need for a database administrator to monitor the servers.

Solid provides a sample watchdog that you can use as a basis for building a custom watchdog that meets your needs. If you are using this sample program, you need to configure a *[Watchdog]* section in the Solid configuration file (`solid.ini`), which resides in the current working directory of the watchdog. If the watchdog is running in the same directory as the Primary or Secondary server, then you will have only one `solid.ini` file, which will be shared by the server and the watchdog. If the watchdog is running in a separate directory, then the watchdog will have its own `solid.ini` file.

Although the sample watchdog provided by Solid uses a `solid.ini` file, not all watchdogs need such a file. If you write your own watchdog, you may choose whether or not to use a configuration file, and whether or not to name that file "`solid.ini`".

Following are the parameters for implementing the watchdog application on the Secondary server or a separate machine. Read Chapter 7, *Monitoring HotStandby Server Pairs with a Watchdog Application* for more details on using these watchdog parameters.

6.4.1 Watchdog Section

The `solid.ini` file for the watchdog contains a *`[Watchdog]`* configuration section to specify watchdog-specific parameters.

The parameters that are documented here apply to the sample C-language watchdog program supplied by Solid, which is designed to read a `solid.ini` configuration file and search for the *`[Watchdog]`* section. If you write your own watchdog program, you may create and use any parameters you wish; you are not required to use the ones documented here, and you are not limited to the ones documented here. You may even write a custom watchdog program that does not use a `solid.ini` file at all.

Note that the Solid sample watchdog program, like the servers, does not care whether the section name (*`[Watchdog]`*) is upper case, lower case, or mixed case.

Connect1 Parameter [Watchdog]

The *`Connect1`* parameter in the *`[Watchdog]`* section enables the watchdog application to connect to one of the two servers. This is a required parameter that defines the protocol and network address for the *`Connect1`* server.

For example:

```
[Watchdog]
;The watchdog application connects to the server using the
;TCP/IP protocol at server port 1313 with following connect
;string.
connect1 = tcp primarymachine 1313
```

Connect2 Parameter[Watchdog]

The *`Connect2`* parameter in the *`[Watchdog]`* section enables the watchdog application to connect to the other server (the one not specified by *`Connect1`*). This is a required parameter that defines the protocol and network address for the *`Connect2`* server.

For example:

```
[Watchdog]
```

```
;The watchdog application connects to the server using the
;TCP/IP protocol at server port 1313 with following connect
;string.
Connect2 = tcp secondarymachine 1313
```

Username1 and Password1 Parameters [Watchdog]

The *Username1* and *Password1* parameters in the *[Watchdog]* section are optional. They are used to specify the username and password that is authorized for using the connect1 server. For example:

```
[Watchdog]
Username1 = dba
Password1 = dba
```

If (for security reasons) these parameters are not specified in the `solid.ini` configuration file, the watchdog will prompt for

```
username1
```

and

```
password1
```

when the watchdog is started.

Username2 and Password2 Parameters [Watchdog]

The *Username2* and *Password2* parameters in the *[Watchdog]* section are optional. They are used to specify the username and password that is authorized for using the connect2 server. For example:

```
[Watchdog] Username2 = dba
Password2 = dba
```

If (for security reasons) these parameters are not specified in the `solid.ini` configuration file, the watchdog will prompt for

```
username2
```

and

```
password2
```

when the watchdog is started.

NumRetry Parameter [Watchdog]

The *NumRetry* parameter in the *[Watchdog]* section lets you specify the number of times that the watchdog attempts to connect to a Secondary or Primary server before concluding that the server has failed or become inaccessible. The default for this parameter is 0. This parameter is optional. For example:

```
[Watchdog]
NumRetry = 3
```

The retries are in addition to the original try. If number of retries is set to 3, then the total number of attempts is 4. The retries are immediate. The watchdog does not wait a certain number of milliseconds before retrying.

PingInterval Parameter [Watchdog]

The *PingInterval* parameter in the *[Watchdog]* section lets you specify how frequently the watchdog should check the servers to make sure that they are still accessible and are still connected to each other. To detect server failure, the watchdog sends the hotstandby status connect command to both Primary and Secondary servers at regular intervals. The amount of time between each check is specified by the *PingInterval* parameter. The units of this parameter are milliseconds. For example:

```
[Watchdog]
PingInterval = 2000
```

This parameter is optional. The default for this parameter is 1000 milliseconds (1 second).

Note that the *PingInterval* parameter for the watchdog is different from the *PingTimeout* parameter for the servers.

AutoSwitch Parameter [Watchdog]

If the *AutoSwitch* parameter is set to yes, the watchdog automatically does the following:

1. If the Secondary server fails, then the watchdog tells the Primary server to switch to PRIMARY ALONE state (rather than stay in PRIMARY UNCERTAIN) state.

2. If the Primary server fails, then the watchdog automatically sends the command 'hsb switch primary' to switch the original Secondary to be the new Primary and 'hsb set primary alone' to start that new Primary.

The default for this parameter is Yes. This parameter is optional. For example:

```
[Watchdog]
AutoSwitch = NO
```

WatchdogLog Parameter [Watchdog]

The WatchdogLog parameter in the *[Watchdog]* section lets you specify the file name of the watchdog log. The watchdog log is created in the current working directory. It is used to record watchdog messages, alerting administrators of the need to issue watchdog commands. The default for this parameter is `Watchdog.log`. This parameter is optional.

For example:

```
[Watchdog]
WatchdogLog = Watchdog.log
```

6.5 Configuration File Examples

Below is a sample excerpt of the Solid Configuration file (`solid.ini`) for the first HotStandby server:

```
[Com]
; The first server listens to the network with this
; name
Listen = tcp 1320
[HotStandby]
HSBEnabled=yes
; The first server connects to the second server
; using the following connect string.
Connect = tcp machine2 1321
AutoPrimaryAlone=No
```

Below is a sample excerpt of the Solid Configuration file (`solid.ini`) for the second HotStandby server:

```
[Com]
; The second server listens to the network using the following
; connect string.
Listen = tcp 1321
[HotStandby]
HSBEnabled=yes
```

```
; The second server connects to the first server
; using the following connect string.
Connect = tcp machine1 1320
AutoPrimaryAlone=No
```

Chapter 7. Monitoring HotStandby Server Pairs with a Watchdog Application

In this chapter we explain possible failure scenarios and what commands should be issued to recover from those scenarios. Although these commands may be issued by either a human administrator or a software program, we will assume for simplicity that the commands are issued by a software program called a "watchdog". A watchdog is an separate program that monitors Primary and Secondary servers, and gives commands to change those servers" states when necessary. We recommend that you use a watchdog so that you can determine when the Primary or Secondary server itself has failed or when just the communication link between these servers is down.

solidDB Development Kit includes the C-language source code for a sample watchdog program. The sample watchdog program uses its own separate `solid.ini` configuration file to store information such as the *`PingInterval`* (described later). If you write your own watchdog, you may choose whether to use a configuration file or whether to simply hard-code the values you want.

Note

The watchdog application does not use the CarrierGrade Connectivity to connect to the servers. The reason is that the watchdog has to communicate with each server explicitly.

7.1 How the Watchdog Application Works

The sample Solid watchdog application notifies you when the Primary server is down. In normal mode, the watchdog checks the connection status of servers using the **hotstandby status connect** command in both Primary and Secondary servers. The watchdog performs this check between servers at regular intervals. The interval time is set with the *`PingInterval`* parameter in the watchdog's `solid.ini` configuration file.

The watchdog reaches the conclusion that there is a problem in the HotStandby system when it receives no response from the Primary or Secondary node or both nodes after a given number of polling attempts. The number of attempts is set in the

`NumRetry`

parameter in the watchdog configuration file (`solid.ini`).

The watchdog also observes whether the Primary server and the Secondary server are connected to each other. If the Primary or Secondary server returns a successful connect status to the watchdog, this means the Primary and Secondary are still connected. If it returns an error, on the other hand, then the Primary and Secondary are no longer connected.

If the *AutoSwitch* parameter in the watchdog configuration file is set to YES, then the watchdog is also responsible for automatically switching server states in the event of a Primary failure. For example, when the Primary server is down, the watchdog switches the Secondary server to make it the new Primary and put it in PRIMARY ALONE state. If the *AutoSwitch* parameter is set to NO, the watchdog does not change the server state itself, but instead writes a message to the watchdog log to notify the user to switch server states.

To continue monitoring, the watchdog switches to failure mode, which means it continuously keeps checking failed servers for a working connection.

7.1.1 Failure Mode

When the sample watchdog program knows that HotStandby Primary and Secondary servers are connected, the watchdog stays in normal mode. If one of the servers fails, or if the communication link between these servers fails, then the sample watchdog program will take some course of action. If the action fails to connect the servers, the watchdog application goes into failure mode.

In the sample C-language watchdog application provided in solidDB Development Kit, after the watchdog enters failure mode, the watchdog waits for the system administrator to fix the problem with the Primary and Secondary servers. If, in the meantime, a second failure occurs, the watchdog does not handle the failure. This limitation in the watchdog sample application is deliberate. There are situations where a series of failures and even seemingly appropriate responses can cause the error of having two Primary servers (either in PRIMARY ALONE or STANDALONE states). This is especially true if there are brief failures in the network, but no failures in the database servers themselves. An example that produces two Primary servers is provided in Section 7.1.2, “Coding a Watchdog for Multiple Failures”.

During failure mode, the sample watchdog application polls both the Primary and Secondary servers. When it is able to connect to both servers, it sends the **hotstandby state** command to both servers to see whether it can communicate with them and to see which state each of them is in.

Once the sample watchdog is able to communicate with both servers, it will decide what to do next based on the `solid.ini` parameter DualSecAutoSwitch. If *DualSecAutoSwitch = Yes* and both servers are secondary, then the Watchdog will automatically select one of the two secondaries to be a new primary and switch it to primary. If *DualSecAutoSwitch = No* then the system administrator must switch one server to be the primary. Note that *DualSecAutoSwitch* applies whether the watchdog is in "normal" mode or "failure" mode.

7.1.2 Coding a Watchdog for Multiple Failures

There are two ways to handle multiple failures in your own watchdog application. You can:

- After each failure (and automatic response by the watchdog), require manual (human) intervention to check the situation. Manual intervention may require actions, such as restarting a server, or fixing a network problem. This is the approach that the sample watchdog application uses because it reduces the risk of having two Primary servers.
- Write a watchdog application that can handle multiple failures over time.

  This method does run the risk of having two Primary servers, as shown in the following example.

Example 7.1. Dual Primaries

In this example, Server1 is initially the Primary and Server2 is initially the Secondary.

1. A network failure occurs and Server1 becomes inaccessible.
2. The watchdog switches Server2 from SECONDARY to PRIMARY ALONE.
3. A second network failure occurs, and Server2 becomes inaccessible.
4. The first network failure is repaired, and Server1 becomes accessible again.
5. The watchdog, seeing that Server1 is accessible and Server2 is not, switches Server1 to PRIMARY ALONE.
6. The second network failure is fixed and Server2 becomes accessible again.
7. At this point, both Server1 and Server2 are in the PRIMARY ALONE state.

For more about the dangers of dual primaries, see the warning in Section 4.3.1, “Network Partitions and Dual Primaries”.

7.2 Using the Sample Watchdog Application

If you are using the watchdog application, be sure to configure the `solid.ini` file in the watchdog computer. For details, read Section 6.4, “Configuring Parameters for a Watchdog”.

Initially, you should start the watchdog after both servers are up and connected. To start the sample watchdog, go to the current working directory of the watchdog and at the prompt, issue the command:

watchdog

If you have not specified the usernames and passwords for connect1 and connect2 servers (capable of serving as Primary and Secondary) in the `solid.ini` file, the watchdog prompts you for them.

Once started, the watchdog pings both servers to check which one is Primary. The watchdog remains in normal mode unless it detects a server failure after the number of retry attempts is exceeded. If a failure occurs after the watchdog sends the last retry attempt to the server, then the watchdog switches to failure mode. Once both the Primary and Secondary servers are up and re-connected, the watchdog switches to normal mode.

7.3 HotStandby Failure Scenarios and Watchdog Actions

This section describes how a typical watchdog program should work in specific failure scenarios that are commonly encountered. The scenarios are in the context of either a server failure or a broken communication link between the Primary and Secondary server, or between one of the servers and the watchdog.

7.3.1 Primary is Down

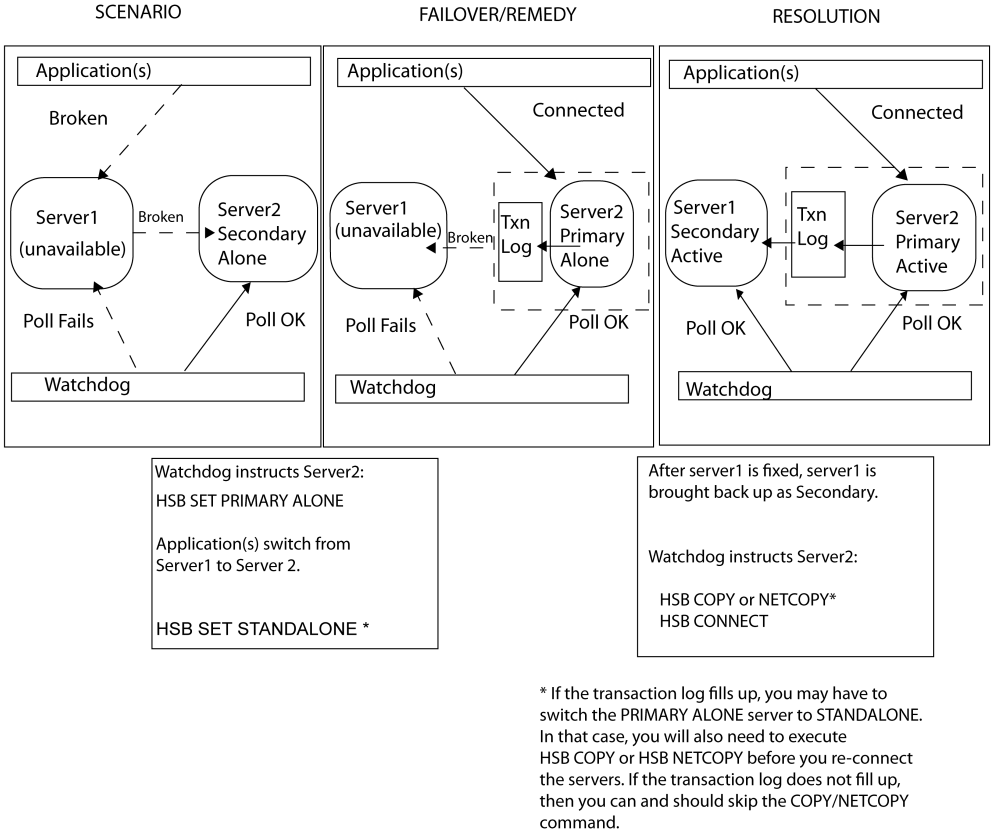
Scenario

All connections to the Primary server are broken.

Remedy

When the Primary is down, switch the Secondary to be the new Primary and set the new Primary to the PRIMARY ALONE state. Later, the old Primary can become a new Secondary.

Figure 7.1. Primary is Down Scenario and Remedy



Symptoms

Applications cannot connect to the Primary. Also, the watchdog poll fails at the Primary. The HSB state of the secondary server is SECONDARY ALONE.

How to Recover

To allow the "HotStandby" (Secondary server) to replace the Primary, do the following:

1. Set the new Primary server to PRIMARY ALONE state by using the command:

   ADMIN COMMAND 'hotstandby set primary alone';

2. Reconnect applications to the new Primary.

3. Start using applications.

4. Fix and start the old Primary server as new Secondary server.

5. If necessary, copy the database from the new Primary to the new Secondary using command:

   ADMIN COMMAND 'hotstandby netcopy';

   Read Section 4.8, “Synchronizing Primary and Secondary Servers” for details.

6. Reconnect the new Primary to the new Secondary using the command:

   ADMIN COMMAND 'hotstandby connect';

Further Scenarios

None.

7.3.2 Secondary is Down

Scenario

All connections to the Secondary server are broken. This may be caused either by a failure in the Secondary, or by a failure in the network that makes it impossible for either the Primary or the Watchdog to communicate with the Secondary server. In this section, we will refer to the Secondary as failing, but in fact the problem may be with either the Secondary or the network.

Remedy

The standard remedy is to switch the Primary server to the PRIMARY ALONE state. After the Secondary is up again, synchronize it with the Primary.

Upon finding a problem with the connection to the Secondary server, the Primary server:

1. Suspends any open transactions, neither committing them nor rolling them back (the Primary does not send an error message — or a "success" message — to the client); and

2. Automatically switches its own state from PRIMARY ACTIVE to PRIMARY UNCERTAIN.

Typically, after making sure that the secondary server is unavailable, the watchdog will switch the Primary from PRIMARY UNCERTAIN to PRIMARY ALONE. After the Primary is switched to PRIMARY ALONE

state, it can continue accepting transactions and saving them to send to the Secondary. Later, when the Secondary is brought back up, the Secondary can be sent the transaction log so that it can "catch up" to the Primary.

