



# IBM solidDB High Availability User Guide

Version 6.1 | June 2008

---

# IBM solidDB High Availability User Guide

Copyright © Solid Information Technology Ltd. 1993, 2008

Document number: SHS-6.01

Product version: 06.10.0014

Date: 2008-06-13

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

"IBM", the IBM logo, "DB2", "Informix", "Solid" and "solidDB" are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

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, 7136912, 6970876, 7139775, 6978396, and 7266702.

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 MERCHANTABILITY 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 contains the Qsort routine in the external sorter, Copyright (c) 1980, 1983, 1990 The Regents of the University of California. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
-

- 
3. All advertising materials mentioning features or use of this software must display the following acknowledgment: This product includes software developed by the University of California, Berkeley and its contributors.
  4. 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 BY THE REGENTS AND CONTRIBUTORS ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product contains the DES cipher (in ECB mode), parts of this code are Copyright (C) 1996 Geoffrey Keating. All rights reserved.

Its use is FREE FOR COMMERCIAL AND NON-COMMERCIAL USE as long as the following conditions are adhered to.

Copyright remains Geoffrey Keating's, and as such any Copyright notices in the code are not to be removed. If this code is used in a product, Geoffrey Keating should be given attribution as the author of the parts used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment: This product includes software developed by Eric Young (eay@mincom.oz.au)

THIS SOFTWARE IS PROVIDED BY GEOFFREY KEATING ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Parts of this code (in particular, the string representing SPtrans below) are Copyright (C) 1995 Eric Young (eay@mincom.oz.au). All rights reserved.

Its use is FREE FOR COMMERCIAL AND NON-COMMERCIAL USE as long as the following conditions are adhered to.

---

---

Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed. If this code is used in a product, Eric Young should be given attribution as the author of the parts used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment: This product includes software developed by Eric Young (eay@mincom.oz.au)

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product is assigned the U.S. Export Control Classification Number ECCN=5D992b.

---

---

# 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 Typographic Conventions .....                       | 2  |
| 1.2.2 Syntax Notation .....                               | 3  |
| 1.3 IBM solidDB Documentation .....                       | 4  |
| 1.4 Server Diagram Illustrations .....                    | 5  |
| 2 Introducing IBM solidDB HotStandby .....                | 7  |
| 2.1 How HotStandby Works .....                            | 8  |
| 2.1.1 HotStandby API (HSB Admin Commands) .....           | 8  |
| 2.1.2 Basic HotStandby Server Scheme .....                | 9  |
| 2.1.3 Heartbeat .....                                     | 9  |
| 2.1.4 The Transaction Log and HotStandby .....            | 9  |
| 2.1.5 Server HotStandby States .....                      | 11 |
| 2.1.6 Combining HotStandby and Advanced Replication ..... | 11 |
| 2.1.7 Failover .....                                      | 12 |
| 2.1.8 Server CatchUp .....                                | 13 |
| 2.1.9 Replication Modes in HotStandby .....               | 14 |
| 2.2 Description of Server States .....                    | 17 |
| 2.3 How Does HotStandby Affect Performance .....          | 19 |
| 2.3.1 Adaptive Durability .....                           | 20 |
| 2.4 High Availability Controller (HAC) .....              | 24 |
| 2.4.1 Recognized Failures .....                           | 25 |
| 2.4.2 Controlling Database Server Processes .....         | 26 |
| 2.4.3 External Reference Entity (ERE) .....               | 26 |
| 2.4.4 Networking in HAC .....                             | 28 |
| 2.4.5 High Availability Manager .....                     | 28 |
| 2.4.6 HAC Logging .....                                   | 30 |
| 3 Getting Started with HotStandby .....                   | 31 |
| 3.1 Before You Begin .....                                | 31 |
| 3.2 HotStandby Demonstration .....                        | 31 |
| 3.3 HotStandby Quick Start Procedure .....                | 32 |
| 3.4 Starting and Stopping HA Controller .....             | 34 |
| 3.5 HSB with HA Controller Quick Start Procedure .....    | 35 |
| 3.6 Summary of Start-up Sequences .....                   | 37 |
| 4 Administering and Configuring HotStandby .....          | 41 |
| 4.1 What You Should Know .....                            | 42 |

|   |    |
|---|----|
| 4.1.1 HotStandby and the IBM solidDB Configuration File .....   | 42 |
| 4.1.2 HotStandby and Access Rights .....  | 42 |
| 4.1.3 IBM solidDB Tools and HotStandby .....  | 43 |
| 4.1.4 Database Migration (disk-based servers only) .....  | 43 |
| 4.1.5 Interoperability .....  | 43 |
| 4.2 Limitations and Warnings with HotStandby .....  | 44 |
| 4.2.1 In-Memory Tables .....  | 44 |
| 4.2.2 Network Partitions and Dual Primaries .....   | 44 |
| 4.2.3 Running Out of Space for Transaction Logs .....   | 45 |
| 4.3 Overview of Administration Tasks .....  | 46 |
| 4.4 Performing HotStandby Recovery and Maintenance .....  | 47 |
| 4.5 Special Configurations: Lower Cost vs. Higher Safety .....  | 48 |
| 4.5.1 Reducing Cost: N + 1 Spare and N + M Spares Scenarios .....                                     | 48 |
| 4.5.2 Increasing Reliability: 2N + 1 Spare and 2N + M Spare Scenarios .....                           | 49 |
| 4.5.3 How IBM solidDB HSB Supports The N+1 (N+M) and 2N+1 (2N+M) Ap-<br>proaches .....                | 50 |
| 4.5.4 Using HAC with Spares .....   | 51 |
| 4.6 Configuring IBM solidDB for HotStandby .....  | 51 |
| 4.6.1 Defining Secondary and Primary Node Configuration (Com Section) .....                           | 52 |
| 4.6.2 Defining Timeouts Between Applications and Servers (Com Section) .....                          | 52 |
| 4.6.3 Transaction Durability .....  | 54 |
| 4.7 Configuring HotStandby-Specific Parameters .....  | 55 |
| 4.7.1 Defining Primary and Secondary HotStandby Configuration .....                                   | 55 |
| 4.7.2 Setting HotStandby Server Wait Time to Help Detect Broken or Unavailable Con-<br>nections ..... | 56 |
| 4.7.3 Defining a Name and Location for HotStandby Database Copy Operation .....                       | 58 |
| 4.7.4 Defining Primary Server Behavior During a Secondary Failure .....                               | 59 |
| 4.8 Performance Tuning .....  | 59 |
| 4.8.1 Tuning Replication Performance with Safeness and Durability Levels .....                        | 59 |
| 4.8.2 Tuning Netcopy Performance (General Section) .....  | 60 |
| 4.9 Configuring HA Controller and HA Manager .....  | 61 |
| 4.9.1 Section .....   | 61 |
| 4.9.2 LocalDB Section .....   | 63 |
| 4.9.3 RemoteDB Section .....  | 64 |
| 4.9.4 ERE Section .....   | 64 |
| 4.10 Configuration File Examples .....  | 64 |
| 4.10.1 The solid.ini Configuration File .....   | 65 |
| 4.10.2 The solidhac.ini Configuration File .....  | 65 |
| 4.10.3 The HAManager.ini Configuration File .....   | 69 |
| 5 Using HotStandby with Applications .....  | 71 |
| 5.1 Two Ways to Connect to HotStandby Servers .....   | 71 |
| 5.1.1 Transparent Connectivity .....  | 71 |