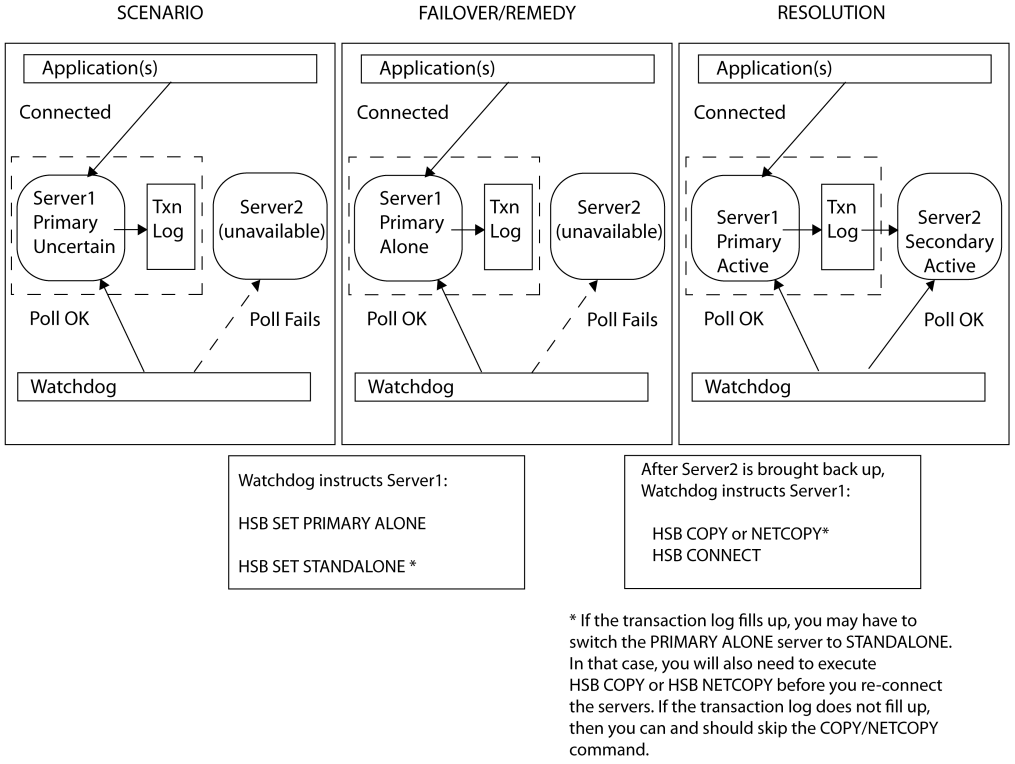
The Primary commits the open transactions after the Primary is set to PRIMARY ALONE state. To avoid the possibility that the Primary will commit the transactions when the Secondary hasn"t, the transactions are kept in the transaction log, as though they had never been sent to the Secondary. When the Secondary is brought back up and starts catching up, the Primary sends that transaction log, and the Secondary checks each of the transactions. If any of the transactions are duplicates (that is, if the Secondary already committed that transaction before the Secondary failed), then the duplicate transactions are not re-executed on the Secondary.

The watchdog or system administrator must be careful in choosing whether to bring the Primary to PRIMARY ALONE state, or choose an alternative action. If the watchdog or system administrator chooses a different action than switching the Primary to PRIMARY ALONE state, she must take into account that the Secondary and Primary may not have the same data i.e. they may not both have rolled back the transaction. It is possible that the failed Secondary actually committed the data and crashed after committing the data but before sending the confirmation to the Primary, while the Primary never committed. In this situation, the secondary may actually be "ahead" of the Primary rather than behind it.

As always, the watchdog or administrator also must be careful not to allow both servers to go into PRIMARY ALONE state at the same time.

The diagram below is divided into three frames. The first frame shows the scenario, which is that the Primary and watchdog have lost contact with the Secondary. The next frame shows how to respond to keep your system working until the problem can be complete solved. The third frame shows the final state after the problem has been solved — that is, after the broken server has been fixed, or after communications have been restored.

Figure 7.2. Secondary is Down Scenario and Remedy



Symptoms

The watchdog poll fails at the Secondary. The state of the primary server is either PRIMARY ALONE or PRIMARY UNCERTAIN.

How to Recover

To allow the Primary server to continue to receive transactions, operating independently of the Secondary server, do the following:

1. If the Primary server is in the PRIMARY UNCERTAIN state, then set the Primary server to PRIMARY ALONE using the command.

   ADMIN COMMAND 'hotstandby set primary alone';

2. After the Secondary server has been repaired and re-started and/or Secondary's network connections have been re-established, check the state of the Primary server using the command.

   ADMIN COMMAND 'hotstandby state';

3. If the state of the Primary server is PRIMARY ALONE, then reconnect the Primary to the Secondary using the command

   ADMIN COMMAND 'hotstandby connect';

4. If the state of the Primary server has earlier been changed to STANDALONE, then:

   1. Copy the database from the new Primary to the Secondary using

      ADMIN COMMAND 'hotstandby netcopy';

   2. Read Section 4.8, “Synchronizing Primary and Secondary Servers” for details.

5. Reconnect the Primary to the Secondary using the command:

   ADMIN COMMAND 'hotstandby connect';

Further Scenarios

If an application receives error message 10047 or 14537 from the Primary:

- Try to connect to the Secondary to check if its state was switched to new Primary.
- If its state is not one of the Primary states (i.e. PRIMARY ACTIVE or PRIMARY ALONE), see the scenario in Section 7.3.1, “Primary is Down”.

7.3.3 Watchdog is Down

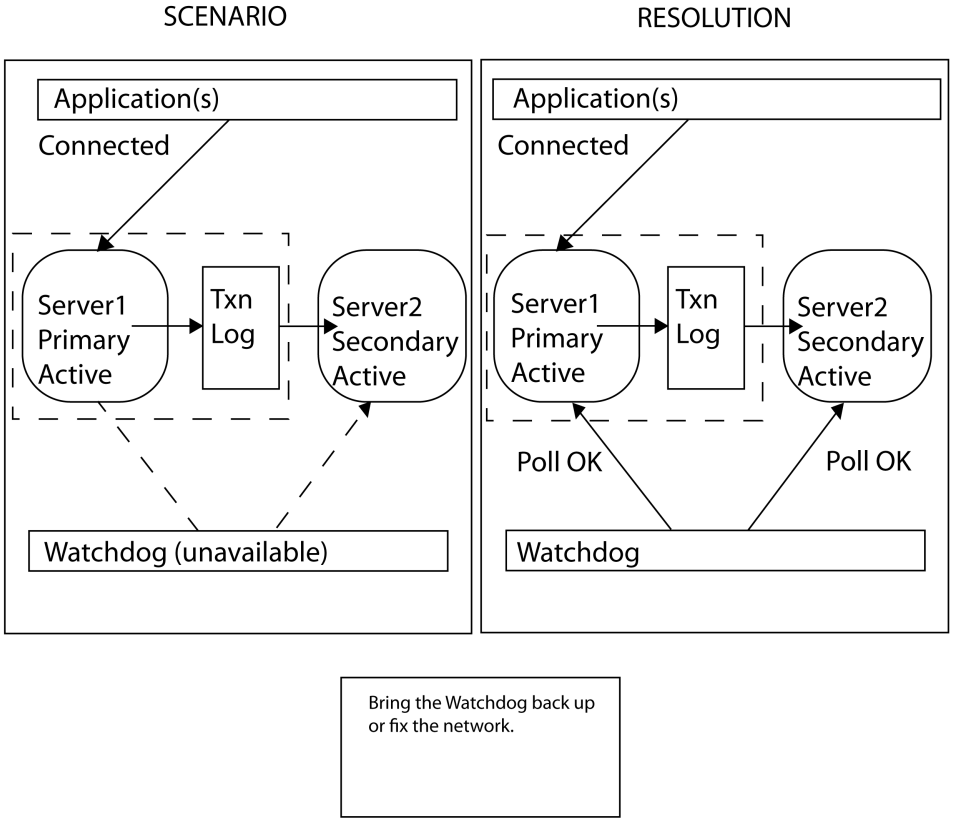
Scenario

All connections to the watchdog are broken.

Remedy

Manual intervention is required. When the watchdog is brought up, be sure to check the Primary and Secondary servers to confirm their states.

Figure 7.3. Watchdog is Down Scenario and Remedy



Symptoms

The watchdog process is down or network connections from the watchdog to both servers are unavailable.

How to Recover

1. Allow the Primary and Secondary servers to continue normal operations.
2. Once the watchdog is brought up, have it check the state of each server with the command:

   ADMIN COMMAND 'hotstandby state';

Further Scenarios

If the servers have changed states and one server is no longer in service, refer to the applicable scenario in this section for instructions.

7.3.4 Communication Link Between Primary and Secondary Is Down

Scenario

The connection between the Primary and Secondary server is broken.

The Primary will switch itself to PRIMARY UNCERTAIN state. (If *AutoPrimaryAlone* is set to Yes, then the server will switch itself to PRIMARY ALONE state.)

Note

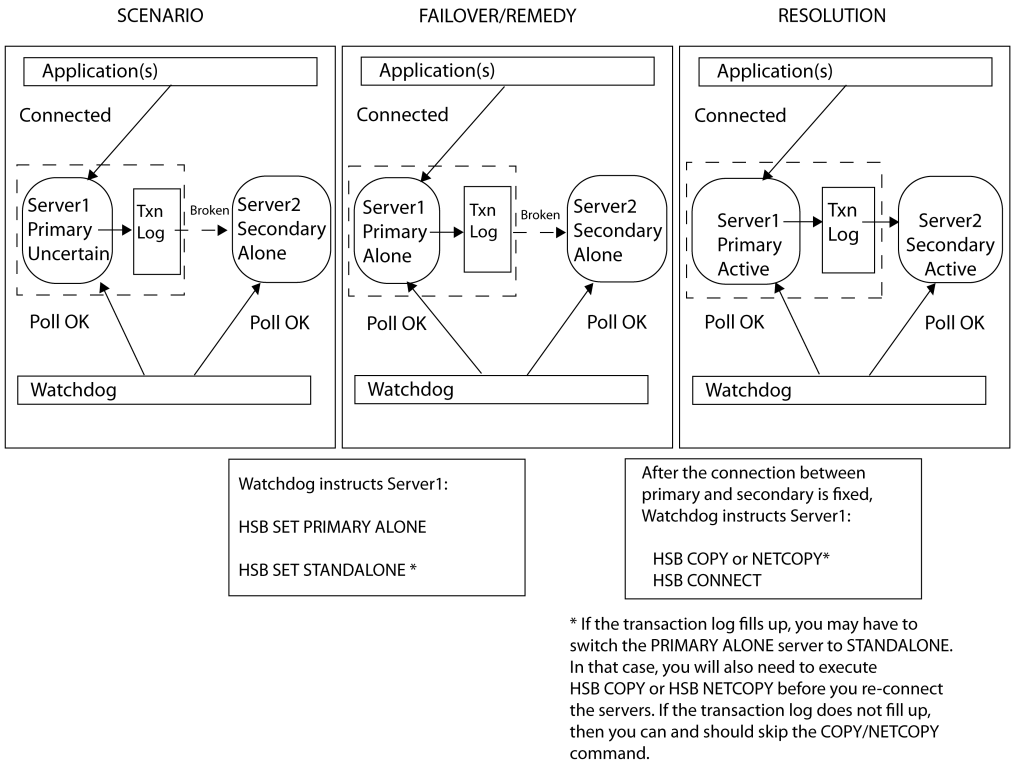
If the Primary server sends a commit message to the Secondary and then detects the failure of the Secondary, the Primary server relies on the watchdog or the administrator to indicate how the Primary server is to proceed. This is because the Primary server is unable to detect whether the transaction was committed or rolled back in the Secondary before the Secondary server failed.

Until the Primary server receives a command from the watchdog or the administrator, it no longer accepts transactions. At this stage, in order for the Primary server to continue operations, the watchdog or administrator can set the Primary server to PRIMARY ALONE state.

Remedy

The Primary server can continue operations even when its link to the Secondary server is down. If the Primary is not already in PRIMARY ALONE state, then switch the Primary to the PRIMARY ALONE state. Once the link between the Primary and Secondary is restored, synchronize the databases.

Figure 7.4. Broken Link Between Primary and Secondary Scenario and Remedy



Symptoms

The Primary server has no Secondary connected and the state is PRIMARY UNCERTAIN or PRIMARY ALONE.

How to Recover

1. Fix the network connection between the Primary and Secondary servers.

2. Check the state of the Primary server using the command:

   ADMIN COMMAND 'hotstandby state';

3. If the state of the Primary server is PRIMARY ALONE, reconnect the Primary to the Secondary using the command:

ADMIN COMMAND 'hotstandby connect';

4. If the state of the Primary server is STANDALONE, then:

   a. Copy the database from the Primary to the Secondary. Read Section 4.8, “Synchronizing Primary and Secondary Servers” for details.

      Before using the

      ADMIN COMMAND 'hotstandby netcopy';

      command, be sure that the Secondary is up and running and is ready to receive the netcopy. Also, make sure that you set the Primary server's state to PRIMARY ALONE.

   b. Reconnect the Primary to the Secondary using the command

      ADMIN COMMAND 'hotstandby connect';

Further Scenarios

If an application receives error message 10047 or 14537 from the Primary:

- Try to connect to the Secondary to check if it is switched as the new Primary.
- If the old Secondary is not switched as the new Primary, see scenario in Section 7.3.1, “Primary is Down”.

7.3.5 Communication Link Between Watchdog and Primary Is Down

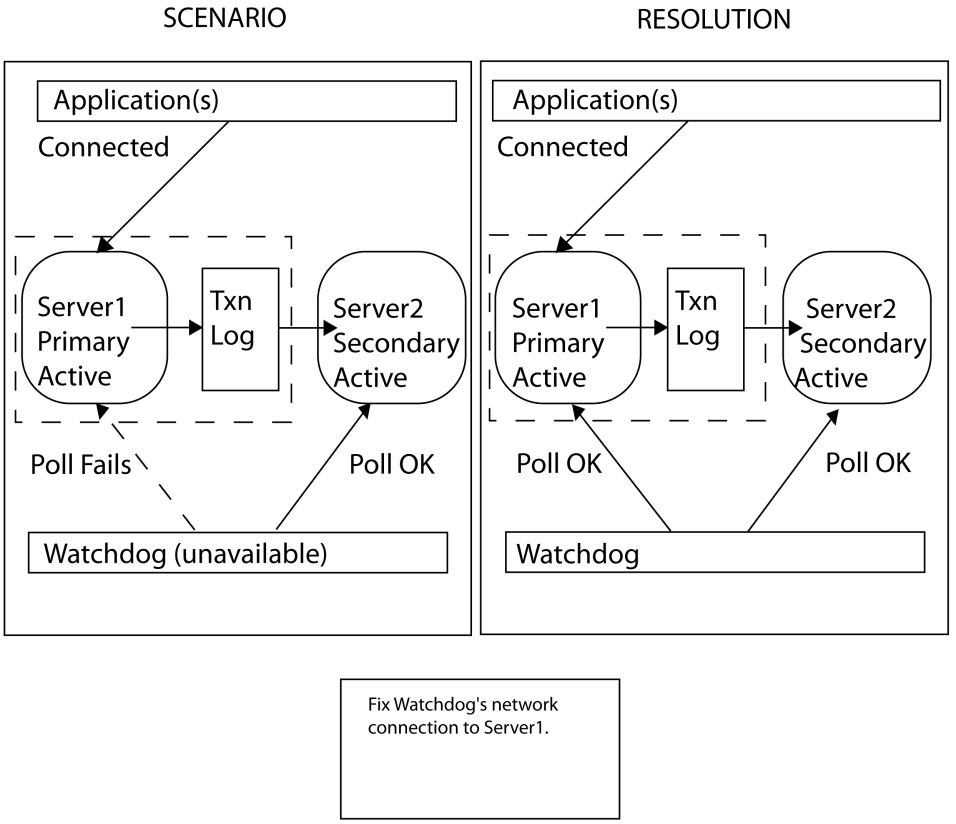
Scenario

The connection between the watchdog and the Primary server is broken.

Remedy

The Primary and Secondary servers can continue operations even when the watchdog link to the Primary server is down. When the watchdog link to the Primary is fixed, be sure to check the states of the Primary and Secondary servers.

Figure 7.5. Broken Link Between Watchdog and Primary Scenario and Remedy



Symptoms

The watchdog poll fails at the Primary server. However, the secondary server state is reported to be SECONDARY ACTIVE. This means that the primary server is very probably OK and that the watchdog has merely lost contact with the Primary.