|  |     |
|--|-----|
| 5.1.2 Basic Connectivity .....   | 71  |
| 5.1.3 Choosing the Connectivity Type .....                                       | 72  |
| 5.2 Using the Transparent Connectivity .....                                     | 72  |
| 5.2.1 Failure Transparency in TC .....   | 72  |
| 5.2.2 Load Balancing in TC .....   | 73  |
| 5.2.3 Syntax of the TC Info .....  | 76  |
| 5.2.4 TC Info Attribute Combinations .....                                       | 79  |
| 5.2.5 Handling TC Info Contradictions .....                                      | 80  |
| 5.2.6 Enacting Transparent Connectivity in JDBC .....                            | 80  |
| 5.2.7 Programming for Connection Switch .....                                    | 83  |
| 5.3 Using the Basic Connectivity .....   | 87  |
| 5.3.1 Reconnecting to Primary Servers from Applications .....                    | 87  |
| 5.3.2 Re-Connecting to Secondary Servers .....                                   | 93  |
| 5.3.3 Advanced Replication Requirements .....                                    | 93  |
| 6 Using HotStandby API Commands .....  | 97  |
| 6.1 Switching Server States .....  | 98  |
| 6.1.1 Switchover and Failover .....  | 98  |
| 6.1.2 Performing Switchovers .....   | 98  |
| 6.1.3 Verifying the Switch .....   | 101 |
| 6.1.4 Performing Failovers .....   | 101 |
| 6.1.5 Running the New Primary in PRIMARY ALONE State .....                       | 101 |
| 6.1.6 Bringing the Secondary Server Back Online .....                            | 103 |
| 6.2 Shutting Off HotStandby Operations .....                                     | 103 |
| 6.3 Synchronizing Primary and Secondary Servers .....                            | 104 |
| 6.3.1 Catchup .....  | 104 |
| 6.3.2 Full Copy .....  | 105 |
| 6.3.3 Verifying the Copy .....   | 108 |
| 6.3.4 Using a Watchdog to Synchronize Servers .....                              | 108 |
| 6.3.5 Copying a Primary Database to a Secondary Over the Network .....           | 108 |
| 6.3.6 Creating a New Database for the Secondary Server .....                     | 109 |
| 6.3.7 Replacing an Existing Database on the Secondary Server .....               | 111 |
| 6.3.8 Verifying Netcopy Status .....   | 112 |
| 6.3.9 Copying a Database File from Primary Server to a Specified Directory ..... | 112 |
| 6.4 Connecting HotStandby Servers .....  | 114 |
| 6.5 Checking HotStandby Status .....   | 115 |
| 6.5.1 Displaying Switch Status Information .....                                 | 116 |
| 6.5.2 Displaying Connect Status Information .....                                | 116 |
| 6.5.3 Displaying Communication Information .....                                 | 117 |
| 6.5.4 Displaying Role Start Time .....   | 117 |
| 6.6 Verifying HotStandby Server States .....                                     | 118 |
| 6.6.1 Server State Combinations .....  | 119 |
| 6.7 Choosing Which Server to Make Primary .....                                  | 120 |

|   |     |
|---|-----|
| 6.8 Changing a HotStandby Server to a Non-HotStandby Server .....                         | 121 |
| 7 Behavior of High Availability Controller in Failure Cases .....                         | 123 |
| 7.1 Primary Database Fails .....  | 123 |
| 7.2 Secondary Database Fails .....  | 124 |
| 7.3 Primary Node Fails .....  | 125 |
| 7.4 Secondary Node Fails .....  | 126 |
| 7.5 HotStandby Link Fails .....   | 127 |
| 8 Upgrading Your Server by Using HotStandby .....   | 129 |
| 8.1 Cold and Hot Migration .....  | 129 |
| 8.2 Migration between HSB-Compatible Versions .....                                       | 129 |
| 8.2.1 Cold Migration .....  | 129 |
| 8.2.2 Hot Migration .....   | 129 |
| 8.3 Migration between HSB-Incompatible Versions .....                                     | 130 |
| 8.3.1 Preparation Steps .....   | 130 |
| 8.3.2 After the Upgrade .....   | 134 |
| A Configuration Parameters .....  | 135 |
| A.1 Ensuring that Primary and Secondary Parameter Values Are Coordinated .....            | 135 |
| A.2 Determining Whether the Primary's Settings Take Precedence Over the Secondary's ..... | 137 |
| A.3 Querying HotStandby Configuration Parameters .....                                    | 137 |
| A.4 Modifying HotStandby Configuration Parameters .....                                   | 138 |
| A.5 Access Mode .....   | 139 |
| A.5.1 Access Mode Values .....  | 139 |
| A.5.2 Saving Parameter Changes .....  | 139 |
| A.6 Cluster Section .....   | 139 |
| A.7 HotStandby Section .....  | 140 |
| A.8 High Availability Controller Configuration Parameters .....                           | 144 |
| A.9 High Availability Manager Configuration Parameters .....                              | 148 |
| B Error Codes .....   | 151 |
| B.1 HotStandby Errors and Status Codes .....  | 151 |
| B.2 High Availability Controller Errors and Status Codes .....                            | 160 |
| B.3 IBM solidDB Database Errors .....   | 163 |
| B.4 IBM solidDB Table Errors .....  | 165 |
| B.5 IBM solidDB Communication Errors .....  | 166 |
| C Summary of HotStandby Administrative Commands .....                                     | 167 |
| C.1 HotStandby Commands .....   | 168 |
| C.2 High Availability Controller Commands .....   | 177 |
| D Server State Transitions .....  | 179 |
| D.1 HotStandby State Transition Diagram .....   | 179 |
| E HSB System Events .....   | 185 |
| F Watchdog Sample .....   | 187 |
| F.1 HotStandby Configuration Using Watchdog .....   | 187 |
| F.1.1 How the Watchdog Application Works .....  | 188 |



|  |     |
|--|-----|
| F.1.2 System Design Issues .....   | 190 |
| F.1.3 Watchdog Configuration .....   | 191 |
| F.1.4 Using the Sample Watchdog Application .....  | 192 |
| F.2 Failure Situations and Watchdog Actions .....  | 193 |
| F.2.1 Primary is Down .....  | 193 |
| F.2.2 Secondary is Down .....  | 195 |
| F.2.3 Watchdog Is Down .....   | 198 |
| F.2.4 Communication Link Between Primary and Secondary Is Down .....   | 200 |
| F.2.5 Communication Link Between Watchdog and Primary Is Down .....  | 202 |
| F.2.6 Communication Link Between Watchdog and Secondary Is Down .....  | 204 |
| F.2.7 Communication Links Between Watchdog and Primary, and Between Primary and<br>Secondary, Are Down .....   | 206 |
| F.2.8 Communication Links Between Watchdog and Secondary, and Between Primary and<br>Secondary, Are Down ..... | 209 |
| F.3 Watchdog Section of <code>solid.ini</code> Configuration File .....  | 211 |
| Glossary .....   | 217 |
| Index .....  | 221 |

---

---

# List of Figures

|  |     |
|--|-----|
| 1.1 Illustration Key .....   | 6   |
| 2.1 HotStandby Architecture .....  | 7   |
| 2.2 HotStandby Server Scheme .....   | 9   |
| 2.3 HotStandby with Master and Replica Server Scheme .....   | 12  |
| 2.4 HotStandby Switchover to New Primary (old Secondary) .....   | 13  |
| 2.5 Server Failover and Catchup Example .....  | 14  |
| 2.6 Synchronous HotStandby Configuration .....   | 15  |
| 2.7 High Availability Controller Architecture .....  | 24  |
| 2.8 External Reference Entity Components .....   | 27  |
| 2.9 High Availability Manager .....  | 29  |
| 3.1 Summary of Start-up Sequences .....  | 38  |
| 5.1 Master Failover .....  | 94  |
| 5.2 Replica Failover .....   | 95  |
| 6.1 State Switch .....   | 99  |
| 6.2 Manual Full Copy Procedure .....   | 107 |
| D.1 HotStandby Server State Transitions .....  | 181 |
| F.1 Heterogeneous HotStandby Configuration with Watchdog .....   | 191 |
| F.2 Primary is Down Scenario and Remedy .....  | 194 |
| F.3 Secondary is Down Scenario and Remedy .....  | 197 |
| F.4 Watchdog is Down Scenario and Remedy .....   | 199 |
| F.5 Broken Link Between Primary and Secondary Scenario and Remedy .....                                    | 201 |
| F.6 Broken Link Between Watchdog and Primary Scenario and Remedy .....                                     | 203 |
| F.7 Broken Link Between Watchdog and Secondary Scenario and Remedy .....                                   | 205 |
| F.8 Broken Link Between Watchdog and Primary, and between Primary and Secondary, Scenario and Remedy ..... | 207 |
| F.9 Broken Link between Watchdog and Secondary and between Primary and Secondary Scenario and Remedy ..... | 210 |