How to Recover

1. Allow the Primary and Secondary servers to continue normal operations.
2. Fix the network connection between the watchdog and the Primary server.
3. Once the network is connected, have the watchdog check the states of each server with the command:

ADMIN COMMAND 'hotstandby state';

Further Scenarios

If the states of the servers have changed and one server is no longer in service, refer to the applicable scenario in this section for instructions.

7.3.6 Communication Link Between Watchdog and Secondary Is Down

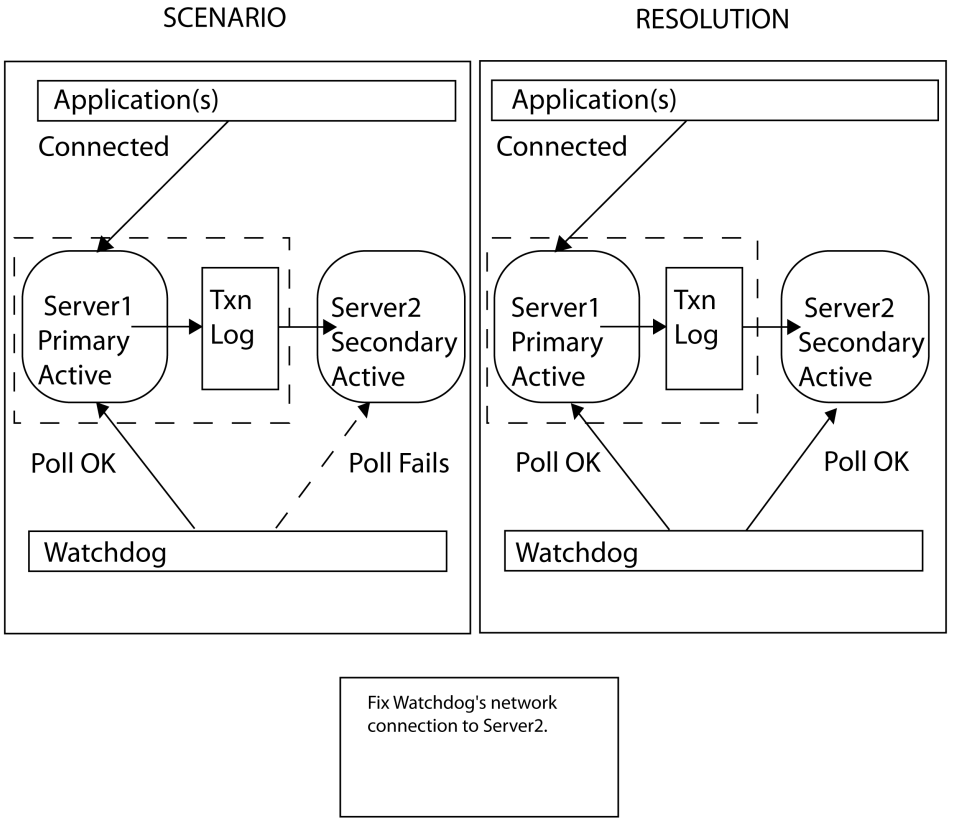
Scenario

The connection between the watchdog and the Secondary server is broken.

Remedy

The Primary and Secondary servers can continue operations even when the watchdog link to the Secondary server is down. When the watchdog link to the Secondary is fixed, be sure to check the Primary and Secondary servers to confirm their states.

Figure 7.6. Broken Link Between Watchdog and Secondary Scenario and Remedy



Symptoms

The watchdog poll fails at the Secondary server.

How to Recover

1. Allow the Primary and Secondary servers to continue normal operations.
2. Fix the network connection between the watchdog and the Secondary server.
3. Once the network is connected, have the watchdog check the state of each server with the command:

   ADMIN COMMAND 'hotstandby state';

Further Scenarios

If the servers states have changed and one server is no longer in service, refer to the applicable scenario in this section for instructions.

7.3.7 Communication Links Between Watchdog and Primary, and Between Primary and Secondary, Are Down

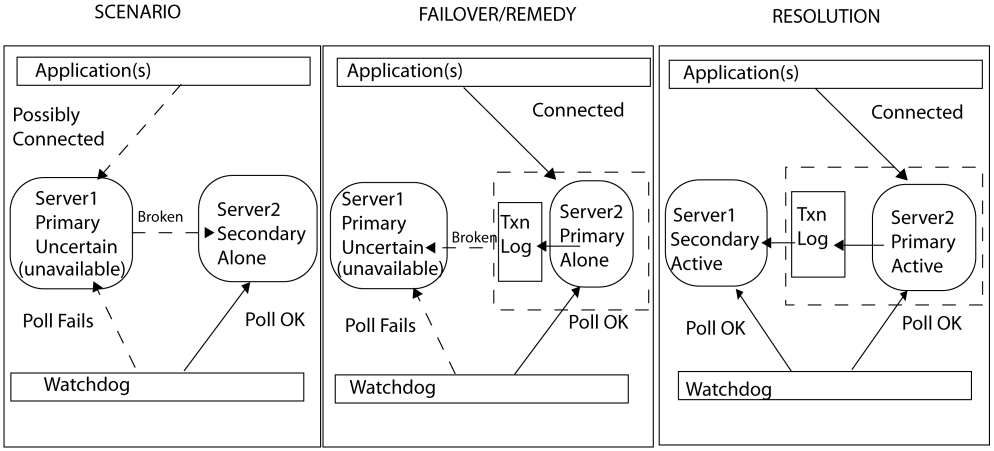
Scenario

The connections between the watchdog and the Primary server, and between the Primary server and Secondary server, are broken.

Remedy

For the watchdog to continue monitoring the Primary server, switch the Secondary server to be the new Primary and set this new Primary to the PRIMARY ALONE state. Later, set up a new Secondary server and synchronize it with the Primary.

Figure 7.7. Broken Link Between Watchdog and Primary, and between Primary and Secondary, Scenario and Remedy



Note that Server1's role is Primary Uncertain. However, from the watchdog's point of view, Server1 is unavailable, not Primary Uncertain.

Watchdog instructs Server2:

HSB SET PRIMARY ALONE

Application(s) switch from Server1 to Server 2.

HSB SET STANDALONE*

Note that both servers believe they are primary. If a program or an administrator (manual intervention) were to switch Server1 from Primary Uncertain to Primary Alone, then we would have two active primaries, both of which could be updating data, and the differences would not be resolvable.

After network connections are fixed,Watchdog instructs Server1:

HSB SWITCH SECONDARY

Watchdog instructs Server2:

HSB COPY or NETCOPY*
HSB CONNECT

* If the transaction log fills up, you may have to switch the PRIMARY ALONE server to STANDALONE. In that case, you will also need to execute HSB COPY or HSB NETCOPY before you re-connect the servers. If the transaction log does not fill up, then you can and should skip the COPY/NETCOPY command.

Symptoms

The watchdog poll fails at the Primary server. The Secondary server and Primary server have lost their connections to each other; therefore Server2 is in the state SECONDARY ALONE, and the Primary (if it can be contacted) will report that its state is PRIMARY UNCERTAIN or PRIMARY ALONE.

The beginning of this scenario assumes that application(s) are possibly connected to the old Primary. However, since the old Primary is in the PRIMARY UNCERTAIN state, the application(s) are unable to perform updates. Note that it is also possible that the application(s) connected to Server1 may have lost their communication link and no longer know that the old Primary exists.

How to Recover

To allow the hot standby server (the Secondary server) to replace the Primary, do the following:

1. If the old Primary is in the PRIMARY UNCERTAIN state or is cut off from the applications as well as the Secondary, then set the Secondary server to PRIMARY ALONE state using the command:

   ADMIN COMMAND 'hotstandby set primary alone';

2. Reconnect applications to the new Primary.

3. Fix the network or the broken connections to the old Primary.

4. Check the server states. Both servers must now be running.

5. If the new Primary is in STANDALONE state (for example, because the new Primary's transaction log filled up while the connections were being fixed):

   1. Set the new primary to PRIMARY ALONE state using the command

      ADMIN COMMAND 'hotstandby set primary alone';

   2. Copy the database from the new Primary to the new Secondary. Read Section 4.8, “Synchronizing Primary and Secondary Servers” for details.

6. If the new Primary is in PRIMARY ALONE state:

   1. Switch the old Primary to be the new Secondary server using the command:

      ADMIN COMMAND 'hotstandby switch secondary';

7. Reconnect the new Primary to the new Secondary using the command:

   ADMIN COMMAND 'hotstandby connect';

Further Scenarios

If an application receives error message 10047 or 14537 from the new Primary:

- Try to connect to the old Secondary to check if it has switched to be the new Primary.
- If the old Secondary is not switched to be the new Primary, re-execute the transaction with the original Primary in PRIMARY ALONE state.

7.3.8 Communication Links Between Watchdog and Secondary, and Between Primary and Secondary, Are Down

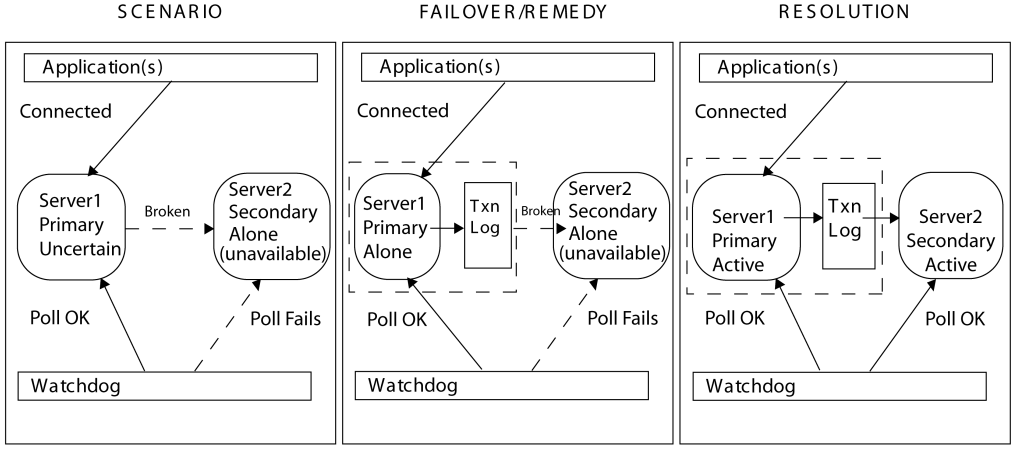
Scenario

The connection between the watchdog and the Secondary server, and the connection between the Primary server and Secondary, server are broken.

Remedy

The Primary server can continue operations even when its links to the Secondary server and the watchdog are down. Switch the Primary server to the PRIMARY ALONE state, if it is not already in PRIMARY ALONE state. Later, when the Secondary is up again, synchronize it with the Primary.

Figure 7.8. Broken Link between Watchdog and Secondary and between Primary and Secondary Scenario and Remedy



Symptoms

The watchdog poll fails at the Secondary; the Primary server has no Secondary connected and switches to state PRIMARY UNCERTAIN or PRIMARY ALONE.

How to Recover

1. Try to fix the connections.

2. After the connections are fixed, check the state of the Primary server using the command ADMIN COMMAND 'hotstandby state'.

3. If the state of the Primary is STANDALONE:

1. Ensure that both servers are running.
2. Set the state of the Primary server to PRIMARY ALONE using command:

   ADMIN COMMAND 'hotstandby set primary alone';
3. Copy the database from the Primary to the secondary using command:

   ADMIN COMMAND 'hotstandby netcopy';

   Read Section 4.8, “Synchronizing Primary and Secondary Servers” for details.

4. Reconnect the Primary to the Secondary using the command

   ADMIN COMMAND 'hotstandby connect';

Further Scenarios

If an application receives error message 10047 or 14537 from the Primary:

- Try to connect to the Secondary to check if it switched to be Primary.
- If the Secondary is not switched as the new Primary, re-execute the transaction with the original Primary in PRIMARY ALONE state.

Chapter 8. Upgrading Your Server by Using HotStandby

8.1 Cold and Hot Migration

When you update the version of the software, we are talking migration. For a highly-available system like Solid HotStandby, the migration may be "cold" or "hot".

Cold migration is the traditional way to migrate. You shut down the whole system (both servers) and restart with new software and configuration data.

Solid's High Availability design allows you to upgrade your solidDB servers without taking your entire system off-line for the amount of time required to upgrade the servers. One server can keep operating while the other server is being upgraded. We call this "hot migration".

Caution

Although your entire system will not be down, users/applications may have to disconnect from one server and connect to the other server.

8.2 Migration between HSB-Compatible Versions

When the versions of the software are HSB-compatible, each of the servers may be a of a different version and still they may be able to communicate with each other. For example, all releases of any major version of Solid are HSB-compatible.

8.2.1 Cold Migration

The cold migration is trivial. You shut down the whole system and restart the servers with the new software with the same roles as before the shutdown.

8.2.2 Hot Migration

The basic outline for hot migration is:

1. Disconnect and shut down the Secondary, and then upgrade the Secondary.

2. set the old Secondary to be the new Primary (in PRIMARY ALONE state); shut down the old Primary; and upgrade the old Primary.

3. set the old Secondary to be the new Primary (in PRIMARY ALONE state); shut down the old Primary; and upgrade the old Primary.

4. Bring the old Primary back up as the new Secondary. Connect the new Primary and new Secondary servers and let the new Secondary "catch up" to the new Primary.

Note that this procedure will "reverse" your servers. At the end of this sequence of steps, the server that was originally the Primary will be the Secondary, and vice-versa.

8.3 Migration between HSB-Incompatible Versions

When the versions are incompatible, certain scenarios have to be followed in the system upgrade. Specifically there conversion command line parameters that have to be used.

Versions migrateable but HSB- incompatible with this version are: 3.1, 3.7 and 4.0. Migration from earlier versions than 3.1 is not supported.

A more detailed step-by-step procedure is shown below:

8.3.1 Preparation Steps

1. We assume that applications will fail over to the new Primary automatically by sensing the state of each connection. Thus a controlled switchover of the servers will not disrupt the applications, except that open transactions may be aborted during a switchover.

   If your applications have not been designed to fail over automatically, then you may need to notify users that they will lose their connections and will need to re-connect to the new Primary server.

2. Prepare your system and your software for upgrades. Among the tasks that you may want to do:

   a. Since each of your solidDB servers will be operating alone (specifically, in PRIMARY ALONE state) during part of the upgrade operation, you should make sure that both computers are in a "healthy" state, e.g. they have sufficient free disk space, reliable network connections, a UPS in case of power failure, etc.

   b. Each of the servers will operate in PRIMARY ALONE state for at least a short time (while the other server is being upgraded). While the server is in PRIMARY ALONE state, it will be storing transactions in the transaction log. You must have enough disk space available for the log file to store all the transactions that will occur while the other server is being upgraded (including the time it takes that other server to "catch up" after it is restarted).

c. Make sure that a copy of the upgrade software is on each computer or is readily available.

3.

Caution

If you have a "watchdog" program, then you should temporarily turn off that watchdog so that it does not issue commands that conflict with the commands that you issue during the upgrade process.

For example, after you disconnect the Primary from the Secondary, you wouldn"t want the watchdog to try to re-connect them before you upgrade the Secondary.

Cold Migration Procedure

The migration steps are:

1. Disconnect the servers and shut them down.
2. Install the new version of the software.
3. Update the `solid.ini` files following the guidelines below.
4. Start Primary with the command line parameter **-x autoconvert**, which instructs the server to convert existing database to the new format.
5. Set the Primary to the PRIMARY ALONE state.
6. Perform 'hsb copy' or 'hsb netcopy' from Primary to Secondary
7. Connect the servers.

Hot Migration Procedure

Note: in the table below, S1 ("OP") and S2 ("OS") represent the *original* Primary and Secondary servers. Each server's state changes as you go through this process.

Table 8.1. Hot Migration

		ADMIN COMMAND	COMMENTS
1	S1	**ADMIN COMMAND 'hsb set broken';** **ADMIN COMMAND 'hsb status connect ping';**	Disconnect Primary from Secondary.

		ADMIN COMMAND	COMMENTS
2	S 2 OS	**ADMIN COMMAND 'shutdown force';**	Shut down server S2 (Secondary).
3	S 1 OP	**ADMIN COMMAND 'hsb set primary alone';** **ADMIN COMMAND 'hsb state';**	Tell server S1 (Primary) server to operate in PRIMARY ALONE state if it has not already automatically switched to that state. Verify that server S1 (Primary) is in the PRIMARY ALONE state.
4	S 2 OS	Upgrade server S2 (original Secondary).	You should update the configuration parameters in the `solid.ini` file, as well as updating the software.
5	S 2 OS	**solid -x migratehsbg2**	Bring up server S2 (original Secondary) using the **-x migratehsbg2** command-line switch. The server should come up in Secondary Alone state. This command-line switch has two effects. It instructs the server to accept and convert the existing database (the same effect as the **-x autoconvert** parameter). Also, it enables the new Secondary to communicate with the old Primary using the old replication protocol.
6	S 1 OP	**ADMIN COMMAND 'hsb state';** **ADMIN COMMAND 'hsb connect';**	Check server S1 (Primary) server to make sure that it is still in PRIMARY ALONE state. The "hsb connect" command will connect the Primary server to the Secondary, and will start the process by which the Secondary "catches up" on data changes that occurred while the Secondary was down. (Note that you cannot connect from the Secondary if it is running a newer version of the server.)
7	S 1 OP	Wait for the "catchup" to complete before continuing.	If the catchup fails, then you will have to do the following: 1. Shut down server S2 (the Secondary). 2. Do an "hsb copy" from S1 (the Primary) to copy the entire database to server S2. 3. Recover the copy with the old version of the server (S2). 4. Shut Down S2 (the Secondary).