---

---

# List of Tables

|     |  |     |
|-----|--|-----|
| 1.1 | Typographic Conventions .....  | 2   |
| 1.2 | Syntax Notation Conventions .....                                    | 3   |
| 2.1 | Description of Server States .....                                   | 17  |
| 3.1 | Installation Sequence Steps .....                                    | 38  |
| 4.1 | Administration Tasks .....   | 46  |
| 5.1 | Choosing the Connectivity Type .....                                 | 72  |
| 5.2 | TC Info Abbreviations .....  | 77  |
| 5.3 | Possible Combinations of TC Info Attributes .....                    | 79  |
| 5.4 | Connect Request Errors .....   | 82  |
| 5.5 | Warnings .....   | 82  |
| 5.6 | Connection Switch Request .....                                      | 83  |
| 5.7 | Communication Link Failure .....                                     | 84  |
| 5.8 | Session State Preservation .....                                     | 85  |
| 5.9 | HOTSTANDBY_CONNECTSTATUS Status Values .....                         | 90  |
| 6.1 | ADMIN COMMAND 'hotstandby status' Options .....                      | 115 |
| 6.2 | Connect Status Return Values .....                                   | 116 |
| 6.3 | Server States .....  | 119 |
| 8.1 | Hot Migration .....  | 131 |
| A.1 | Cluster Parameters .....   | 139 |
| A.2 | HotStandby Parameters .....  | 140 |
| A.3 | HAC Configuration Parameters: [ <i>HACController</i> ] Section ..... | 145 |
| A.4 | HAC Configuration Parameters: [ <i>LocalDB</i> ] Section .....       | 146 |
| A.5 | HAC Configuration Parameters: [ <i>RemoteDB</i> ] Section .....      | 147 |
| A.6 | HAC Configuration Parameters: [ <i>ERE</i> ] Section .....           | 147 |
| A.7 | High Availability Manager Configuration Parameters .....             | 148 |
| B.1 | HotStandby Errors and Status Codes .....                             | 151 |
| B.2 | High Availability Controller Errors and Status Codes .....           | 160 |
| B.3 | IBM solidDB Database Errors .....                                    | 163 |
| B.4 | IBM solidDB Table Errors .....                                       | 165 |
| B.5 | IBM solidDB Communication Errors .....                               | 166 |
| C.1 | HotStandby Commands .....  | 168 |
| C.2 | High Availability Controller Commands .....                          | 177 |
| D.1 | Server State Transition Table .....                                  | 182 |
| E.1 | HotStandby Events .....  | 185 |
| F.1 | Watchdog Parameters .....  | 212 |

---

---

## List of Examples

|  |     |
|--|-----|
| 3.1 Starting HAC .....                             | 35  |
| 4.1 Entry in the <code>solid.ini</code> File ..... | 52  |
| 4.2 Partial <code>solid.ini</code> Files .....     | 55  |
| 5.1 Client-side INI File .....                     | 78  |
| 5.2 Connect String in ODBC .....                   | 79  |
| 5.3 Using Transparent Connectivity in JDBC .....   | 81  |
| 5.4 Basic Connection without Transparency .....    | 87  |
| 5.5 Sample Pseudo-Code .....                       | 91  |
| F.1 Dual Primaries .....                           | 190 |

---



---

# Chapter 1. Welcome

The IBM solidDB HotStandby component increases the reliability of your database system, reducing downtime. HotStandby 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, the High Availability Controller (HAC) makes a switch over to the secondary, transparently to applications and 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 HotStandby as part of their IBM solidDB package. It contains information specific to the HotStandby component only.

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

### 1.1.1 Organization

This guide includes the following information:

- Chapter 2, *Introducing IBM solidDB HotStandby*, describes the concepts, components, and physical configuration options of HotStandby.
- Chapter 3, *Getting Started with HotStandby*, provides step-by-step instructions to start two IBM solidDB HotStandby servers (a Primary server and a Secondary server).
- Chapter 4, *Administering and Configuring HotStandby*, 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 HotStandby.
- Chapter 6, *Using HotStandby API Commands*, describes how the high availability control works in IBM solidDB. This chapter can be used to program your own application to manage high availability in IBM solidDB. Topics covered include switching server states, connecting HotStandby servers, choosing which server to make primary, and so on.

- Chapter 7, *Behavior of High Availability Controller in Failure Cases*, explains possible failure scenarios and what commands can be issued to recover from those scenarios. Normally, the High Availability Controller (HAC) implicitly handles failure scenarios.
- 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` and `solidhac.ini` configuration parameters that are used with HotStandby.
- Appendix B, *Error Codes*, provides detailed information about error messages related to HotStandby.
- Appendix C, *Summary of HotStandby Administrative Commands*, provides a reference of commands used to administer HotStandby.
- 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.
- Appendix F, *Watchdog Sample*, discusses the watchdog sample application available in the samples included in your IBM solidDB installation.

## 1.1.2 Audience

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

# 1.2 Conventions

## 1.2.1 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.                              |
| NOT NULL               | Uppercase letters on this font indicate SQL keywords and macro names. |
| <code>solid.ini</code> | These fonts indicate file names and path expressions.                 |

| Format                                  | Used for   |
|---|--|
| SET SYNC MASTER YES;<br>COMMIT WORK;    | This font is used for program code and program output. Example SQL statements also use this font.  |
| <b>run.sh</b>                           | 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.   |
| <i>LockHashSize</i>                     | This font is used for parameter names, function arguments, and Windows registry entries.   |
| <i>argument</i>                         | Words emphasised like this indicate information that the user or the application must provide.   |
| <i>IBM solidDB Administration Guide</i> | 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. |

## 1.2.2 Syntax Notation

This manual uses the following syntax notation conventions:

**Table 1.2. Syntax Notation Conventions**

| Format                        | Used for   |
|-------------------------------|--|
| INSERT INTO <i>table_name</i> | Syntax descriptions are on this font. Replaceable sections are on <i>this</i> font.                |
| 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.                          |

| Format            | Used for   |
|-------------------|--|
| { }               | 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.  |
| .<br>. .<br>. . . | A column of three dots indicates continuation of previous lines of code.   |

## 1.3 IBM solidDB Documentation

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

- *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.
- *IBM solidDB Getting Started Guide*. This manual gives you an introduction to IBM solidDB.
- *IBM solidDB SQL Guide*. This manual describes the SQL commands that IBM 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 IBM solidDB "administrative commands" related to the High Availability (HotStandby) component are described in the *IBM solidDB High Availability User Guide*, not the *IBM solidDB SQL Guide*.
- *IBM solidDB Administration Guide*. This guide describes administrative procedures for IBM solidDB servers. This manual includes configuration information. Note that some administrative commands use an SQL-like syntax and are documented in the *IBM solidDB SQL Guide*.
- *IBM solidDB Programmer Guide*. This guide explains in detail how to use features such as IBM solidDB Stored Procedure Language, triggers, events, and sequences. It also describes the interfaces (APIs and drivers) available for accessing IBM solidDB and how to use them with a IBM solidDB database.
- *IBM solidDB In-Memory Database User Guide*. This manual describes how to use the IBM solidDB in-memory database and main memory engine (MME).

- *IBM solidDB Advanced Replication Guide.* This guide describes how to use the IBM solidDB advanced replication technology to synchronize data across multiple database servers.
- *IBM solidDB Linked Library Access User Guide.* Linking the client application directly to the server improves performance by eliminating network communication overhead. This guide describes how to use the linked library access, 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 IBM solidDB 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 IBM solidDB to run without a disk drive.

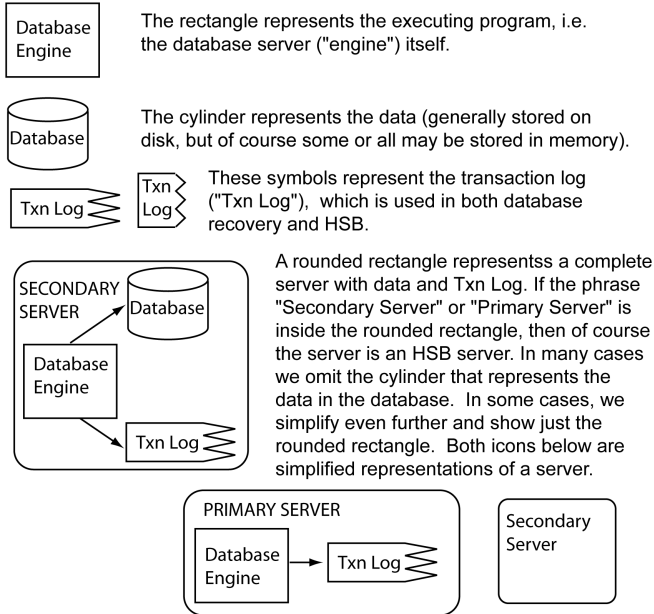
- *IBM solidDB High Availability User Guide.* IBM solidDB HotStandby allows your system to maintain an identical copy of the database in a backup server or "secondary server". This secondary database server can continue working if the primary database server fails.
- *IBM solidDB Connector Guide.* This guide explains in detail how to use the IBM solidDB Cache solution. IBM solidDB Cache provides a high-performance, low-latency database front-end solution for IBM Data Servers, namely DB2™ and Informix™. IBM solidDB Cache solution uses a number of in-memory front-end databases to handle high-volume traffic from the applications. The connectors are applications that manage data between the back-end and the front-ends in IBM solidDB Cache.