		ADMIN COMMAND	COMMENTS
			5. Go back to the previous step. (Note: In the secon step, you must use "hsb copy" rather than "hsb netcopy" because "hsb netcopy" does not work between different server versions.)
8	S 1 OP	After the servers are connected and caught up, perform: **ADMIN COMMAND 'shutdown force';**	After server S2 (Secondary) has caught up, make it the new Primary. Shut down server S1 (the former Primary) to upgrade it. CAUTION: The "force" option will abort any open transactions.
9	S 2 OS	**ADMIN COMMAND 'hsb set primary alone';** **ADMIN COMMAND 'hsb state';**	Set the new Primary server S2 (old Secondary) to operate in the PRIMARY ALONE state if it has not already automatically switched to that state. Verify that server S2 is in the PRIMARY ALONE state.
10	S 1 OP	Upgrade server S1 (your original Primary server).	You should update the configuration parameters in the `solid.ini` file, as well as updating the software.
11	S 1 OP	**solid -x backupserver**	Re-start the solidDB server on server S1 in the OFFLINE state.
12	S 2 OS	**ADMIN COMMAND 'hsb state';** **ADMIN COMMAND 'hsb netcopy';**	Check the new Primary server S2 (old Secondary) to make sure that it is still in PRIMARY ALONE state. Netcopy the database from the new Primary (S2) to the new Secondary (S1).
13	S 2 OS	**ADMIN COMMAND 'hsb status copy';** **ADMIN COMMAND 'hsb connect';**	Verify that the netcopy succeed. The "hsb connect" command will connect the new Primary server to the new Secondary, and will start the process by which the new Secondary "catches up" on data changes that occurred while it was down. If this step fails, then copy the entire database to the Secondary server (using "hsb copy") and then resume from step 10.

8.3.2 After the Upgrade

After the new Secondary server "catches up" to the new Primary, your system should be completely back to normal. Both the new Primary and the new Secondary server will be upgraded and will have the most current data. You may want to run some test queries to make sure that everything is operating properly.

1. Test that both your Primary server and your Secondary server are working correctly. For example, you might choose the following sequence of operations:

   On the Primary:

   ADMIN COMMAND 'hsb state';

   ADMIN COMMAND 'hsb status catchup';

   Issue some type of read-only query.

   On the Secondary:

   ADMIN COMMAND 'hsb state';

   ADMIN COMMAND 'hsb status catchup';

   Issue some type of read-only query.

2. If you had a "watchdog" program, re-start it.

Note

That this same approach works regardless of whether you want to upgrade your hardware, your operating system, or your solidDB. You can take down one machine and direct all users to connect to the server on the other machine.

Appendix A. Configuration Parameters

This appendix discusses the `solid.ini` configuration parameters that are specific to the Solid CarrierGrade option. These parameters are set in the *`[HotStandby]`* section of the `solid.ini` configuration file. For a discussion of other solid.ini parameters, see *solidDB Administration Guide*.

For information about how to format parameter names and section headings, see *solidDB Administration Guide*, which has an appendix on configuration parameters. That appendix explains the rules you must follow when formatting parameter names and values, etc. The *`[HotStandby]`* section of the `solid.ini` file follows those same rules.

This appendix also explains a few of the parameters that can be used in the *`[Watchdog]`* section of the `solid.ini` file. These parameters are specific to the sample watchdog program that Solid provides. If you write your own watchdog program, you do not need to use any of these parameters.

Note that some parameters in sections other than the *`[HotStandby]`* section also affect HotStandby functionality. These other parameters are documented in Chapter 6, *Configuring HotStandby* and in *solidDB Administration Guide*.

A.1 Ensuring that Primary and Secondary Parameter Values Are Coordinated

This section explains which parameters should be the same on the Primary and Secondary servers, and which parameters should be different.

Certain parameters should be the same on both the Primary and the Secondary. The reason for this is that after a failover, the original Secondary becomes the new Primary, and it should behave the same as the old Primary. Note that using the same values is not an absolute requirement; the servers will not fail if you use different values, but clients may see different behavior.

Some parameters that are not in the *`[HotStandby]`* section, but which are indirectly related, should also be the same on both the Primary and Secondary servers. For example, the *`DurabilityLevel`* parameter generally should be the same on the Primary and Secondary.

Certain parameters should be different on the Primary and Secondary servers. The reason for this is so that the servers can be uniquely identified and can talk to each other.

The following HotStandby parameters should be the SAME on both the Primary and Secondary:

- *[HotStandby]*
  - *2SafeAckPolicy*
  - *AutoPrimaryAlone*
  - *ConnectTimeout*
  - *HSBEnabled*
  - *PrimaryAlone* (deprecated, but should be the same if used)
- *[IndexFile]*
  - *FileSpec* should be "compatible" meaning that the number of *FileSpecs* should be the same and the sizes of the corresponding *FileSpecs* should match.
  - *BlockSize*
- *[Logging]*
  - *BlockSize*

The following parameters should be DIFFERENT:

- *[HotStandby]*
  - *Connect*

The following parameters may be the same or different, depending upon circumstances such as the disk drive configuration on the computer:

- *[General]*
  - *BackupDirectory*
- *[HotStandby]*
  - *CopyDirectory*

There are also some settings of "non-HSB" parameters that affect HSB performance. For example, the *DurabilityLevel* parameter in the *[Logging]* section of the `solid.ini` file has a setting that allows you to optimize performance with HotStandby. See Section 2.3.1, “Adaptive Durability” and see the description of *DurabilityLevel* in *solidDB Administration Guide*.

A.2 Determining Whether the Primary's Settings Take Precedence Over the Secondary's

As we described above, some parameters should be the same for both the Primary and Secondary servers. If you do not set the values the same, you might expect that each server will use the value defined in that server's `solid.ini` file. However, this is not necessarily the case.

Even for some parameters that control the Secondary's behavior, like *`2SafeAckPolicy`*, the value on the Primary is the value that determines the behavior. The principle is that all safeness and durability parameters are controlled at the Primary. For example, the Primary reads its value of *`2SafeAckPolicy`* and sends that value to the Secondary to use. The value stored in the Secondary's solid.ini file is used only if the Secondary becomes the Primary.

Parameters for which the Primary's value takes precedence include:

- *`HotStandby.SafenessLevel`*
- *`HotStandby.2SafeAckPolicy`*
- *`Logging.DurabilityLevel`*
- *`HotStandby.NetcopyRpcTimeout`*

At the time the command '**hsb connect**' is executed, the following parameters residing at the Primary take precedence

- *`HotStandby.PingTimeout`*
- *`HotStandby.PingInterval`*

A.3 Querying HotStandby Configuration Parameters

Standard parameter manipulation commands may be used to query the values and properties of the HotStandby parameters. The commands are:

```
ADMIN COMMAND '[describe] parameter[section_name[.parameter_name]]';
```

For example:

```
ADMIN COMMAND 'parameter logging.durabilitylevel';
```

```
       RC TEXT
       -- ----
        0 Logging DurabilityLevel 3 3 2

ADMIN COMMAND 'parameter hotstandby.MaxLogSize';
       RC TEXT
       -- ----
        0 HotStandby MaxLogSize 10000000 0 0
```

The three values shown in the result row are, from the left:

- *Current value* - set dynamically or inherited from the default or factory value.
- *Default value* - read originally from the `solid.ini` file or inherited from the factory value.
- *Factory value* - preset in the product.

A.4 Modifying HotStandby Configuration Parameters

Normally, you change the value of a parameter by changing the value in the `solid.ini` configuration file and then re-starting the server. However, most of the HotStandby parameters can also be changed with an ADMIN COMMAND:

```
ADMIN COMMAND 'parameter section_name.parameter_name=value
    [temporary]';
```

Note

There is also a deprecated command ADMIN COMMAND "hotstandby parameter ..." that may be used to modify the HotStandby parameters. Its syntax is the following:

```
ADMIN COMMAND 'hotstandby parameter parameter_name value';
```

A.5 Access Mode

When the value of a parameter is changed with an ADMIN command, the change may or may not apply immediately, and may or may not apply the next time that the server is started. If a parameter value is written to the `solid.ini` file, then it will take effect the next time that the server starts. The parameter's Access Mode tells whether the parameter can be changed dynamically, and when the change takes effect.

A.5.1 Access Mode Values

The table later in this appendix lists the Access Mode for each parameter. The possible Access Modes are:

- *RO* (read-only): the value cannot be changed; the current value is always identical to the startup value.
- *RW*: can be changed through an ADMIN COMMAND, and the change takes effect immediately.
- *RW/Startup*: can be changed through an ADMIN COMMAND, and the change takes effect the next time that the server starts.
- *RW/Create*: can be changed through an ADMIN COMMAND, and the change applies when a new database is created.

A.5.2 Saving Parameter Changes

Unless the option temporary is used, all the changes made to the parameters will be saved in the `solid.ini` file at the next checkpoint. The saving may be also expedited with the command:

```
ADMIN COMMAND 'save parameters [file_name]';
```

By default, the command rewrites the default `solid.ini` file. By using the *`file_name`* option, the output may be directed to a different location.

A.6 Cluster Section

Table A.1. Cluster Parameters

`[Cluster]`	**Description**	**Factory Value**	**Access Mode**
`ReadMostlyLoadPercentAtPrimary`	Percentage of read load directed to the Primary	50	RW/Startup

A.7 HotStandby Section

Table A.2. HotStandby Parameters

[HotStandby]	Description	Factory Value	Access Mode
1SafeMaxDelay	In 1-Safe replication, the maximum delay before a committed transaction is sent to the Secondary (in milliseconds).	1000	RW
2SafeAckPolicy	This specifies the timing of the Secondary's acknowledgement when it receives a transaction from the Primary. Valid values are: • 1 = 2-safe received. The Secondary server acknowledges when it receives the data. • 2 = 2-safe visible. The Secondary server acknowledges when the data is "visible", i.e. when the Secondary has executed the transaction. • 3 = 2-safe durable. The Secondary server acknowledges when it has made the data durable, i.e. when it has committed the data and written the data to the disk. Not surprisingly, 2-safe durable is the safest approach, and 2-safe received has the fastest response time. However, in practice, the 2-safe received mode provides in most cases sufficient guarantees for data safety hence providing the best compromise between safety and speed. This parameter applies only if the server is using 2-safe replication. NOTE! Although this parameter controls the Secondary server's behavior, this parameter is set on the Primary. The value in the Secondary's `solid.ini` value is ignored.	1	RW

[HotStandby]	Description	Factory Value	Access Mode
	(See chapters "Determining Whether the Primary's Settings Take Precedence Over the Secondary's" and "Ensuring that Primary and Secondary Parameter Values Are Coordinated" in *solidDB High Availability User Guide*.)		
AutoPrimaryAlone	If this parameter is set to Yes, then the server is automatically put in PRIMARY ALONE state (rather than PRIMARY UNCERTAIN state) when the connection to the Secondary is broken. If you plan to set this to "yes", please read the very important warnings in "Network Partitions and Dual Primaries" in *solidDB High Availability User Guide*.	No	RW
CatchupSpeedRate	While the server is performing catchup, it also continues to service database requests from clients. You may use the *CatchupSpeedRate* parameter to give greater importance to responding to application requests and lower priority to catchup, or vice versa. The speed rate is expressed as a percentage of the maximum available speed dictated by the link and Secondary throughput. Larger numbers mean more emphasis on catchup and less on servicing client requests. Allowed values are 1-99. The factory value is 70.	70	RW
Connect	The *Connect* parameter indicates the "address" of the other HotStandby server in the pair. The format of the Connect string in the HotStandby section is the same as the format of the *Listen* parameter in the *[Com]* section (see *solidDB Administration Guide* for more details). If you omit this parameter in a server that you intend for HotStandby, then you can set this parameter dynamically by using an ADMIN COMMAND. Until the server has a Connect string, the server can only be in the states that do not involve a connection, i.e. PRIMARY ALONE, SECONDARY ALONE, and STANDALONE.	No factory value.	RW

[HotStandby]	Description	Factory Value	Access Mode
	The *Connect* parameter is ignored unless the *HSBEnabled* parameter is set to "yes".		

ConnectTimeout	By specifying a connect timeout value, you can set the maximum time in seconds that a HotStandby connect operation waits for a connection to a remote machine.
	
	The *ConnectTimeout* parameter (which is useful only on certain platforms) is only used with certain administration commands. These are:
	
	hotstandby connect
	
	hotstandby switch primary
	
	hotstandby switch secondary
	
	For example, to set the timeout to 30 seconds (30000 milliseconds)
	
	`[HotStandby]`
	`ConnectTimeout=30000`
	
	See also *PingTimeout*.
		0 (no timeout)
		
		Unit: 1 ms
			RW

CopyDirectory	The *CopyDirectory* parameter in the *[HotStandby]* section defines a name and location for the HotStandby copy operation that is performed when the user executes the command:
	
	ADMIN COMMAND 'hotstandby copy';
	
	For example, the parameter may look like:
	
	`[HotStandby]`
	`CopyDirectory=`
	`C:\Solid\secondary\dbfiles`
		No factory value	RW

[HotStandby]	Description	Factory Value	Access Mode
	If you provide a relative path for the *CopyDirectory* parameter, the path will be relative to the directory that holds the Primary server's `solid.ini` file. This parameter has no factory value, so if the directory is not specified in the `solid.ini` file, it must be provided in the copy command. Please note that **ADMIN COMMAND 'hotstandby netcopy'** as the more flexible solution is the recommended way to copy the database.		
HSBEnabled	If this parameter is set to yes, the server may act as a HotStandby Primary or Secondary server. If this parameter is set to no, then the server may not act as a HotStandby server. Setting this parameter to Yes will implicitly define the default initial state of the server to be SECONDARY ALONE when the server first starts. Valid values are "yes" and "no". To use HotStandby, you must also specify the *Connect* parameter, either by setting it in the `solid.ini` file or by using an ADMIN COMMAND to set it.	no	RO
MaxLogSize	Maximum size of the disk-based HSB log. The factory value: unlimited	0 Unit: 1 byte k=KB m=MB	
MaxMemLogSize	When the file-based logging is disabled (*Logging.LogEnabled=No*), the size of the in-memory log holding transactions before they are sent to the Secondary. The value affects the time the server may stay in the PRIMARY ALONE state, before the in-memory log becomes full.	8M Unit: 1 byte k=KB m=MB	RO
NetcopyRpcTimeout	Data transmission acknowledgment timeout for netcopy operation (in milliseconds)	30000 Unit: 1 ms	RW

[HotStandby]	Description	Factory Value	Access Mode
PingInterval	The Primary and Secondary "ping" each other at regular intervals to make sure that they are still connected. (These pings are independent of the transaction information that the Primary sends to the Secondary.) The value is equal to the interval (in milliseconds) between two consecutive pings sent by a server. The factory value is 1000 (one second).	1000 Unit: 1 ms	RW
PingTimeout	The parameter specifies how long a server should wait before concluding that the other server is down or inaccessible. After the time specified (in milliseconds) has passed the server concludes that a connection is broken and changes the state accordingly. The factory value is 4000 (four seconds). See "PingTimeout and PingInterval Parameters [HotStandby]" in *solidDB High Availability User Guide*. See also *ConnectTimeout*.	4000 Unit: 1 ms	RW
PrimaryAlone	This parameter is deprecated. Use the *AutoPrimaryAlone* parameter.	No	RW
SafenessLevel	To set the safeness level of the replication protocol.	Possible values are: 1safe, 2safe and auto By using the "auto" value, you can allow for the safeness level to be dynamically controlled with Solid's standard	RW

[HotStandby]	Description	Factory Value	Access Mode
		SQL durability control statements. If you set *Safeness-Level* to "auto" and set the durability to relaxed by using the command SET DURABILITY, the safeness level is set to 1-safe, and when you set the durability to strict, the safeness level is set to 2-safe. 2safe	

A.8 Watchdog Section

Important

The parameters in the *[Watchdog]* section of the `solid.ini` file are NOT all pre-defined by Solid. Depending upon how you write your watchdog and whether you want it to read parameter information from the `solid.ini` file, you may use any mix of the parameters defined here and parameters that you have defined. You may also ignore parameters. The parameters shown here are for the sample C-language watchdog program that Solid provides.

Note

Note that in Solid's sample watchdog program, parameter names are not case-sensitive.