## 1.4 Server Diagram Illustrations

This document contains several server diagrams depicting different scenarios in the HotStandby environment. The figure below provides an illustration key for the server diagrams:

### Figure 1.1. Illustration Key

This figure explains the symbols used in the Hot Standby (HSB) illustrations.



---

# Chapter 2. Introducing IBM solidDB HotStandby

This chapter describes the IBM solidDB HotStandby component. HotStandby 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. The server states are controlled by an entity called a watchdog. The IBM solidDB HotStandby component contains a watchdog implementation called the IBM solidDB High Availability Controller (HAC).

Internally, HAC uses the HotStandby commands to control the server states. A solution such as 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 IBM solidDB (such as IBM solidDB master or replica), HotStandby allows the secondary database to replace the failed one.



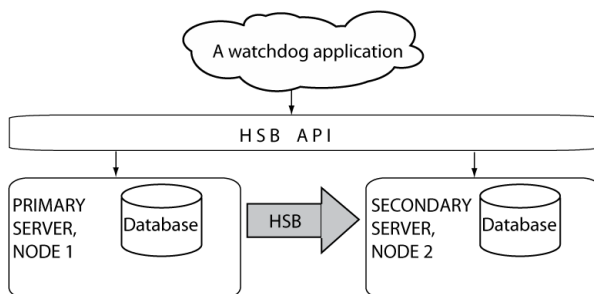
## Note

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 IBM solidDB's specific implementation of this general technique. The abbreviation for HotStandby is HSB.

Similarly, a watchdog refers to a technology that supervises the state of two databases and can switch the states, if necessary. HAC is the IBM solidDB watchdog implementation. The Watchdog sample that is presented in one of the appendixes is a programming example that shows how to use the HSB API to program your own watchdog application.

See below for a conceptual presentation of the HotStandby architecture:

**Figure 2.1. HotStandby Architecture**



Looking at the figure above, HotStandby provides:

- A watchdog application called the IBM solidDB High Availability Controller (HAC).
- The HotStandby API (HSB admin commands)
- IBM solidDB servers

## 2.1 How HotStandby Works

HotStandby (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 HotStandby API (HSB Admin Commands)

The HSB servers' HA behavior is controlled with an API that is based on a subset of IBM solidDB admin commands. This subset is identified by the main command being **hotstandby** that can be abbreviated by **hsb**. These commands can be issued by using any SQL-capable tool (like solsql) or programmatic interfaces like ODBC or JDBC. The syntax of the HotStandby admin commands is the following:

```
admin command 'hotstandby hsb-command options';
```

or

```
admin command 'hsb hsb-command options';
```

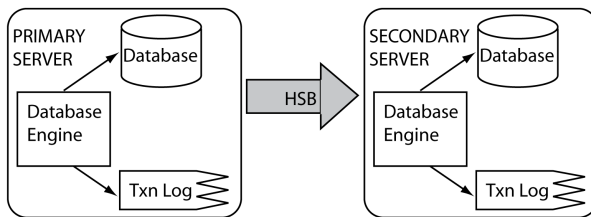
In the sequel, it will be shown how you can use the hsb admin commands to control the HA state of a IBM solidDB HSB server, or to retrieve state information. Note that the commands can be issued manually or programmatically. IBM solidDB HA management tools, HA Controller and Watchdog, use the commands programmatically.



## 2.1.2 Basic HotStandby Server Scheme

Figure 2.2, “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 *IBM solidDB Administration Guide* for a more detailed explanation of logging and recovery.)

**Figure 2.2. HotStandby Server Scheme**



## 2.1.3 Heartbeat

Internally, IBM solidDB HSB uses a technique, which is referred to as heartbeat, to monitor the connection between servers. A sequence of keepalive messages are sent between active and standby servers. Both servers continuously send these unidirectional "I am here" messages to the other server. The messages are sent on a fixed time interval. Counterwise, a message from the other server is expected to arrive within a predefined time window. In IBM solidDB, the heartbeat technique is called "ping".

### ! Important

In IBM solidDB the heartbeat technique is called "pinging" although there are no ping requests sent. It should not be confused with the Ping protocol used in TCP/IP networks.

## 2.1.4 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.

If the Primary server fails, a watchdog application tells the secondary to become the Primary. Once the new Primary is in operation, the clients can connect to it 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 fails, 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.<sup>1</sup>

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 4.2.3, “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.

---

<sup>1</sup>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 *IBM 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 Section 4.8, “Performance Tuning”.)

On the other hand, you can allow for the safeness level to change dynamically in relation to the durability level by using the *SafenessLevel* parameter "auto" value. For more information on the "auto" value, refer to the *SafenessLevel* parameter description in Section A.7, “HotStandby Section”.

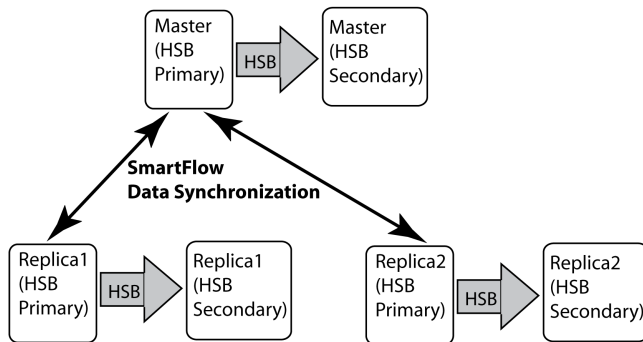
## 2.1.5 Server HotStandby States

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 HAC 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.

## 2.1.6 Combining HotStandby and Advanced Replication

The IBM solidDB HotStandby component can be used in combination with IBM solidDB advanced replication. Advanced replication provides bi-directional, periodically occurring data synchronization that allows you to create a distributed system that contains "master" and "replica" servers. With HotStandby, 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. Advanced replication 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. Advanced replication can now continue with the new Primary server.

**Figure 2.3. HotStandby with Master and Replica Server Scheme**

## 2.1.7 Failover

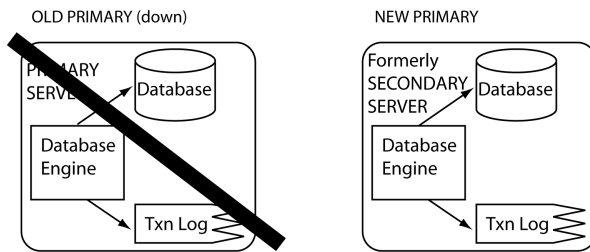
In a failover, the secondary is switched to be the new primary. There are several reasons for switching the secondary to new primary:

1. when the primary fails
2. when you want to administer the primary
3. when you must choose a primary when there is no existing primary on the system.

The secondary is switched to be the new primary by issuing the command below on the Secondary server:

**ADMIN COMMAND 'hotstandby set primary alone';**

In the case of a failover, 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. If Transparent Connectivity (TC) is used, connections are not lost on the failover. However, the on-going transactions are aborted and must be re-executed. For more information, see Section 5.2.1, "Failure Transparency in TC". 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)**

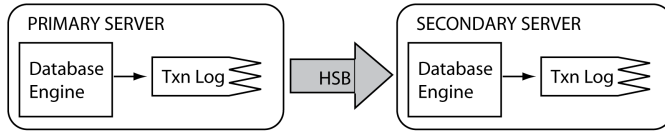
The server that was originally the Secondary becomes the new Primary after the old Primary server fails.

## 2.1.8 Server CatchUp

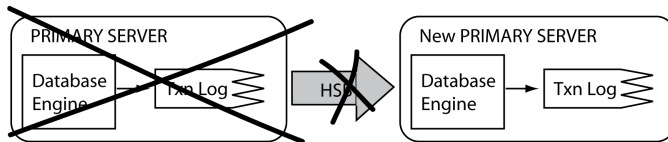
Once the old Primary is back online - assuming that there is an existing Primary - it becomes the new Secondary. At this stage, the information in the new Secondary lags behind that of the new Primary as new transactions have been committed to the new Primary database. To bring the new Secondary up to 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 6.3, "Synchronizing Primary and Secondary Servers".

**Figure 2.5. Server Failover and Catchup Example**

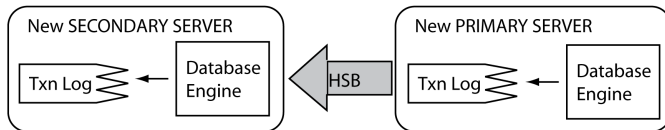
1) Normal operation. Primary sends data to Secondary.



2) Primary fails. Secondary takes over as new Primary. New Primary saves transaction information in its transaction log so that it can send the data to the new Secondary later.



3) After the old Primary is brought up as the new Secondary, the information in the new Primary's Txn Log is sent to the new Secondary so it can "catch up".



## 2.1.9 Replication Modes in HotStandby

### 1-Safe and 2-Safe Replication

IBM solidDB 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, IBM solidDB 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.)<sup>2</sup>

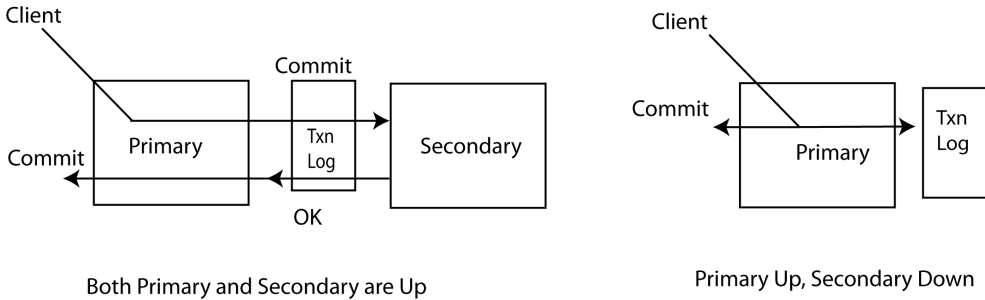
Before committing changes to a transaction in the Primary database, the Primary server sends the transaction data to the Secondary server. The Secondary server must send acknowledgement to the Primary that it has committed (or at least received) the data. Otherwise, the Primary server times out and changes 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 HAC 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**

The Secondary server sends an 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:

<sup>2</sup>For more about 2-safe vs. 1-safe algorithms, see *Transaction Processing: Concepts and Techniques*, by Jim Gray and Andreas Reuter, Morgan Kaufman, 1993.

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.)
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 HAC detects that the Secondary is down or the network failed, then it 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 cannot 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 Hot-



Standby **netcopy** command. For details on **copy** and **netcopy**, read Section 6.3.5, “Copying a Primary Database to a Secondary Over the Network” and Section 6.3.9, “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 6.4, “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 can 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. |

| STATE             | DESCRIPTION   |
|-------------------|---|
| PRIMARY ALONE     | <p>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).</p> <p>This server (the Primary) is actively accepting and executing read-write transactions and collecting them to be sent to the Secondary later.</p>  |
| PRIMARY UNCERTAIN | <p>The servers have disconnected abnormally and the <i>AutoPrimaryAlone</i> 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 HAC changes the server to another state.</p> <p>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.</p> <ol style="list-style-type: none"> <li>1. If the server is re-connected to the Secondary, then the transactions are committed on the Primary.</li> <li>2. If the state is changed to PRIMARY ALONE, then the open transactions are committed on the Primary.</li> <li>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.</li> </ol> <p>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 <i>AutoPrimaryAlone</i> configuration parameter.</p> <p> <b>Note</b></p> <p>HAC 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 4.2.2, “Network Partitions and Dual Primaries”.)</p> |

| STATE            | DESCRIPTION  |
|------------------|--|
| 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 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 <b>netcopy</b> or <b>copy</b> operation to send a complete copy of the database to the Secondary.  |
| OFFLINE          | <p>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.</p> <p>You cannot directly observe the OFFLINE state because a server in OFFLINE state does not accept client connections. If you attempt to connect to a server in the OFFLINE state, error code 14552 "Server is in backup server mode, no connections are allowed" is given. A server in the OFFLINE state will respond only to a netcopy operation (described later).</p> |

## 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) — that is, 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 be 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 *IBM 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". For more information on the "auto" value of the *SafenessLevel* parameter, see the *SafenessLevel* parameter description in Section A.7, "HotStandby Section".

## 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. IBM solidDB 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) component, 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.

#### **Tip**

You can run the backup (using the **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 (Adaptive Durability and 2-safe received mode), you are still protected by HSB unless there are at least two failures. You sacrifice very little safety for much higher performance. IBM solidDB HotStandby gives you high performance and safety at the same time.

## 2.4 High Availability Controller (HAC)

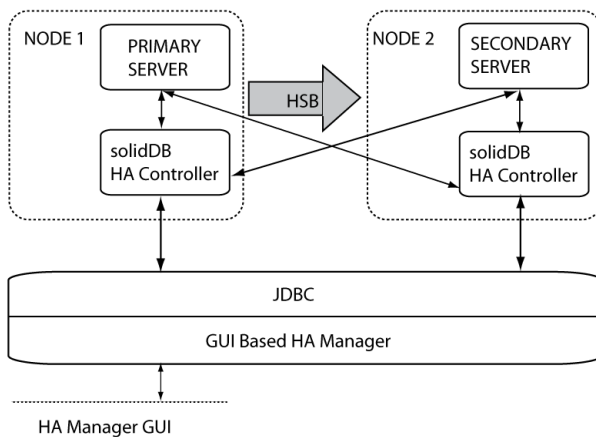
High Availability Controller (HAC) is an automatic redundancy management program for IBM solidDB HotStandby configurations. It maintains the availability of the database service by detecting failures, performing failovers to standby units, and restarting failed processes when necessary.

A failure can be caused by a hardware problem in a database node, a database process failure, or a broken HSB link. HAC monitors the HSB states of the servers, and in the case of a failure, ensures that a server, which is not affected by the failure, holds the Primary role, and that the server is ready to accept the transaction load.

In other words, HAC plays the role of a watchdog program. In its implementation, the IBM solidDB's event mechanism is used to monitor the server states. Every time the state of an HSB server changes, it sends an event to HAC, which then deduces potential needs for actions that are executed by using the HotStandby admin command API.

The HAC architecture is depicted in the figure below:

**Figure 2.7. High Availability Controller Architecture**



HAC has two main purposes:

1. HAC can be used as a watchdog, to automatically maintain the availability of the database service. In this mode, called the AUTOMATIC mode, HAC performs the following actions:
  - starts, restarts and terminates the database server processes (optional)



- monitors the state of the servers and the HSB link between them
  - infers the necessary action to be performed
  - performs the action
2. HAC can also play a role of a monitor for a HSB system by monitoring the state changes of the HSB servers (triggered by someone else) and reporting the status of the system. This is called the ADMINISTRATIVE mode. In this mode, HAC does not execute any HSB state transitions, or otherwise modify the HSB system.

A Graphical User Interface (GUI) component called High Availability Manager (HAM) is also available for HAC. The download information is available at the IBM solidDB product web portal<sup>3</sup>.

High Availability Controller is configured through the `solidhac.ini` configuration file. Before HAC is started, this file should be located in the HAC working directory. In the product package, there is a configuration file template, which includes all available configuration parameters with comments, and examples. The HA Manager is configured in a similar way.

### 2.4.1 Recognized Failures

HAC monitors the health and status of the HotStandby servers. In failure situations, such as a database process failure or a computer node failure, HAC performs failovers and other necessary state transitions to maintain the best possible availability of the database service.

For all failures considered, it is assumed that they happen in a normal, fully operational state expressed by the PRIMARY ACTIVE and SECONDARY ACTIVE states of the two HSB servers. HAC takes care of single failures only. In other words, it is assumed that a failure cannot occur before the system has recovered from a previous failure. There is, however, a number of pre-defined multiple-failure scenarios that HAC can handle.

As far as single failures are concerned, HAC maintains an almost uninterruptable database service. If multiple failures occur, HAC attempts to avoid an erroneous system state (such as dual Primaries).

The failures HAC can handle are:

- Single failures
  - The primary (ACTIVE) database server process fails
  - The secondary (ACTIVE) database server process fails

---

<sup>3</sup> <http://www.ibm.com/software/data/soliddb>

- Primary node fails
- Secondary node fails
- If an External Reference Entity is used, HAC can also handle a HotStandby link failure, that is, a lost connection between the two HotStandby database processes. For more information on the External Reference Entity, see Section 2.4.3, “External Reference Entity (ERE)”.
- Double failures
  - While recovering from a previous failure, HAC recognizes an error in the synchronization between the Primary and the Secondary database.
  - HAC also takes care several less common failures, such as a server process failure while servers are establishing HSB link after previous failure, etc.

## 2.4.2 Controlling Database Server Processes

HAC can be configured to start database processes, and restart failed processes. When a HAC instance loses connection with a local database server, it calls the start script specified in the `solidhac.ini` configuration file. The script is provided by the user. An example script is provided with the package.

### Important

A HAC instance assumes that the server is running, and responsive at the moment the start script terminates. Since HAC does not handle failures that occur in server startup, the script should not exit unless the server accepts connections.