Table A.3. Watchdog Parameters

[Watchdog]	Description	Factory Value

AutoSwitch	If the *AutoSwitch* parameter is set to yes, the watchdog automatically does the following:
	
	1) If the Secondary server fails, then the watchdog tells the Primary server to switch to PRIMARY ALONE state (rather than stay in PRIMARY UNCERTAIN) state.
	
	2) If the Primary server fails, then the watchdog automatically sends the commands
	
	`'hsb switch primary'`
	`'hsb set primary alone'`
	
	to switch the original Secondary to be the new Primary. For example:
	
	`[Watchdog]`
	`AutoSwitch = NO`
	
	This parameter is optional.
		Yes

Connect1	The *Connect1* parameter in the *[Watchdog]* section enables the watchdog application to connect to the Primary or Secondary server. This is a required parameter that defines the protocol and network address for the *Connect1* server.	None
Connect2	The *Connect2* parameter in the *[Watchdog]* section enables the watchdog application to connect to the Primary or Secondary server. This is a required parameter that defines the protocol and network address for the *Connect2* server.	None

[Watchdog]	**Description**	**Factory Value**
DualSecAuto-Switch	If *DualSecAutoSwitch = Yes* and both servers are secondary, then the Watchdog will automatically select one of the two secondaries to be a new primary and switch it to primary. If *DualSecAutoSwitch = No* then the system administrator must switch one server to be the primary. Note that *DualSecAutoSwitch* applies whether the Watchdog is in "normal" mode or "failure" mode.	Yes

NumRetry	The *NumRetry* parameter in the *[Watchdog]* section lets you specify the number of watchdog attempts to connect to a Secondary or Primary server before the connection attempt is considered a response failure or error. For example:
	
	`[Watchdog]`
	`NumRetry = 3`
	
	The retries are in addition to the original try. If number of retries is set to 3, then the total number of attempts is 4. Note that the retries are immediate. The watchdog does not wait for an interval of time (such as *PingTimeout*) in between retries when there is a failure.
	
	This parameter is optional.
		0

Password1 *Password2*	See the description of the *Username1* and *Username2* parameters below.	No factory value.
Pessimistic	Setting this parameter to Yes can speed up watchdog reactions. When *Pessimistic = No*, the watchdog checks its connections with the servers, but does not actually act (e.g. change the state of a server to PRIMARY ALONE) until after one of the servers detects that there is a problem and changes its state (e.g. to PRIMARY UNCERTAIN). When *Pessimistic = Yes*, the watchdog acts as soon as the watchdog itself loses contact with one of the	No

[Watchdog]	Description	Factory Value
	servers; the watchdog does not wait for the remaining server to change states. This can speed up the reaction time, but may also increase the odds of false alarms, for example due to network problems. When *Pessimistic = Yes*, the watchdog reacts as follows: If the watchdog has lost contact with the Primary, then the watchdog switches the Secondary to be the Primary; if the watchdog loses contact with the Secondary, then the watchdog sets the Primary to PRIMARY ALONE. **Caution** Setting *Pessimistic = Yes* may cause extra switching or even dual primaries. This parameter should not be set to Yes unless the network is much less likely to fail than the server. You can also turn on Pessimistic behavior by using the optional command-line switch "**-p**".	

PingInterval	The *PingInterval* parameter in the *[Watchdog]* section lets you specify the interval in milliseconds between querying status connect information in normal watchdog mode. To detect server failure, the watchdog sends the hotstandby status connect command to both Primary and Secondary servers after every *PingInterval* milliseconds. For example:
	
	`[Watchdog]`
	`PingInterval = 5000`
	
	This parameter is optional.
	
	Caution
	
	Previous sample watchdogs required that the *PingInterval* be specified in seconds, not
		1000
		
		(1 second)

[Watchdog]	**Description**	**Factory Value**
	milliseconds. If you are using an older `solid.ini` file, you should update it.	

Username1
 	
Username2
	The *Username* and *Password* parameters in the *[Watchdog]* section are optional. They specify the username and password that are authorized for using the connect1 server. For example:
	
	`[Watchdog]`
	`Username1 = Tom`
	`Password1 = dr17xy`
	`Username2 = Jerry`
	`Password2 = M89tvt`
	
	If (for security reasons) these parameters are not specified in the `solid.ini` configuration file, the watchdog will prompt for them when the watchdog is started.
		No factory value.

WatchdogLog	The *WatchdogLog* parameter in the *[Watchdog]* section lets you specify the file name of the watchdog log. The watchdog log is created in the current working directory. It is used to record watchdog messages, alerting administrators of the need to issue watchdog commands.
	
	For example:
	
	`[Watchdog]`
	`WatchdogLog = Watchdog.log`
	
	Note that quotation marks around the file name are not required (as long as it does not contain special characters such as the blank or certain punctuation marks).
	
	This parameter is optional.
		Watchdog.log

Note

HotStandby is affected by other configuration parameters such as *DurabilityLevel*.

When using the parameter

```
[Logging]
DurabilityLevel
```

the *DurabilityLevel* parameter value affects only the Primary server. The logging mode of the Secondary server is dictated by the *2SafeAckPolicy* parameter in the *[HotStandby]* section.

Appendix B. Error Codes

This appendix documents error codes that are related to HotStandby. Most other server error codes are documented in *solidDB Administration Guide*.

Some of the errors documented in this chapters are values of the RC column of the ADMIN COMMAND result set, whereas some other errors are returned as the error code of the ODBC or JDBC driver. For example, all "Solid HotStandby Errors" are ADMIN COMMAND result set values, whereas all "Solid Communication Errors" are returned by the driver.

The error categories covered in the tables contained in this appendix are:

- Solid HotStandby Errors (14009 - 147xx, 307xx)

  These errors occur when using specific HotStandby commands, which are Solid SQL extensions.

- Solid Database Errors (10002 - 10050)

  These errors are detected by Solid and are sent to the client application. They may demand administrative actions.

- Solid Database Table Errors (13xxx)

  These errors are caused by erroneous SQL statements and are detected by Solid. Administrative actions are not needed.

- Solid Communication Errors (21306, 21308)

  These errors are caused by network errors. These errors demand administrative actions.

B.1 HotStandby Errors and Status Codes

Table B.1. HotStandby Errors and Status Codes

Error/Status code	Description
14003	ACTIVE ADMIN COMMANDs that may return this status in the result set of the command: **ADMIN COMMAND 'hotstandby status switch'**

Error/Status code	Description
	ADMIN COMMAND 'hotstandby status catchup' **ADMIN COMMAND 'hotstandby status copy'** Meaning: The switch process, catchup process, or copy/netcopy process is still active.
14007	CONNECTING ADMIN COMMANDs that may return this status in the result set of the command: **ADMIN COMMAND 'hotstandby status connect'** Meaning: The Primary and Secondary servers are in the process of connecting.
14008	CATCHUP ADMIN COMMANDs that may return this status in the result set of the command: **ADMIN COMMAND 'hotstandby status connect'** Meaning: The Primary server is connected to the Secondary server, but the transaction log is not yet fully copied. This message is returned only from the Primary server.
14009	No server switch occurred before. ADMIN COMMANDs that may return this status in the result set of the command: **ADMIN COMMAND 'hotstandby status switch'** Meaning: The switch process has never happened between the servers.
14501	Operation failed. Meaning: The operation failed and the server is shutting down. Failure may be due to issuing the command to a non-HotStandby server, or to either a Primary or Secondary server in which the command does not apply. ADMIN COMMANDs that may return this status in the result set of the command: **ADMIN COMMAND 'hotstandby switch primary'** **ADMIN COMMAND 'hotstandby switch secondary'** **ADMIN COMMAND 'hotstandby cominfo'**

Error/Status code	Description
	ADMIN COMMAND 'hotstandby connect' **ADMIN COMMAND 'hotstandby status switch'** **ADMIN COMMAND 'hotstandby set standalone'** **ADMIN COMMAND 'hotstandby copy'** **ADMIN COMMAND 'hotstandby netcopy'**
14502	RPC parameter is invalid Meaning: some of the connection info provided in the HSB connect string is erroneous and the connection to another server failed. ADMIN COMMANDs that may return this status in the result set of the command: **ADMIN COMMAND 'hotstandby status connect'**
14503	Communication error, connection lost. Meaning: There was a communication error and the other server was not found. There was a failure to connect to the other server. ADMIN COMMANDs that may return this status in the result set of the command: **ADMIN COMMAND 'hotstandby status connect'**
14520	Server is HotStandby secondary server, no connections are allowed.
14522	HotStandby copy directory not specified. Meaning: No copy directory is specified. ADMIN COMMANDs that may return this status in the result set of the command: **ADMIN COMMAND 'hotstandby copy'** To solve this problem, either specify the directory as part of the command, e.g. **ADMIN COMMAND 'hotstandby copy \Secondary\dbfiles\'** or else set the *`CopyDirectory`* parameter in the `solid.ini` configuration file.
14523	Switch process is already active.

Error/Status code	Description
	Meaning: The switch process is already active in the HotStandby server. If you only need to complete the current switch, then wait. If you are trying to switch a second time (that is, switch back to the original configuration), then you must wait for the first switch to complete before you can start the second switch. ADMIN COMMANDs that may return this status in the result set of the command: **ADMIN COMMAND 'hotstandby switch primary'** **ADMIN COMMAND 'hotstandby switch secondary'** **ADMIN COMMAND 'hotstandby status switch'**
14524	HotStandby databases have a different base database, database time stamps are different. Meaning: Databases are from a different seed database. You must synchronize databases. You may need to perform netcopy of the Primary's database to the Secondary. ADMIN COMMANDs that may return this status in the result set of the command: **ADMIN COMMAND 'hotstandby connect'** **ADMIN COMMAND 'hotstandby status switch'**
14525	HotStandby databases are not properly synchronized. Meaning: Databases are not properly synchronized. You must synchronize the databases. You may need to start one of the database servers (the one that you intend to become the Secondary) with the command line parameter **-x backupserver** and then netcopy the Primary's database to the Secondary. ADMIN COMMANDs that may return this status in the result set of the command: **ADMIN COMMAND 'hotstandby connect'** **ADMIN COMMAND 'hotstandby status switch'**
14526	Invalid argument. Meaning: An argument used in the HotStandby ADMIN COMMAND is unknown or invalid.

Error/Status code	Description
	All HotStandby commands can return this error in the result set of the ADMIN COMMAND. Note: In the following HotStandby commands, the invalid argument error is a syntax error when the specified Primary or Secondary server can not apply to the switch: **ADMIN COMMAND 'hotstandby switch primary'** **ADMIN COMMAND 'hotstandby switch secondary'**
14527	This is a non-HotStandby server. Meaning: The command was executed on a server that is not configured for HotStandby. ADMIN COMMANDs that may return this status in the result set of the command: **ADMIN COMMAND 'hotstandby connect'** **ADMIN COMMAND 'hotstandby status switch'** **ADMIN COMMAND 'hotstandby switch primary'** **ADMIN COMMAND 'hotstandby switch secondary'** **ADMIN COMMAND 'hotstandby state'**
14528	Both HotStandby databases are primary databases. Meaning: Both databases are Primary. This is a fatal error because there may be conflicting changes. Both databases are automatically dropped to Secondary state by the system. You must decide which database is the real Primary database and then synchronize the databases. ADMIN COMMANDs that may return this status in the result set of the command: **ADMIN COMMAND 'hotstandby connect'** **ADMIN COMMAND 'hotstandby status switch'**
14535	Server is already a primary server. Meaning: The server you are trying to switch to Primary is already in one of the PRIMARY states.

Error/Status code	Description
	ADMIN COMMANDs that may return this status in the result set of the command: **ADMIN COMMAND 'hotstandby switch primary'**
14536	Server is already a secondary server. Meaning: The server you are trying to switch to Secondary is already in one of the SECONDARY states. ADMIN COMMANDs that may return this status in the result set of the command: **ADMIN COMMAND 'hotstandby switch secondary'**
14537	HotStandby connection is broken. Meaning: This command is returned from both the Primary and Secondary server. ADMIN COMMANDs that may return this status in the result set of the command: **ADMIN COMMAND 'hotstandby status connect'** One possible cause of this problem is an incorrect Connect string in the Secondary's `solid.ini` file. If the netcopy operation succeeds but the connect command fails, check the Connect string. (Netcopy does not require the Secondary to open a separate connection to the Primary, and thus may succeed even if the Connect string on the Secondary is wrong.)
14538	Server is not HotStandby primary server. Meaning: To issue this command, the server must be a HotStandby Primary server. ADMIN COMMANDs that may return this status in the result set of the command: **ADMIN COMMAND 'hotstandby copy copy_directory'** **ADMIN COMMAND 'hotstandby netcopy'** **ADMIN COMMAND 'hotstandby connect'** **ADMIN COMMAND 'hotstandby set primary alone'** **ADMIN COMMAND 'hotstandby set standalone'**
14539	Operation Refused.

Error/Status code	Description
	This error code is given when one of the following situations occurs: • The user issued a **netcopy** command to a Primary server, but the server that should be Secondary is not actually in a Secondary state, or is not in "netcopy listening mode". (Both the Primary and the "Secondary" server are probably in PRIMARY ALONE state.) To solve the problem, re-start the "Secondary" with the **-x backupserver** command-line option, then try again to issue the **netcopy** command to the Primary. **Warning** If both servers were in PRIMARY ALONE state, and if both servers executed transactions while those servers were in PRIMARY ALONE state, then they probably each have data that the other one does not. This is a serious error, and doing a **netcopy** to put them back in sync would result in writing over some transactions that have already been committed in the "Secondary" server. • This message can be generated when you use a callback function(s) and the callback function refuses to shut down or accept a backup/netcopy command. When you use AcceleratorLib, you can provide "callback" functions by using the `SSCSetNotifier` function. Your callback functions will be notified when the server has been commanded to shut down or to do a netcopy operation. If for some reason your application doesn't want the command to be followed, then your callback can return a value that cancels the command. In this situation, you will see error 14539. To solve the problem, wait until the client code finishes the operation that it does not want to interrupt, then re-try the command (e.g. the shutdown or netcopy).
14540	Server is already a non-HotStandby server.
14541	HotStandby configuration in `solid.ini` conflicts with **ADMIN COMMAND 'HSB SET STANDALONE'**.
14542	Server in backupserver mode. Operation refused.
14543	Invalid command. The database is a hotstandby database but, hotstandby section not found in `solid.ini` configuration file.

Error/Status code	Description
14544	Operation failed. This command is not supported on diskless server.
14545	Primary can only be set to primary alone when its role is primary broken.
14546	Switch failed. The server or the remote server cannot switch from primary alone to secondary server. Catchup should be done first before switch. Meaning: This command is returned when a state switch to SECONDARY is executed from a local or remote Primary server that is in the PRIMARY ALONE state and it is detected that the Primary and Secondary server are not in sync. You must connect the Primary server to the Secondary server and wait for the catchup process to complete before switching the Secondary to the Primary. HotStandby commands that return this error: **ADMIN COMMAND 'hotstandby switch secondary'**
14547	The value for the -R option (Read Timeout) was missing or invalid.
14548	Switch failed. The server in Standalone cannot be switched to a secondary. Meaning: This command is returned when a state switch to SECONDARY is executed from a local or remote Primary server that is in the STANDALONE state and it is detected that the Primary and Secondary server are not in sync. You must connect the Primary server to the Secondary server and wait for the catchup to complete before switching the Secondary to the Primary. HotStandby commands that return this error: **ADMIN COMMAND 'hotstandby switch secondary'**
14549	HotStandby transaction is active. Meaning: If the HotStandby connection is broken, Primary server must be set to alone mode or switched to secondary mode before shutdown.
14550	Hotstandby connect parameter can be changed only when the primary is not connected to secondary.
14551	Maximum number of START AFTER COMMIT statements reached.
14552	Server is in backup server mode, no connections are allowed.
14700	Rejected connection, both servers in PRIMARY role. Meaning: Command **'hsb connect'** returns this error if both nodes are in same role.

Error/Status code	Description
14701	14701:Rejected connection, both servers in SECONDARY role. Meaning: Command **'hsb connect'** returns this error if both nodes are in same role.
14702	14702:Operation failed, catchup is active. Meaning: While the servers are performing catchup, you will get this error if you issue any of the following commands on the Primary: **'hsb switch secondary'**, **'hsb set secondary alone'**, **'hsb set standalone'**, **'hsb connect'**, **'hsb copy'** or **'hsb netcopy'**. While the servers are performing catchup, you will get this error if you issue any of the following commands on the Secondary: **'hsb switch primary'**, **'hsb set secondary alone'**, **'hsb set primary alone'**, **'hsb set standalone'**, or **'hsb connect'**.
14703	Operation failed, copy is active. Meaning: While the Primary is doing copy or netcopy, the following commands returns this error: **'hsb switch secondary'**, **'hsb set secondary alone'**, **'hsb set standalone'**, **'hsb connect'**, **'hsb disconnect'**, **'hsb copy'** or **'hsb netcopy'**.
14704	HotStandby copy or netcopy is only allowed when primary is in alone state. Meaning: This error is returned if the server is in PRIMARY ACTIVE state and the command **'hsb copy'** or **'hsb netcopy'** is issued.
14705	Setting to STANDALONE is not allowed in this state. Meaning: If the server is in PRIMARY ACTIVE state and you issue the command **'hsb set standalone'**, then you will get this message.
14706	Invalid read thread mode for HotStandby, only mode 2 is supported.
14707	Operation not allowed in the STANDALONE state.
14708	Catchup failed, catchup position was not found from log files.
14709	Hot Standby enabled, but connection string is not defined.

Error/Status code	Description
14710	Hot Standby admin command conflict with an incoming admin command.
14711	Failed because server is shutting down.
14712	Server is secondary. Use primary server for this operation.
30787	pri_dologskip:bad type, log pos, log size This error refers to a failed operation on the HSB primary server. The error returns the failed operation and its location in the log, and the log size. Operations in the replication log are skipped.
30788	pri_hsblogcopy_write:bad type, log pos, log size This error refers to a failed operation on the HSB primary server. The write to the replication log file fails. The error returns the failed operation and its location in the log, and the log size.
30789	Failed to open hot standby replication log file
30790	Failed to allocate memory for HotStandby log. Max Log size is <logsize> This error concerns a diskless database using hotstandby. In these systems, the hotstandby log is written to memory. This error is given if allocating more memory for the log file fails.

B.2 Solid Database Errors

Table B.2. Solid Database Errors

Error code	Description
Database Error 10002	Operation failed. Meaning: The connect operation failed with an unexpected error. Most likely, the servers are not properly synchronized.
Database Error 10013	Transaction is read-only. Meaning: You have tried to write inside a transaction that is set read-only, or the server is temporarily set to read-only mode, for example during the state switch. Updateable transactions are not allowed.
Database Error 10019	Backup is already active.

Error code	Description
	Meaning: You have tried to start a backup or copy when one is already in progress.
Database Error 10024	Illegal backup directory "*directory_name*". Meaning: The backup or copy directory is either an empty string or a dot indicating that the backup or copy will be created in the current directory.
Database Error 10030	Backup or copy directory '*directory_name*' does not exist. Meaning: Backup or copy directory is not found. Check the name of the backup or copy directory.
Database Error 10045	This operation cannot be executed on a HotStandby Secondary server. Meaning: This operation cannot be executed on a HotStandby Secondary server. In order for the requested operation to succeed, the server must be a Primary.
Database Error 10046	Operation failed, data dictionary operation is active. Meaning: A data dictionary operation is currently in progress.
Database Error 10047	Replicated transaction is aborted. Meaning: Transactions are aborted, for example, in a state switch. When the server state is switched from Primary to Secondary, all active transactions are aborted.
Database Error 10048	Replicated transaction contains data dictionary changes, normal update operations are not allowed. Meaning: HotStandby mode restricts data dictionary operations; for example, CREATE TABLE cannot be mixed with normal update operations. This message is obsolete in version 4.1 and later, which allow you to mix DML and DDL operations within a transaction while using HSB.
Database Error 10049	The remote server is not a secondary server. Meaning: The server that you specified in the command is not in a SECONDARY state.
Database Error 10050	Replicated operation updated BLOB columns. Meaning: BLOB columns cannot be replicated to the Secondary server.
Database error 10078	User rolled back the transaction.
Database error 10079	Cannot remove filespec. File is already in use.

Error code	Description
Database error 10080	HotStandby Secondary server can not execute operation received from Primary server. Meaning: A possible cause for this error is that the database did not originate from the Primary server using HotStandby copy or netcopy command.
Database error 10081	The database file is incomplete or corrupt. Meaning: If the file is on a hot standby secondary server, use the 'hotstandby copy' or 'hotstandby netcopy' command to send the file from the primary server again.
Database error 10082	Backup aborted.
Database error 10083	Failed to abort hsb trx because commit is already sent to secondary.
Database error 10084	Table is not locked.
Database error 10085	Checkpointing is disabled.
Database error 10087	HotStandby not allowed for main memory tables.
Database error 10088	Specified lock timeout is too large.
Database error 10089	Operation failed, server is in HSB primary uncertain mode.

B.3 Solid Table Errors

Table B.3. Solid Table Errors

Error code	Description
Table Error 13068	Server shutdown in progress Meaning: You are unable to complete this operation because server shutdown is in progress.
Table Error 13123	Table '*table_name*' is not empty Meaning: This operation can only be executed when a table is empty. For example, you can only change a table from disk-based to in-memory (or vice-versa) when the table is empty.
Table Error 13167	Only M-tables can be transient Meaning: You cannot create a transient table that is disk-based. For example, the following SQL statement will get this error message:

Error code	Description
	`CREATE TRANSIENT TABLE t1 (i INT) STORE DISK;`
Table Error 13170	Only M-tables can be temporary Meaning: You cannot create a temporary table that is disk-based. For example, the following SQL statement will get this error message: `CREATE TEMPORARY TABLE t1 (i INT) STORE DISK;`

B.4 Solid Communication Errors

Table B.4. Solid Communication Errors

Error code	Description
Communication Error 21306	Server "*`server_name`*" not found, connection failed. Meaning: The Secondary server was not found. • Check that the server is running. • Check that the network name is valid. • Check that the server is listening to the specified network name.
Communication Error 21308	"Connection is broken (%s '%s' operation failed with code %d)". For example, `"Connection is broken (TCP/IP 'Write' operation failed with code 7)."` See the recommended actions for error 21306.

Appendix C. Summary of HotStandby Administrative Commands

The chapter summarizes the administrative commands available with HotStandby.

The tool you are using (for example Solid SQL Editor (solsql) or Solid Remote Control (solcon)) affects how you must enter the command.

Using Solid SQL Editor (teletype), you enter the command using the syntax shown below:

```
ADMIN COMMAND 'hotstandby switch primary';
```

If you are entering these commands in Solid Remote Control (teletype), be sure to specify the command only (without quotes and without "ADMIN COMMAND"); for example:

```
hotstandby switch primary
```

Note also that you may abbreviate 'hotstandby' to 'hsb', for example

```
hsb switch primary
```

In the table below, we use the shortest possible form; we omit the "ADMIN COMMAND" and the quotes, and we also use the abbreviation 'hsb'.

For more information about Solid Remote Control (teletype) or Solid SQL Editor, refer to "Using SOLID Data Management Tools" in *solidDB Administration Guide*.

Note that an ADMIN COMMAND always return a success return code (0) if there is no syntax error in the command. The actual result code of the command is included in the "RC" field of the result set.

Table C.1. HotStandby Commands

Command	Explanation
hsb cominfo	Returns the communication information (connect string) used to connect to the other server. This is usually the value of the *Connect* parameter in the *HotStandby* section of the solid.ini configuration file, but might also have been set with the command:

Command	Explanation
	ADMIN COMMAND 'hsb parameter connect <connect_string>'; You can use this information in an application to connect to other servers.
hsb connect	If the connection between the Primary and Secondary servers is broken or has not yet been established, this command connects the Primary server to the Secondary server and starts HotStandby replication. This command is always needed to connect the servers since there is no automatic mechanism for connecting between servers. After a successful connect, the state of the Primary server is automatically set from PRIMARY ALONE to PRIMARY ACTIVE. If unsuccessful, the state remains PRIMARY ALONE. This command can be executed on either the Primary or the Secondary. Note: When you execute this command, if the Primary server and Secondary server are connected, but the transaction log is not yet fully copied to the Secondary, the following message is displayed: Catchup is active
hsb copy [*`directory_name`*]	Note: This command is deprecated. It is recommended that you use the hsb netcopy command instead. You can use the hsb copy command to initially create the Secondary database from the Primary. This command copies the database into a directory that is local to the Primary node (and also local to the Secondary node, of course). After the copy is completed, you may start the Secondary server. After you connect the Primary to the Secondary, the Primary automatically brings the Secondary server up-to-date by copying the transaction log to the Secondary server. You can also use this command to synchronize the Primary database with a Secondary database (when it has been off line for a considerable period of time) that is in a directory local to the Primary node. Read Section 4.8, “Synchronizing Primary and Secondary Servers”. If the optional *`directory_name`* is specified, the database files are copied to that directory; otherwise it is copied to the directory specified with the *`copydirectory`* parameter in the *`[Hotstandby]`* section of the `solid.ini` configuration file. Because the hsb copy command does not copy the `solid.ini` configuration file or log files, it is recommended that you make this directory different from the normal backup directory. The Primary can execute the hsb copy command only if the Primary is in PRIMARY ALONE state. During and after the command, the server remains in PRIMARY

Command	Explanation
	ALONE state. After the command has been completed, you may start the Secondary server and then connect the two servers.
hsb disconnect	This tells the server to gracefully disconnect from the other member of the HSB pair. This command is valid on either the Primary or the Secondary server. A typical reason to use this command is to disconnect the servers before upgrading one of them. (The other server can be set to PRIMARY ALONE state so that it can continue responding to client requests.) This command normally causes both servers to go into an "Alone" mode; i.e. the Primary server switches from PRIMARY ACTIVE to PRIMARY ALONE, while the Secondary server switches from SECONDARY ACTIVE to SECONDARY ALONE. This command is valid on both the Primary and the Secondary. Note that using the shutdown command **ADMIN COMMAND 'shutdown';** causes the server to do a controlled disconnect before it shuts down. If the Secondary is shut down (and disconnects), then the Primary knows that it is safe to go to PRIMARY ALONE state, and will do so.

hsb logpos	These two commands can be used by a Watchdog program (or a person) to determine which of two servers should be switched to Primary and which should be switched to Secondary. (The server that was the Primary before the servers lost contact with each other is not necessarily the server that should become the Primary now.) This approach detects which of the servers is "ahead" (that is, which has accepted more transactions) and thus should be made the Primary before establishing the HSB connection.
	
	For a detailed explanation of how to use this command, see Section 4.12, “Choosing Which Server to Make Primary”.
	
	A typical output is shown below:
	
	`ADMIN COMMAND 'hsb logpos';`
	`RC TEXT`
	`-- ----`

Command	Explanation
	`0 00000000000000000871:PRIMARY` The output consists of the log operation ID and the server's previous role, which in this example was Primary.
hsb netcopy	This command is used to copy the Primary database or diskless in-memory data to a Secondary server using the *`Connect`* parameter specified in the *`[HotStandby]`* section of `solid.ini`. Once the connect string is used to connect to the Secondary server, the database files are copied through the network link. You can use this command to synchronize a Primary database with a Secondary database that has been off line for a long time. Read Section 4.8, “Synchronizing Primary and Secondary Servers”. You can also use this command to create a new Secondary database. Reasons for this may be a corrupt Secondary database, creation of the initial Secondary database for a new HotStandby configuration, or the addition of a new Secondary to an existing configuration. Read Section 4.8.5, “Copying a Primary Database to a Secondary Over the Network” for details. The Primary server must be in PRIMARY ALONE state to issue this command. After the command has completed (successfully or unsuccessfully), the Primary server remains in the same state. If the copy is completed successfully, then the Secondary server is automatically switched to SECONDARY ALONE state. The netcopy command is usually followed by the "connect" command to connect the Primary and Secondary servers. After the Primary server is connected to the Secondary, the Primary automatically brings the Secondary up-to-date by copying the transaction log.
hsb parameter	(Deprecated) This command allows you to set HSB-specific parameters such as *`AutoPrimaryAlone`*, *`Connect`*, and *`PingTimeout`*. For a complete description of each of these parameters, see Appendix A, *Configuration Parameters*. Note that when you set the value of one of some parameters, the command takes effect immediately, but is not written to the `solid.ini` configuration file before a shutdown is executed. The syntax for this command is:

Command	Explanation

	`ADMIN COMMAND 'hsb parameter` *`param_name param_value`*`';`
	
	Note that this command does not use an equals sign. Thus it differs from the otherwise similar command (recommended:
	
	`ADMIN COMMAND 'parameter` `hotstandby.`*`param_name = param_value`*`';`

hsb role	Note: This command is deprecated. Please use **hsb state** instead. Returns one of the following roles in the result set: • PRIMARY, if the connected server is a normal Primary server. In this role, the transactions at the Primary server are sent to the Secondary server. • PRIMARY NOHSBLOG, indicating that the Primary server accepts transactions and stores them in the database, however, it does not store the transactions so that it can later send them to the Secondary. To re-synchronize the Secondary with the Primary, the entire database at the Primary must be copied to the Secondary server. • PRIMARY BROKEN, if the Primary server has a broken connection to the Secondary server. Only read-only transactions can be executed in the Primary server. • PRIMARY ALONE, if the Primary server is working by itself. The connection to the Secondary is broken, but transactions are accepted and added to the transaction log at the Primary so that later they can be sent to the Secondary. • PRIMARY CATCHUP, if the catchup is in progress. During catchup, the Primary automatically sends the transaction log changes to the Secondary server after the 'hsb connect' command has been issued at the Primary. After the catchup process is completed, the role of the server is switched automatically to PRIMARY. The Primary can continue to accept transactions if its role was PRIMARY ALONE before the connect.

Command	Explanation
	• SECONDARY, if the connected server is a normal Secondary server. This means the server receives and applies transactions from the Primary. • SECONDARY BROKEN, if the Secondary server has a broken connection to the Primary server. • SECONDARY CATCHUP, if the Secondary server is catching up with the changes from the Primary server after the 'hsb connect' command was issued at the Primary server. After the catchup process is completed, the role of the Secondary is switched automatically to SECONDARY. If ADMIN COMMAND 'hsb role' is issued on a server that is not configured for HotStandby, the following error message is returned: 14527: This is a non-HotStandby Server. This command returns the same information as the SQL function: HOT-STANDBY_ROLE.
hsb set primary alone	Sets a HotStandby Primary server unconditionally to the PRIMARY ALONE state. The command is legal in the following states: PRIMARY ACTIVE, SECONDARY ACTIVE, SECONDARY ALONE and STANDALONE. This command can be used to implement fast failovers. When Secondary is in the SECONDARY ACTIVE state, the server will not make any attempt to communicate with the Primary, having received this command. Instead, it will immediately switch to the PRIMARY ALONE state. This behavior may be utilized in cases when the information about the Primary failure reaches the Watchdog (or other HA manager) before the Secondary has detected the failure (the delay is dictated by the *`PingTimeout`* and *`PingInterval`* parameters). However, if it happens (e.g. because of incorrect failure detection) that the Primary is "alive" and in the PRIMARY ACTIVE state when this command is executed in the Secondary, the Primary will be automatically forced to PRIMARY UNCERTAIN state. It can be then moved to the SECONDARY ALONE state and reconnected without any loss of transactions. NOTE: the alternative way of executing failovers is to use the "hsb switch primary" command. In the PRIMARY ALONE state, the connection to the Secondary server is broken, but this state allows the Primary server to run with continuous updates to the transaction log. The PRIMARY ALONE state persists until the Primary server is shut down,

Command	Explanation
	a connection is successfully made to the Secondary server, or the server runs out of space for the transaction log. Note that when you set a server to PRIMARY ALONE state, it does not automatically make any attempt to re-establish connections with the other server. Important: Before executing this command on a server, try to make sure that the other server in the pair is not already in PRIMARY ALONE state (or STANDALONE state). It is very important to avoid "dual primaries" (see Section 4.3.1, “Network Partitions and Dual Primaries”). See also the command '**hsb switch primary**'.
hsb set secondary alone	This command sets the server state to SECONDARY ALONE. This command is available if the server is currently in one of the following states: PRIMARY ALONE, PRIMARY UNCERTAIN, STANDALONE.
hsb set standalone	When this command is issued, the state of the Primary server becomes STANDALONE. The server stops storing transactions for the Secondary server. The Primary (STANDALONE) can continue accepting read/write transactions. This option is useful in the Primary server when the Secondary server is offline for a significant period of time and the transaction log may grow too large. This command is available if the server is currently in one of the following states: PRIMARY ALONE or SECONDARY ALONE.
hsb state	Returns one of the following states in the result set: • PRIMARY ACTIVE, if the connected server is a normal Primary server. In this state, transactions on the Primary server are sent to the Secondary server. • STANDALONE, indicating that the Primary server accepts transactions and stores them in the database, but it does not store the transactions to forward them to the Secondary. • PRIMARY UNCERTAIN, if the Primary server has a broken connection to the Secondary server and has not yet been switched to another state, such as PRIMARY ALONE. Only read-only transactions can be executed in the Primary server. • PRIMARY ALONE, if the Primary server is working by itself. The connection to the Secondary is broken, but transactions are accepted and stored in the Primary's transaction log so that they can be forwarded to the Secondary.