HAC restarts the database server whenever it fails or disappears for some other reason, except in two cases:

- if the database process has not disappeared from the process list of the operating system, or
- if the database server was shut down by using HA Manager.

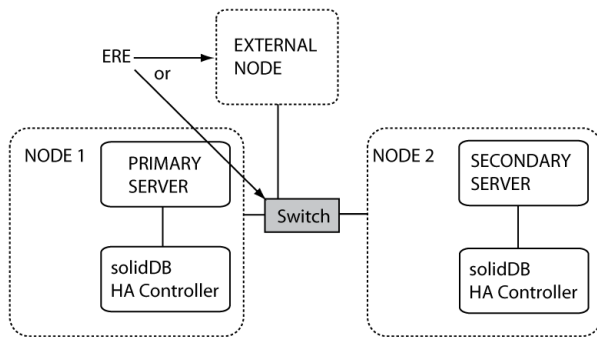
## 2.4.3 External Reference Entity (ERE)

One of the more difficult failure situations to be handled is when the communication link between the database nodes fails, and both database servers might assume that the other one is down. This can lead into a dual primary ('split brain') situation and possibly losing transactions when databases are later synchronized. To better avoid a wrong decision by IBM solidDB HAC, it is recommended that an External Reference Entity, ERE (i.e. a network reference device) is used for checking the health of the network. For example, in the case of a network adapter failure in one computer, HAC can detect this situation, and is able to set the correct database node to continue as the Primary database (while the other one continues as Secondary).

If ERE is used, HAC checks the status of the physical link between the HotStandby node and the ERE device by pinging ERE. If the physical link to the nearest ERE is not operational, the local HAC sets the local server to the **SECONDARY ALONE** state. If the nearest physical link is operational, and no connection is available to the other server, the local HAC concludes that the local server is the one to continue offering the service, and sets it to **PRIMARY ALONE**. Consequently, The HotStandby node, which loses its connection to the opposite HotStandby node and to the nearest ERE, becomes the Secondary. In this way, two Primaries (=split brain) situation is prevented in the case of network failure.

For further information about configuring HAC for ERE, see Section 4.9, “Configuring HA Controller and HA Manager”.

**Figure 2.8. External Reference Entity Components**



The figure above depicts two possible locations of ERE:

- the cluster switch, or
- any computer in the network outside the cluster. If a redundant network (that is, duplicate network controllers, cables and switches) is used in a cluster, it is recommended to define ERE outside the cluster.

### **!** Important

If the HotStandby link is considered unreliable (including all the cases where ERE is used), the following HotStandby server parameter must be set to its factory value:

```
HotStandby.AutoPrimaryAlone=no
```

ERE must use the same HSB link that the keepalive messages do.

### 2.4.4 Networking in HAC

It is recommended to use a single logical network access (IP address), in each HSB server, for all communications. This recommendation does not preclude the use of multiple, or redundant, lower level network components (network interface cards, cables and switches), that are transparent at the network interface API level.

The servers can use different port numbers for different purposes.

When only one logical network access is used, HAC (with ERE) detects network breaks that affect both the HSB transaction replication, and the database client applications. Since HAC monitors the health of the HSB link only, failures in database client communications will not be detected by HAC, if separate interfaces are used.

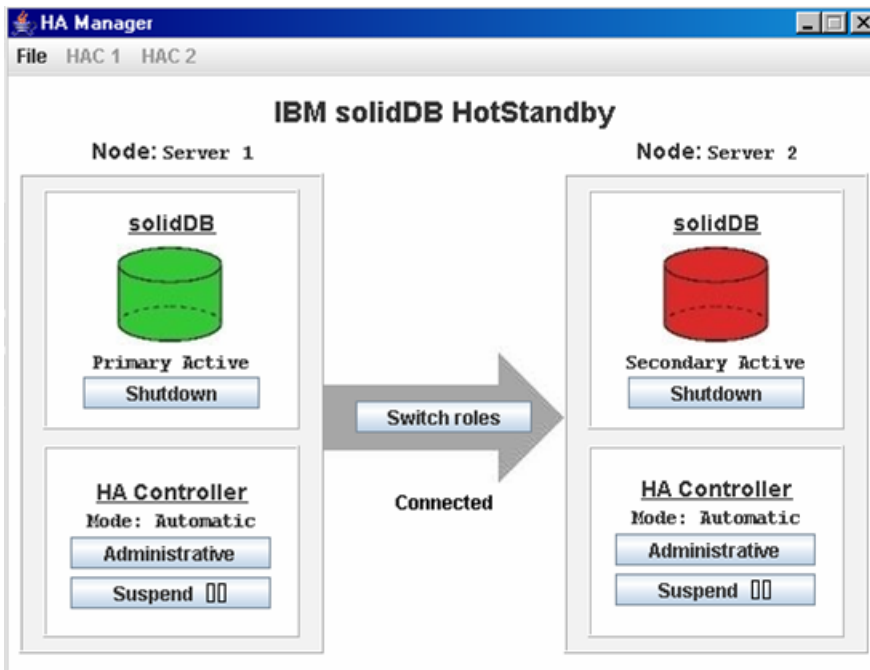
However, regardless of the underlying network access implementation, the ERE feature of HAC can be used if the network access is regarded unreliable for whatever reason.

### 2.4.5 High Availability Manager

High Availability Manager is a Java-based graphical interface tool for displaying the state of the HotStandby servers and the state of the HACs. It also provides basic functionality for managing the HAC, for example, by switching the roles of HotStandby servers and to suspend and resume HAC.

High Availability Manager is configured through the `HAManager.ini` configuration file.

See below for a screenshot of the High Availability Manager:

**Figure 2.9. High Availability Manager**

You can perform the following actions by using the GUI of the High Availability Manager:

- Perform a switchover between the HotStandby servers
- Switch the HAC mode between AUTOMATIC and ADMINISTRATIVE
  - in the ADMINISTRATIVE mode, HAC only monitors the HSB cluster, and the user can perform administrative tasks on HSB servers
  - in the AUTOMATIC mode, HAC acts as a watchdog, handles failures, and maintains the availability of the database service.
- Shut down a HotStandby database server process
- Start up a HotStandby database server process
- Suspend a High Availability Controller
- Resume a High Availability Controller



### Note

Shutting down and starting up the database server process is only possible if the following configuration parameter is set in the `solidhac.ini` configuration file:

```
HAController.EnableDBProcessControl=Yes
```

## 2.4.6 HAC Logging

HAC writes log records to the `hacmsg.out` file located in the HAC working directory.

The log contains information on the following:

- Warnings
- Fatal errors
- Configuration -related information
- Initialization-related information
- All input events
- HotStandby state changes
- User commands, which cause changes in the system
- HAC state changes
- HAC mode changes (AUTOMATIC/ADMINISTRATIVE)
- Events, which cause state changes in the system

HAC log file has a maximum size of 64 megabytes. When the size limit exceeds, `hacmsg.out` is renamed to as `hacmsg.bak`, and a new `hacmsg.out` is created. These files contain at most 128 megabytes of the most recent logs.

---

# Chapter 3. Getting Started with HotStandby

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

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

Instructions for setting up the High Availability Controller (HAC) are in Chapter 4, *Administering and Configuring HotStandby*.

## 3.1 Before You Begin

Before you set up HotStandby, please note the following information:

1. Read Section 4.1, “What You Should Know”. This section contains important information about using the IBM solidDB HotStandby component.
2. To learn about HotStandby features, run the demonstration contained in the IBM solidDB package. For details, read the section below.
3. Refer to Section 3.3, “HotStandby Quick Start Procedure” for step-by-step instructions on setting up and configuring HotStandby.
4. Get acquainted with the IBM solidDB command line SQL editor `solsql`.

## 3.2 HotStandby Demonstration

The IBM solidDB software package that you installed includes all the files that you need to run demonstrations of the HotStandby component using HAC or the sample `watchdog` implementation. The demonstrations are simplified, but will increase your understanding of how to use the HotStandby component.

The HAC demonstration can be found in `samples\hac`. Detailed information about the HSB/HAC demonstration can be found in file:

`samples\hac\readme.txt`

For instructions on setting up a HAC system, refer to Section 3.5, “HSB with HA Controller Quick Start Procedure”.

Detailed instructions for the Watchdog demonstration are in the file:

- `samples\hsb\readme.txt`

## 3.3 HotStandby Quick Start Procedure

This section describes a quick start procedure for HotStandby. Following the procedure, you will reach a state where your HSB system is ready to serve applications. This method does not assume any other IBM solidDB components than HSB servers. For example HAC is not needed. A similar step-by-step procedure for HAC is described in Section 3.4, “Starting and Stopping HA Controller”. There is also a sample watchdog application included in the IBM solidDB HotStandby package. To use the sample watchdog, you need to provide configuration settings for it. For details, read Appendix F, *Watchdog Sample*.

To setup and run an HSB server pair, you need two networked computers. To set up your HotStandby servers (without any other IBM solidDB components), follow the procedure below.

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=Yes`

If you omit the `HSBEnabled` parameter or if it has a "no" value in a server that you intend for HotStandby, the server will be a non-HotStandby server when it is started.

- `Connect=connect string for the opposite HSB server`

If you omit the `Connect` parameter, the server will start as a HotStandby server, but you will have to provide the connection string by using an ADMIN COMMAND before the servers can connect.

Transaction logging of the HotStandby servers for local recovery purposes can 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 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 parameter in your `solid.ini` configuration file:

```
[Logging]
```



LogEnabled=no

In addition to the parameters above, there are other parameters available. For more details about these parameters, read Section 4.7.1, “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 4.6.2, “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 4.7, “Configuring HotStandby-Specific Parameters”.

2. Start both HSB servers the way you would start any IBM solidDB server. Servers will read the HotStandby configuration information from their own `solid.ini` file. The (HSB) state of both servers after start up is SECONDARY ALONE.
3. Choose the server that will become the Primary, and switch the state of the chosen server to 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'**

4. Connect the Primary to the Secondary by issuing the following command in either server:

**ADMIN COMMAND 'hsb connect'**

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

**ADMIN COMMAND 'hsb state'**

The Primary server should respond that its state is "PRIMARY ACTIVE". If, however, the state of the Primary is something else than expected, PRIMARY ALONE, for example, the status of **hsb connect** can be checked by entering the command:

**ADMIN COMMAND 'hsb status connect'**

If the result is **ACTIVE**, then connect process is still active. If, however, the result is **BROKEN**, then the databases of the servers must be synchronized before connecting them.

5. Synchronizing the databases can be done by executing the following command in the Primary:

**ADMIN COMMAND 'hsb netcopy'**

The status of database copy process can be checked by entering the command:

**ADMIN COMMAND 'hsb status copy'**

6. As soon as the result is **SUCCESS**, the databases are in sync, and the servers can be connected. Re-execute the following command in either server:

**ADMIN COMMAND 'hsb connect'**

Verify the success as instructed in step four.

7. Start using applications.

## 3.4 Starting and Stopping HA Controller

Before you can start HAC, you must have a properly filled-in `solidhac.ini` configuration file in the HAC working directory. See Section 3.5, “HSB with HA Controller Quick Start Procedure” for a short description of configuring HAC. A more complete description about configuring HAC can be found in Section 4.9, “Configuring HA Controller and HA Manager” and Section 4.10.2, “The `solidhac.ini` Configuration File”. For a full list of parameters and detailed parameter descriptions, see Section A.8, “High Availability Controller Configuration Parameters”.



### Note

Depending on the platform you are using, the HAC binary is named either `solidhac`, or `solidhac.exe`. We use `solidhac.exe` for clarity. Similarly, the name of the sample IBM solidDB start script is either `start_solid.sh`, or `start_solid.bat`. In the following, we use `start_solid.bat`.

The HA Controller command syntax is:

```
solidhac.exe [-c working directory | -?]
```

The `?` argument, or any other argument except `c` prints the usage message.

### Example 3.1. Starting HAC

Assume that `solidhac.exe` is located in `c:\solid\hac`, which is also the current directory, and we use `c:\solid\run\server1` as the working directory for HAC. HA Controller is started by using the following command:

```
solidhac.exe -c c:\solid\run\server1
```

or

```
solidhac.exe -c ..\run\server1
```

When HAC starts, it starts to listen to the port specified in the `solidhac.ini` configuration file. That port is used for transporting commands between HAC, and the HAManager (Java GUI). Additionally, HAC accepts the termination command sent to that port by using an ADMIN COMMAND.

You can stop (terminate) HAC by executing the command below. To give the command, use an SQL tool such as `solsql`, or the ODBC interface, and send the command to HAC process. Use the port mentioned in the `listen` parameter in the `solidhac.ini` configuration file. The command is:

**ADMIN COMMAND 'hacontroller shutdown'**

For further information about managing HAC, see Section 2.4.5, “High Availability Manager”.

## 3.5 HSB with HA Controller Quick Start Procedure

This section describes a quick start procedure for HotStandby with HAC. The procedure is quite similar to that in Section 3.3, “HotStandby Quick Start Procedure”. However, instead of just setting up two HSB servers, this procedure guides you to setup a highly available HSB system, where availability is guaranteed by the HA Controller. Following the procedure below, you will get a failure tolerant HSB system ready to serve applications.

To setup and run a Highly Available HSB system, you need two networked computers. To set up your HotStandby servers, and HA Controller (one instance per HSB server), follow the procedure below.

1. Configure the HSB servers similarly regardless of whether you are using HAC. For further information, see Section 3.3, “HotStandby Quick Start Procedure”.
2. Configure both HACs. HAC reads its configuration from the `solidhac.ini` file located in the working directory. The lists below contains the mandatory configuration parameters.

In the `[HAController]` section:

- *Listen*=<listen address, 'tcp' and chosen port #>
- *Username*
- *Password*
- *DBUsername*
- *DBPassword*

In the [LocalDB] section:

- *Connect*=<connect address, protocol, ip/hostname, port #>
- *StartScript* (mandatory if *EnableDBProcessControl*=Yes, which is the default value)

In the [RemoteDB] section:

- *Connect*=<connect address, protocol, ip/hostname, port #>

For a full list of parameters and detailed parameter descriptions, see Section A.8, “High Availability Controller Configuration Parameters”.

For further information about configuring HA Controller, see Section 4.9, “Configuring HA Controller and HA Manager” and Section 4.10.2, “The solidhac.ini Configuration File”.

#### 3. Start HA Controller in both nodes.

HAC automatically finds out, which server should become the new Primary according to their previous roles, and log positions. HAC uses the mechanism described in Section 6.7, “Choosing Which Server to Make Primary”. In some special situations, for example when started with empty databases, both servers are equally good candidates for the new Primary. In these situations HAC chooses its local server if in the *PreferredPrimary* parameter in the [LocalDB] section in *solidhac.ini* is set to 'Yes'.

#### 4. After the Second step, there should be one server in the PRIMARY ACTIVE state and the other in the SECONDARY ACTIVE state. You can switch the roles, if necessary, by pressing the Switch roles button in HAManager, or executing the following command on the Secondary server:

**ADMIN COMMAND 'hsb switch primary'**

or on the Primary server:

**ADMIN COMMAND 'hsb switch secondary'**

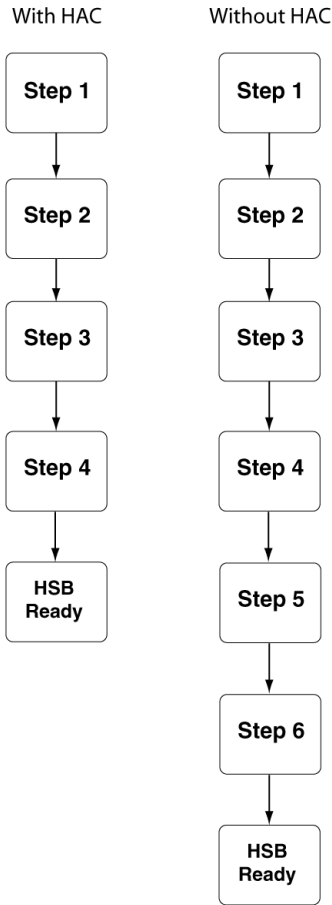
After this point, HAC switches the roles of the servers when necessary. That is, it keeps at least one server executing read and write transactions.

5. Start using applications.

## 3.6 Summary of Start-up Sequences

The following figure and table present side-by-side sequences for an installation with HAC and for an installation without HAC. In the figure, the installation sequence for HAC is on the left and the installation sequence without HAC is on the right. The table after the figure explains each numbered step in the figure for both installation types.