Command	Explanation
	• SECONDARY ACTIVE, if the connected server is a normal Secondary server. This means the server receives and applies transactions from the Primary. • SECONDARY ALONE, if the Secondary server has a broken connection to the Primary server. If **ADMIN COMMAND 'hsb state'** is issued on a server that is not configured for HotStandby, the following error message is returned: 14527: This is a non-HotStandby Server. This command returns the same information as the SQL function: `HOT-STANDBY_STATE`. Read section *Using Function HOTSTANDBY_STATE* in the section called "Detecting HotStandby Server Failure in Client Applications" for details on this function. Refer to Appendix D, *Server State Transitions*, for an overview of HotStandby state transitions that occur while performing administrative and troubleshooting operations.
hsb status *option*	Returns HotStandby status information. The option may be any of the following: • catchup • connect • copy • switch For more details, see the descriptions of the individual commands/options below, e.g. **'hsb status catchup'**. The intention of the status command is give the information about the outcome of operations that take a prolonged time, after they have started successfully. The command will return status of the last successfully started operation. If the starting of operation fails (e.g. because of incorrect state) the status command will not return the status of that operation but the one executed previously.
hsb status catchup	Indicates whether or not the server is doing catchup, i.e. when the Secondary reads the Primary's transaction log and applies the changes. Possible values are: • ACTIVE

Command	Explanation
	• NOT ACTIVE
hsb status connect	Status information returned: • CONNECTED - Connect active. This information is returned from both the Primary and Secondary servers. • CONNECTING - The Primary server and Secondary server are connecting to each other. This information is returned from both the Primary and Secondary servers. • CATCHUP - The Primary server is connected to the Secondary server, but the Primary HotStandby database log is not fully copied to the Secondary server. This information is returned from both the Primary and Secondary servers. • BROKEN - Connection between the Primary and Secondary server is broken. This information is returned from both the Primary and Secondary servers. NOTE: This command returns the same information as the SQL function `HOTSTANDBY_CONNECTSTATUS`. Read section *Using Function HOTSTANDBY_CONNECTSTATUS* in the section called “Switching the Application to the New Primary” for details on this function.
hsb status copy	This command allows you to check the result of the last **hsb copy** or **hsb netcopy** command. Note that this status command always uses the keyword "copy", even if you are checking the result of a **netcopy** rather than a **copy**. Status information returned: • SUCCESS - Copy completed successfully. • ACTIVE - Copy process is still active. • ERROR *`number`* - Copy failed with error code *`number`*.
hsb status switch	Returns HotStandby switch status information. Status information returned: • ACTIVE - Copy process is still active. • SUCCESS - Copy completed successfully. • ERROR *`number`* - Copy failed with error code *`number`*.

Command	Explanation
	• NO SERVER SWITCH OCCURRED BEFORE - No switch has happened before.
hsb switch primary	Switches the database server to PRIMARY. The command starts a switch process, which can be monitored using command hsb status switch. If the servers are connected at the time that you execute this command, then the servers simply reverse states — i.e. the old Primary changes from PRIMARY ACTIVE to SECONDARY ACTIVE, and the Secondary server switches from SECONDARY ACTIVE to PRIMARY ACTIVE. If the servers are not connected and the server is in SECONDARY ALONE state, then when you switch the server to Primary it will end up in PRIMARY ALONE state. The new Primary server will not automatically try to connect to the other server and switch to PRIMARY ACTIVE state. Because the command is available both in the SECONDAR ACTIVE and SECONDARY ALONE states, it can be used to perform failovers. However, because the server will always make attempt to communicate with the Primary, the network timeout may be involved. Thus, this method is slower than using the **'hsb set primary alone'** command. On the other hand, this method secures better against a possibility of "dual primaries". See also the command **'hsb set primary alone'**.
hsb switch secondary	Switches the database to Secondary state. All active write transactions are terminated. **Note** If the connected database server is a Primary server, it becomes a Secondary server. If the old Secondary server is available, then the old Secondary server is switched to Primary (see the **hsb switch primary** command). **Note** f the **switch** command is issued inside an open transaction (Microsoft Windows after the transaction has started and before you execute the COMMIT statement), then when you issue the COMMIT statement, the COMMIT fails with an error: 'replicated transaction is aborted'. All transactions are rolled back during the switch, including the transaction in which the switch statement is executed. The switch itself is successful (i.e. is not rolled back) because ADMIN COMMANDs are not transactional commands. (NOTE: Adminis-

Command	Explanation
	trative commands do force the start of a new transaction if one is not already open, however.)

Appendix D. Server State Transitions

This chapter describes the possible state transitions (e.g. the transition from OFFLINE to SECONDARY ALONE). A description of each of the server states is in Section 2.2, “Description of Server States”.

D.1 HotStandby State Transition Diagram

The diagram in this appendix shows the state transitions that can occur, and the circumstances under which they may occur. For example, you can change the state of a server from PRIMARY UNCERTAIN to PRIMARY ALONE by executing the command **'hsb set primary alone'**:

ADMIN COMMAND 'hsb Set Primary Alone';

As you use this diagram, please remember the following:

1. We do not show the complete syntax of the commands. For example, we show:

   ```
   'hsb set primary alone'
   ```

   rather than

   ```
   ADMIN COMMAND 'hsb Set Primary Alone';
   ```

2. The state transition path(s) shown for **'hsb copy'** also apply to **'hsb netcopy'**.

3. Some commands may fail when they are executed. When a command might succeed or fail, we show both possibilities. If the branch is intended to describe what happens if the command fails, it will have the word 'failed':

   'Disconnect' failed.

4. In some situations, the behavior depends upon the setting of the `solid.ini` configuration parameter named *`AutoPrimaryAlone`*. We often use the abbreviation "*`APA`*" to represent this parameter.

5. When the diagram refers to "events", it refers to internally-generated notifications. These are not the same as the "events" that users can post and wait on, as described in the SQL commands for CREATE EVENT, etc.

6. Near the top left of the diagram, you will see the text `"Start with '-x backupserver'"`. If you want to start a new Secondary server and you want it to get a copy of the database from the Primary via the "netcopy" command, then you start the server (from the operating system command line) with the command-line option **-x backupserver**. This tells the server to wait for a netcopy from the Primary. Note that while the server is waiting to receive the netcopy, the server will not respond to queries about its state (or role). For example, if you issue the command:

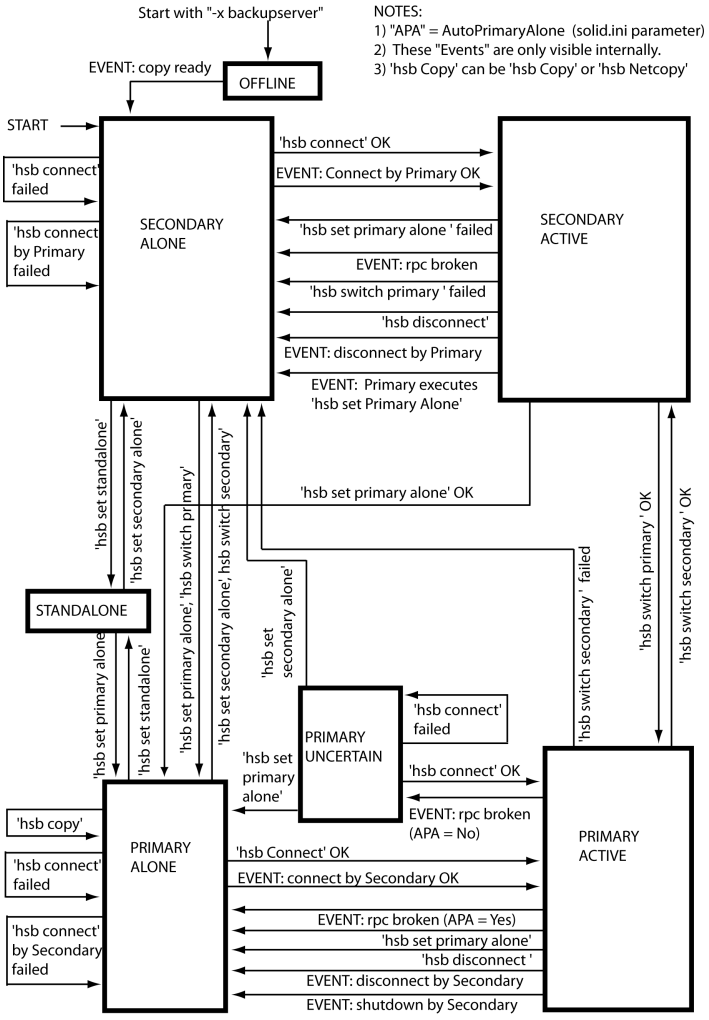
   ADMIN COMMAND 'hsb state';

   the server will not respond and therefore you won't actually see it return the state "OFFLINE".

7. "rpc" stands for "Remote Procedure Call". "rpc broken" means that the Primary and Secondary lost connection with each other without doing an explicit Disconnect. The connection may be lost if the network fails, or if one server crashes, etc.

8. When an arrow loops back to the same state that it started from, it means that the state doesn"t change. For example, if a server is in the state PRIMARY ALONE, and if it tries to connect to the other server but fails, then the state remains PRIMARY ALONE.

Figure D.1. HotStandby Server State Transitions



The following table shows server states and the ways in which a HotStandby command can change the server state.

Table D.1. Server State Transition Table

Server State	If this condition occurs, or if this HSB command is issued...	Then server state becomes...	If command is unsuccessful, then the state is...
OFFLINE	If the Primary server executes **ADMIN COMMAND 'hotstandby netcopy'** then the Secondary's state will change to SECONDARY ALONE after the database has been copied.	SECONDARY ALONE	unchanged
PRIMARY ACTIVE	HotStandby timeout (automatic) when *`AutoPrimaryAlone = Yes`*. NOTE: The HSB timeout occurs automatically when the Secondary server is down or a connection between the Primary and Secondary is broken.	PRIMARY ALONE	(not applicable)
PRIMARY ACTIVE	HotStandby timeout (automatic) when *`AutoPrimaryAlone = No`*. NOTE: The HSB timeout occurs automatically when the Secondary server is down or a connection between the Primary and Secondary is broken.	PRIMARY UNCERTAIN	(not applicable)
PRIMARY ACTIVE	ADMIN COMMAND 'hotstandby set standalone' at the Primary	STANDALONE	Unchanged
PRIMARY ACTIVE	**ADMIN COMMAND 'hotstandby switch secondary'** at the Primary or **ADMIN COMMAND 'hotstandby switch primary'** at the Secondary.	SECONDARY ACTIVE	SECONDARY ALONE
PRIMARY ACTIVE	**ADMIN COMMAND 'hotstandby disconnect'** at the Primary.	PRIMARY ALONE	PRIMARY ALONE
PRIMARY ALONE	**ADMIN COMMAND 'hotstandby copy'** or **ADMIN COMMAND 'hotstandby netcopy'** at the Primary. Note that the state of the Primary server does not change. The server stays in PRIMARY ALONE state. To change the	PRIMARY ALONE	PRIMARY ALONE

Server State	If this condition occurs, or if this HSB command is issued...	Then server state becomes...	If command is unsuccessful, then the state is...
	state to PRIMARY ACTIVE, you must issue the "connect" command: **ADMIN COMMAND 'hotstandby connect';** NOTE: If you are using a diskless server without file access to the Secondary server, you must use netcopy, not copy.		
PRIMARY ALONE	**ADMIN COMMAND 'hotstandby connect'** at the Primary NOTE: The above command is used to connect to the Secondary server, which is now fixed, or a server other than the failed Secondary.	PRIMARY ACTIVE (after the catchup is completed)	Unchanged
PRIMARY ALONE	**ADMIN COMMAND 'hotstandby set standalone'** at the Primary or the transaction log is full.	STANDALONE	Unchanged
PRIMARY ALONE	**ADMIN COMMAND 'hotstandby set secondary alone'** or **ADMIN COMMAND 'hotstandby switch secondary'** at the Primary.	SECONDARY ALONE	SECONDARY ALONE
PRIMARY UNCERTAIN	**ADMIN COMMAND 'hotstandby set primary alone'** at the Primary server	PRIMARY ALONE	Unchanged
PRIMARY UNCERTAIN	**ADMIN COMMAND 'hotstandby connect'** at the Primary. **Note** The above command is used to connect to the Secondary server (which is now fixed) or to connect to a server other than the failed Secondary.	PRIMARY ACTIVE	Unchanged
PRIMARY UNCERTAIN (HSB timeout	**ADMIN COMMAND 'hotstandby set standalone'** at the Primary	STANDALONE	Unchanged

Server State	**If this condition occurs, or if this HSB command is issued...**	**Then server state becomes...**	**If command is unsuccessful, then the state is...**
has occurred for connecting to the Secondary)			
PRIMARY UNCERTAIN	**ADMIN COMMAND 'hotstandby set secondary alone'** or **ADMIN COMMAND 'hotstandby switch secondary'** at the Primary.	SECONDARY ALONE	Unchanged
SECONDARY ACTIVE	HotStandby timeout (automatic) **Note** The HSB timeout occurs automatically when the Secondary server is down or a connection between the Primary and Secondary is broken.	SECONDARY ALONE	(not applicable)
SECONDARY ACTIVE	**ADMIN COMMAND 'hotstandby switch secondary'** at the Primary or **ADMIN COMMAND 'hotstandby switch primary'** at the Secondary.	PRIMARY ACTIVE	Unchanged
SECONDARY ACTIVE	**ADMIN COMMAND 'hotstandby set primary alone'** at the Secondary.	PRIMARY ALONE	Unchanged
SECONDARY ACTIVE	**ADMIN COMMAND 'hotstandby disconnect'** at the Secondary or Primary.	SECONDARY ALONE	SECONDARY ALONE
SECONDARY ALONE	**ADMIN COMMAND 'hotstandby connect'** at the Secondary or Primary	SECONDARY ACTIVE	Unchanged
SECONDARY ALONE	**ADMIN COMMAND 'hotstandby set standalone'** at the Secondary.	STANDALONE	Unchanged
SECONDARY ALONE	**ADMIN COMMAND 'hotstandby set primary alone'** or **ADMIN COMMAND 'hotstandby switch primary'** at the Secondary	PRIMARY ALONE	Unchanged

Appendix E. HSB System Events

This appendix covers only HSB-specific events. For a discussion of other types of events, see other manuals, such as solidDB SQL Guide.

Each HotStandby operation generates an event. To monitor these events you can use an application, such as a watchdog application.

Events are objects with a name that signal that a specific action occurred in the server. Special statements in stored procedures are required to receive events. HotStandby events are no different from other events created and supported by solidDB. They are sent to those users who are registered to receive the event in a stored procedure. For details on posting, registering, and waiting for events, read Chapter 3, "Stored Procedures, Events, Triggers, and Sequences", in *solidDB SQL Guide*, and Appendix B, Solid SQL Syntax, also in *solidDB SQL Guide*.

The following table lists the events that are currently available for HotStandby. Note that most events include five parameters, but not all of those parameters are necessarily used.

Table E.1. HotStandby Events

HSB Event	Event Parameters	Cause of Event
SYS_EVENT_HSBCONNECT-STATUS	*ENAME WVARCHAR, POSTSR-VTIME TIMESTAMP, UID IN-TEGER, NUMDATAINFO IN-TEGER, TEXTDATA WVARCHAR* For TEXTDATA, the possible valid values are `TEXTDATA = { CONNECTED | CONNECTING | CATCHUP | BROKEN}`	Change in connect status between the Primary and Secondary server
SYS_EVENT_HSB-STATESWITCH	*ENAME WVARCHAR, POSTSR-VTIME TIMESTAMP, UID IN-*	Each state switch sends a state switch event.

HSB Event	Event Parameters	Cause of Event
	TEGER, NUMDATAINFO INTEGER, TEXTDATA WVARCHAR For TEXTDATA, the possible valid values are `TEXTDATA = { PRIMARY ACTIVE | PRIMARY ALONE | PRIMARY UNCERTAIN | SECONDARY ACTIVE | SECONDARY ALONE | STANDALONE }`	
SYS_EVENT_NETCOPYEND	*ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER, TEXTDATA WVARCHAR* None of the parameters are used.	HotStandby NETCOPY operation ended. This event can be caught by the user *only* if the user is using AcceleratorLib.
SYS_EVENT_NETCOPYREQ	*ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER, TEXTDATA WVARCHAR* None of the parameters are used.	A HotStandby NETCOPY was requested. If the user application's callback function returns non-zero, then netcopy is not performed. This event can be caught by the user *only* if the user is using AcceleratorLib.

Glossary

This glossary describes some terms used in this manual.

Numeric

1-Safe Algorithm
: In a 1-safe system, transactions are committed at the primary first and then later are propagated to the Secondary. If the Primary server fails before it sends the transactions to the Secondary, then the transactions will not be visible on the Secondary, and normally will be lost.

  See also "2-safe algorithm".

2-Safe Algorithm
: A 2-safe algorithm is an algorithm for making sure that transactions are not lost when a Primary and a Secondary server are involved. Specifically, in a 2-safe algorithm, the Primary server does not consider the transaction committed until the Secondary has committed the transaction. The primary and the Secondary are kept in lock-step; all updates to the data are applied to both copies in complete synchrony. Thus we guarantee survival of all committed transactions in the case of failure. If, for some reason, the Secondary cannot commit (for example, the Secondary is no longer working), then the Primary will not commit the data; instead, the Primary will report an error to the user. The advantage of a 2-safe algorithm is, of course, that the Primary and Secondary cannot have different data. The disadvantage is that if the Secondary goes down, then the Primary stops accepting transactions. Furthermore, 2-safe algorithms mean that the user experiences a longer delay before receiving confirmation that a COMMIT was successful. Although Solid HotStandby defaults to a 2-safe algorithm, you can change it to use a 1-safe algorithm if the Secondary is unavailable.

  See also "1-safe algorithm".

B

Binary Large Object (BLOB)
: "BLOB" is an acronym for Binary Large OBject. A BLOB is a large block of information such as a picture, video clip, sound excerpt, or a document that contains any non-printable formatting characters.

  BLOB information is usually stored in a high capacity, variable-length binary data type. With solidDB database servers, BLOB data is usually stored in VARBINARY. However, this is not always necessary. Although BLOBs are generally Binary and Large, and are usually stored in variable-length data types, none of these characteristics are required. Depending upon the actual data value, you might store your

data in a fixed-length BINARY field rather than a variable-length VARBINARY field. If your data is composed entirely of standard characters, then you might store the data in one of the various high-capacity character data types, such as VARCHAR. (BLOBs that are composed entirely of printable characters are sometimes called CLOBs. Since BINARY fields can store any data that CHAR fields can store, CLOBs can be stored in either CHAR or BINARY fields. CLOBs are a subset of BLOBs.)

For a complete list of the BINARY and CHAR data types supported by solidDB, see "Binary Data Types" on page A-4 and "Character Data Types" on page A-1 in *solidDB SQL Guide*.

Note that if you are using in-memory tables, BLOB lengths are restricted to approximately the size of the page. See the appendices in *solidDB In-Memory Database User Guide* for an explanation of how to calculate the approximate maximum size of a BLOB in an in-memory table.

With the exception of the in-memory table restriction listed above, solidDB generally treats BLOB/CLOB the same way as any other BINARY/CHAR data. You do not need to do anything special to store or retrieve such data.

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. TCP/IP is an example of a commonly-used communication protocol.

D

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

Diskless
: A server that does not store the data on the disk drive is called a diskless server. If you are using the HotStandby functionality in a diskless server, then the Transaction Log file (see below) is not written to disk either.

  If you run a server without storing the data on the disk drive, then if the server is shut down, the data is lost. If you want to run diskless servers, then we recommend that you either replicate important data to another server (using Solid SmartFlow technology) or you use the HotStandby functionality to ensure that a copy of the data is on another server.

For more information about running diskless servers, see solidDB AcceleratorLib User Guide, which has a chapter on the topic and a description of the `SSCStartDisklessServer()` function.

Note that although running without a disk drive may increase performance by reducing disk I/O, Solid's diskless capability is not optimized to enhance performance. If you want to minimize I/O to maximize performance, we recommend that you use Solid In-memory Engine's in-memory tables feature.

H

High Availability
: High Availability is a system design and implementation that ensures a certain absolute degree of of operational continuity during a specified measurement period. In solidDB "HotStandby" is the name of the technique used by the CarrierGrade option. See also "HotStandby".

HotStandby (HSB)
: The Solid CarrierGrade option uses an approach known as "hot standby". In this approach, a second database server is linked to the first, and is ready to take over in case the first fails. Solid HotStandby technology ensures that the "secondary" server has an exact copy of all committed data that is on the "primary" server.

  Note that "hot standby" is the name of the general technique used to ensure high availability of data; "HotStandby" is the name of the Solid implementation of this general technique.

  The HSB abbreviation can be used in some administrative commands; e.g.

  ADMIN COMMAND 'hsb connect';

HotStandby Transaction Log
: To ensure that the Secondary server has a copy of all data that has been committed on the Primary server, the Primary writes data to a transaction log file, which is then read by the Secondary. Note that in version 4.1 and later, there is no separate HotStandby Transaction Log file; the server uses the regular transaction log file.

L

Log File
: See Transaction Log File.

N

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 parameter settings in the clients `solid.ini` file or in Microsoft Windows operating systems' registry settings.

O

One-Safe Algorithm
: See 1-Safe Algorithm

T

Transaction Log File
: 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. In version 4.1 and later, when the HotStandby functionality is used, this same transaction log is used to store transactions to send to the Secondary server. For more information, see Section 2.1.1, “The Transaction Log and HotStandby”.

Two-Safe Algorithm
: See 2-Safe Algorithm

U

UPS
: Uninterruptible Power Supply

Index

Symbols

-x autoconvert, 130
-x backupserver (command line option), 49
-x backupserver (command), 47
-x migratehsbg2, 130
1-Safe Algorithm
  defined, 183
1SafeMaxDelay (parameter), 138
2-Safe Algorithm
  defined, 183
2SafeAckPolicy (parameter), 138
=
  use of the equals sign when setting parameter values, 89

A

Access Mode, 136
  RO (read-only), 137
  RW (read-write), 137
  RW/Create, 137
  RW/Startup, 137
access rights, 30
Adaptive Durability, 19
ADMIN COMMAND 'hotstandby cominfo'
  viewing connect settings, 52
ADMIN COMMAND 'hotstandby connect'
  connecting HotStandby servers, 52
ADMIN COMMAND 'hotstandby copy'
  copying database contents, 50
ADMIN COMMAND 'hotstandby netcopy'
  copying database contents, 49
  copying to secondary, 47
ADMIN COMMAND 'hotstandby set primary alone'
  Running the server in PRIMARY ALONE state, 39
ADMIN COMMAND 'hotstandby set standalone'
  shutting off HotStandby operations, 41
ADMIN COMMAND 'hotstandby state'
  verifying server states, 56
ADMIN COMMAND 'hotstandby status connect'
  displaying connect status information, 54
ADMIN COMMAND 'hotstandby status copy'
  verifying a copy procedure, 46, 50
ADMIN COMMAND 'hotstandby status switch'
  verifying the switch process, 38
ADMIN COMMAND 'hotstandby status'
  querying HotStandby status, 52
ADMIN COMMAND 'hotstandby switch primary'
  switching server states, 35
ADMIN COMMAND 'hotstandby switch secondary'
  switching server states, 35
ADMIN COMMANDs
  list of available commands, 163
  list of HotStandby commands, 163
administering CarrierGrade option, 53, 54
  switching server states, 35
administrative commands, 163
applications
  switching to the new primary, 83
  using Basic Connectivity, 80
  using Transparent Connectivity, 66
autoconvert, 130
AutoPrimaryAlone (parameter), 23, 39, 97, 139
  and 'hotstandby switch' command, 36
AutoSwitch (parameter), 102, 144
Availability
  administrative commands, 163

B

backup, 8
backup listening mode, 47
  (see also netcopy listening mode)
BackupBlockSize (parameter), 98
BackupDeleteLog (parameter), 8
Basic Connectivity, 65
BLOB
  defined, 183
bringing a secondary back online, 40

C

CarrierGrade Option
  administering, 29
  configuring, 89
catchup, 12, 42
CatchupSpeedRate (parameter), 99, 139
CatchupStepsToSkip (parameter), 99
checkpoint, 8
CheckpointDeleteLog (parameter), 8
Choosing Which Server to Make Primary, 58
CLUSTER, 70
Communication protocol
  defined, 184
Configuration, 22
  HotStandby, 22
  watchdog, 23
configuration, 89
  Secondary and Primary node configuration, 90
  timeouts between applications and servers, 90
configuration and setup, 25
Configuration parameters, 133
Configuring CarrierGrade option
  netcopy performance, 98
  watchdog application, 99
Connect (parameter), 30, 60, 93, 139
connect settings
  viewing, 52
Connect1 (parameter), 100, 144
Connect2 (parameter), 100, 144
connection switch, 65
  in Transparent Connectivity, 76
connectivity
  basic, 65
  choosing connectivity type, 66
  Transparent Failover, 65
ConnectTimeout (parameter), 94, 95, 140
CopyDirectory (parameter), 96, 140
copying
  database contents, 49, 50
  primary to local secondary, 50
  verifying procedure, 46, 50
copying database contents, 49, 50
Copying Primary database to Secondary server over the network, 46
Creating a New Database for the Secondary Server, 47
current value, 136

D

Data management tools, 31
Database
  In-Memory Tables, 32
database
  copying contents, 49, 50
  verifying a copy procedure, 50
default value, 136
Defining Primary Server Behavior During a Secondary Failure, 97
Determining Whether the Primary's Settings Take Precedence Over the Secondary's, 135
displaying communication information, 54
displaying connect status information, 54
displaying switch status information, 53
Dual Primaries, 32
DualSecAutoSwitch (parameter), 145
DurabilityLevel (parameter), 92

E

Ensuring that Primary and Secondary Parameter Values Are Coordinated, 133
equals sign
  use of when setting parameter values, 89
Error Codes, 149
events, 181

F

factory value, 136
failure mode
  watchdog application, 106
failure scenarios and watchdog actions, 108
Failure Transparency, 66
  choosing connectivity type

CONNECTION, 66
NONE, 66
SESSION, 67

H

hotstadby copy, 178
HotStandby
administrative commands, 163
defined, 185
shutting off operations, 41
turning off, 60
upgrading your server, 127
HotStandby events
SYS_EVENT_HSBCONNECTSTATUS, 181
SYS_EVENT_HSBSTATESWITCH, 181
SYS_EVENT_NETCOPYEND, 182
SYS_EVENT_NETCOPYREQ, 182
hotstandby netcopy, 178
HotStandby recovery model
sending data, 14
synchronous hot standby, 13, 16
HotStandby Status, 52
HOTSTANDBY_CONNECTSTATUS
SQL function, 54
HOTSTANDBY_CONNECTSTATUS (SQL function), 83
HOTSTANDBY_STATE (SQL function), 84
hsb
abbreviation for "hotstandby" in admin commands, 27
HSB
defined, 185
hsb status catchup, 170
hsb status connect, 171
hsb status copy, 171
hsb status switch, 171
HSBEnabled (parameter), 29, 60, 93, 141

I

In-Memory Tables, 32
installation, 25

L

Listen (parameter), 90
load balancing
dynamic control of, 68
with Transparent Connectivity, 67
load balancing methods
PREFERRED_ACCESS=READ_MOSTLY, 67
PREFERRED_ACCESS=WRITE_MOSTLY, 67
LogEnabled (parameter), 93
logpos (admin command), 165
logpos (hotstandby command), 58

M

MaxLogSize (parameter), 141
MaxMemLogSize (parameter), 141
migratehsbg2, 130

N

netcopy, 98, 178
(see also netcopy listening mode)
Primary must be in PRIMARY ALONE state, 16
netcopy listening mode, 47, 49
ADMIN COMMAND 'hotstandby netcopy', 47
tuning performance, 98
NetcopyRpcTimeout (parameter), 141
Network name
defined, 186
network names, 90
Network Partitions, 32
Network Partitions and Dual Primaries, 32
NumRetry (parameter), 102, 145

O

OFFLINE (state), 178
One-Safe Algorithm
defined, 183

P

parameters
AutoPrimaryAlone, 36, 39, 97

AutoSwitch, 102, 144
BackupBlockSize, 98
BackupDeleteLog, 8
CatchupSpeedRate, 99
CatchupStepsToSkip, 99
CheckpointDeleteLog, 8
Connect, 30, 60, 93
Connect1, 100, 144
Connect2, 100, 144
ConnectTimeout, 94, 95
CopyDirectory, 96
DualSecAutoSwitch, 145
DurabilityLevel, 92
HSBEnabled, 29, 60, 93
Listen, 90
LogEnabled, 93
NumRetry, 102, 145
Password1, 101, 145
Password2, 101, 145
Pessimistic, 145
PingInterval, 94, 95, 102, 146
PingTimeout, 94, 95
ReadMostlyLoadPercentAtPrimary, 67
Username1, 101, 147
Username2, 101, 147
WatchdogLog, 103, 147
Partition
network, 32
Password1 (parameter), 101, 145
Password2 (parameter), 101, 145
Performing Read-Only Transactions on the Secondary, 21
performing recovery and maintenance, 35
Pessimistic (parameter), 145
ping, 95
PingInterval (parameter), 94, 95, 102, 142, 146
PingTimeout (parameter), 94, 95, 142
PRIMARY ACTIVE (state), 178
PRIMARY ALONE, 39
PRIMARY ALONE (state), 178, 179
PRIMARY ALONE state
running the new Primary in PRIMARY ALONE state, 39
PRIMARY UNCERTAIN (state), 178
PrimaryAlone (parameter), 142

R

READ COMMITTED
transaction isolation level, 32
Read-Only Transactions on the Secondary, 21
ReadMostlyLoadPercentAtPrimary (parameter), 67, 137
Reconnecting to Primary Servers from Applications, 80
recovery
maintenance, 35
REPEATABLE READ
transaction isolation level, 32
rights
access, 30
RO
Access Mode, 137
Running out of space for transaction log, 33
Running the server in PRIMARY ALONE state, 39
RW
Access Mode, 137
RW/Create
Access Mode, 137
RW/Startup
Access Mode, 137

S

Safeness Level, 13
secondary server
bringing back online, 40
SERIALIZABLE
transaction isolation level, 32
Server state
OFFLINE, 18
PRIMARY ACTIVE, 16
PRIMARY ALONE, 16, 39
PRIMARY UNCERTAIN, 17

SECONDARY ACTIVE, 17
SECONDARY ALONE, 18
STANDALONE, 18
server state transitions, 175
Server states
described, 16
server states
switching server states, 35
verifying, 56
servers
connecting, 52
SET TRANSACTION WRITE, 68
SET WRITE, 68
setup and configuration, 25
sever catchup, 12
(see also catchup)
sever names, 90
(see also network names)
Shutting Off HotStandby Operations, 41
SmartFlow
using solidDB CarrierGrade option with, 10
Solid Data Dictionary
defined, 31
Solid data management tools, 31
Solid Export
defined, 31
Solid SpeedLoader
defined, 31
SQL functions
HOTSTANDBY_CONNECTSTATUS, 54, 83
HOTSTANDBY_STATE, 84
STANDALONE (state), 40, 60, 178
state
OFFLINE, 178
PRIMARY ACTIVE, 178
PRIMARY ALONE, 178, 179
PRIMARY UNCERTAIN, 178
STANDALONE, 178
State switch
Secondary switch to new Primary, 11
state switch
verifying, 38
states
STANDALONE, 40, 60
transitions, 175
verifying server states, 56
status, 52
displaying communication information, 54
displaying connect status information, 54
displaying switch status information, 53
HotStandby, 52
list of, 54, 75, 76
Store Mode, 137
switching
displaying connect status information, 54
displaying switch status information, 53
switching states
verifying, 38
synchronization
using solidDB CarrierGrade option with SmartFlow, 10
synchronizing Primary and Secondary servers, 41
SYS_EVENT_HSBCONNECTSTATUS (event), 181
SYS_EVENT_HSBSTATESWITCH (event), 181
SYS_EVENT_NETCOPYEND (event), 182
SYS_EVENT_NETCOPYREQ (event), 182

T

TC Connection, 65
TC Info, 70
attribute combinations, 72
handling contradictions, 73
JDBC syntax, 73
proprietary ODBC attributes, 79
syntax, 70
TF Connectivity, 65
The Transaction Log and HotStandby, 7
Tools
Solid data management tools, 31
transaction isolation level
in-memory tables, 32
transaction log, 7, 14
running out of space for, 33

Transaction log file
defined, 186
transitions
server state transitions, 175
Transparent Connectivity, 65
Two-Safe Algorithm
defined, 183

U

upgrading your server, 127
cold and hot migration, 127
cold migration, 127
procedure, 129
hot migration, 127
procedure, 129
migration between hsb-compatible versions, 127
migration between hsb-incompatible versions, 128
preparing, 128
UPS
defined, 186
Username1 (parameter), 101, 147
Username2 (parameter), 101, 147

V

verifying a copy procedure, 50
verifying connect status information
HOTSTANDBY_CONNECTSTATUS, 83
verifying the state switch, 38
verifying the switch process
ADMIN COMMAND 'hotstandby status switch', 38
viewing current connect settings, 52

W

watchdog, 7
Watchdog
configuration, 23
watchdog actions and failure scenarios, 108
watchdog application
configuring, 99
defined, 12
described, 105
failure mode, 106
failure scenarios, 108
monitoring HotStandby with, 105
using, 107
watchdog failure scenarios
communication link between primary and secondary is down, 115
communication link between watchdog and primary is down, 117
communication link between watchdog and secondary is down, 119
communication links between watchdog and primary, and between primary and secondary, are down, 121
communication links between watchdog and secondary, and between primary and secondary, are down, 124
primary is down, 108
secondary is down, 110
watchdog is down, 113
WatchdogLog (parameter), 103, 147