**Figure 3.1. Summary of Start-up Sequences**



**Table 3.1. Installation Sequence Steps**

| Installation with HAC                      | Installation without HAC   |
|--|--|
| Step 1. Configure HSB servers              |  |
| Step 2. Configure HA Controller            | Step 2. Start HSB servers in both nodes                          |
| Step 3. Start HA Controllers in both nodes | Step 3. Switch the state of the database server to PRIMARY ALONE |
| Step 4. Switch HSB roles, if necessary     |  |

### 3.6 Summary of Start-up Sequences

---

| <b>Installation with HAC</b>                    | <b>Installation without HAC</b>  |
|---|--|
|   | Step 4. Connect Primary to the Secondary<br><br>Step 5. If connect failed, start db copy from Primary to Secondary<br><br>Step 6. After netcopy, reconnect |
| HSB is ready. You can Start using applications. |  |

---



---

# Chapter 4. Administering and Configuring HotStandby

This chapter describes how to maintain your HotStandby installation, including HSB servers, HA Controller instances, and HA Manager. Furthermore, this chapter describes the parameter settings for implementing and maintaining HotStandby functionality. This description supplements the information in the "Configuring IBM solidDB" chapter in *IBM solidDB Administration Guide*.

Parameters are grouped according to section categories in the `solid.ini` configuration file. When you are using HotStandby, you are required to configure the `[HotStandby]` section of the `solid.ini` configuration file.

The High Availability Controller (HAC) configuration can be found in the `solidhac.ini` configuration file. The parameters in the `solidhac.ini` configuration file are also group according to different section categories.

The High Availability Manager configuration can be found in the `HAManager.ini` configuration file. The parameters in the `HAManager.ini` are mainly used to identify the HA Controller instances so that the High Availability Manager has access to them.

You can change configuration parameters in the following ways:

- Manually editing the configuration files `solid.ini`, `solidhac.ini` and `HAManager.ini`. Note that the server reads the configuration files during startup only, and therefore any changes to any configuration file will not take effect until the next time that the corresponding program is started.
- If you want to change the setting(s) of a running IBM solidDB server, you may use ADMIN COMMANDs in IBM solidDB SQL Editor `solsql` to do that. Refer to Appendix A, *Configuration Parameters*, for more details on parameter manipulation. The ADMIN COMMAND syntax is presented below:

```
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**

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*.

## 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 IBM solidDB Configuration File

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

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.

Note that each server's *Connect* string must match with the other server's *Listen* string. Read Section 4.7, "Configuring HotStandby-Specific Parameters", for details on the *Connect* parameter and other parameter settings. Refer to Chapter 3, *Getting Started with HotStandby*, if you are setting up HotStandby for the first time.

For examples on the *solid.ini* file on the Primary and Secondary server, see Section 4.10.1, "The *solid.ini* Configuration File".

### 4.1.2 HotStandby and Access Rights

Administrators require no special access rights to run HotStandby, normal access rights suffice. For administration purposes, *SYS\_ADMIN\_ROLE* or *SYS\_CONSOLE\_ROLE* are required to execute the HotStandby administrative commands. These commands are executed with the IBM solidDB SQL command:

```
ADMIN COMMAND 'hotstandby command_string';
```

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

### ADMIN COMMAND 'hotstandby status connect';

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

**hotstandby status connect**

## 4.1.3 IBM solidDB Tools and HotStandby

All tools available for performing administration with IBM solidDB apply also to HotStandby. You can issue HotStandby-specific administrative commands in IBM solidDB SQL Editor, and IBM solidDB Remote Control, *solsql* and *solcon*, respectively. In addition, IBM solidDB Speedloader (SOLLOAD), IBM solidDB Export (SOLEXP), and IBM solidDB Data Dictionary (SOLDD) can be used with HotStandby. For a description of these tools, read "Using IBM solidDB Data Management Tools" in *IBM solidDB Administration Guide*.

The High Availability Manager can be used to monitor IBM solidDB HotStandby states, and to control HSB servers and HA Controllers. HA Manager can be used with the HA Controller only. For a description of the High Availability Manager, see Section 2.4.5, "High Availability Manager".

## 4.1.4 Database Migration (disk-based servers only)

IBM solidDB 4.x databases can be converted to the latest IBM 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 IBM solidDB databases are no longer in use by HotStandby, they remain compatible with IBM solidDB.

## 4.1.5 Interoperability

The servers in a HSB system 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 and Warnings with HotStandby

The following chapters introduce a limitation for using in-memory tables with HotStandby. The warnings deal with avoiding dual primaries and running out of space for transaction logs.

### 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`, or `SERIALIZABLE` (in-memory tables do not support `SERIALIZABLE` at all on either the Primary or the Secondary.)

If load balancing is used, the isolation level is `READ COMMITTED` by default. For more information about load balancing, and Transparent Connectivity, see Section 5.2, “Using the Transparent Connectivity”.

### 4.2.2 Network Partitions and Dual Primaries

In some circumstances, it is possible to have both servers acting in the `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.

The dual primaries problem is most likely to be caused by a “network partition” — 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 primaries scenario can be avoided by using a single-instance watchdog in a node that is external to the HSB system. By using such a watchdog, it is easy to decide which server should be set to Primary, and to ensure that clients see only one Primary.

Although HAC is composed of two instances running in the same nodes with HSB servers, the dual primary situation is virtually impossible, when HAC is used with ERE.

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. Using ERE is highly recommended.

The chances of dual primaries increase 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, but it also means that the system no longer has any independent observer (HAC, watchdog or human) to prevent dual primaries. If you have any doubts on your network reliability, keep the *AutoPrimaryAlone* parameter in its factory value, that is, No.

## 4.2.3 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 does not run out of disk space for transaction logs.

In a non-HotStandby server, if you checkpoint frequently, then the transaction log does not grow very large because after each checkpoint the server deletes the "old" transaction log(s) — that is, the logs with the data changes that occurred before the checkpoint. (For more information about checkpointing, see *IBM 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.

If HAC is used, it identifies the situation described above, and leads the servers to the ACTIVE state by automatically performing the necessary actions.

## 4.3 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            | <p>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:</p> <ul style="list-style-type: none"> <li>• Switching server states</li> <li>• Shutting off HotStandby operations</li> <li>• Synchronizing Primary and Secondary servers</li> <li>• Connecting HotStandby servers</li> </ul> | Section 4.4, “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 6.3.5, “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 6.5, “Checking HotStandby Status”                                   |
| Verifying HotStandby Server States                              | Describes how to check the state (Primary, Secondary, or standalone) of a HotStandby server.  | Section 6.6, “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 6.8, “Changing a HotStandby Server to a Non-HotStandby Server” |

## 4.4 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:

- a. 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.
- b. Synchronize the servers to be sure the Primary and Secondary databases are identical.
- c. Connect the Primary server to the Secondary server if the communication link is broken for some reason.

The same steps can be taken with HAC and HA Manager as follows:

- a. Press the Switch button in HA Manager. If the server needs to be shut down, press the Shutdown button in HA Manager.
- b. Set HAC instances to the ADMINISTRATIVE mode by pressing the Administrative buttons in HA Manager.
- c. Set HAC instances to AUTOMATIC mode by pressing the Automatic buttons in HA Manager.

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”.

### **Important**

If HAC is used, either use HA Manager to perform administrative steps, or set HAC instances to ADMINISTRATIVE mode before starting the administration.

## 4.5 Special Configurations: Lower Cost vs. Higher Safety

IBM solidDB'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 one or more additional spare servers so that when a Primary fails 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, IBM solidDB 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 one 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 one 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 IBM solidDB features that help you implement these.

### 4.5.1 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 fails, 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 IBM solidDB's advanced replication technology, and you can copy or re-create enough of the data by reading from the advanced replication "master" or "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.

### **4.5.2 Increasing Reliability: $2N + 1$ Spare and $2N + M$ Spare Scenarios**

Normal IBM solidDB 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.

## 4.5.3 How IBM solidDB 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.

IBM solidDB has two features to support these needs:

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

These are explained in more detail below:

1. Although IBM solidDB 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 6.3.6, “Creating a New Database for the Secondary Server”, and also see the explanation of command-line options in *IBM solidDB Administration Guide*.

2. IBM solidDB'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';
```

## 4.5.4 Using HAC with Spares

HAC has limited support on spare scenarios. It can be used in a spare, but before HAC in the Primary node is able to start monitoring the HSB state of the spare server, the connect information of the Primary HAC, *RemoteDB.Connect*, must be updated. That requires updating the `solidhac.ini` file and restarting the HAC in the Primary node.

Similarly, the HAC in the spare node needs the connect information of the Primary server. If that is not known beforehand, the information must be added in the `solidhac.ini` file. After you have added the information to `solidhac.ini`, you must restart HAC.

## 4.6 Configuring IBM 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.

## 4.6.1 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 IBM 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 4.1. 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 *IBM solidDB Administration Guide*.

## 4.6.2 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 IBM solidDB SQL Editor (*solsql*), IBM solidDB Remote Control teletype (*solcon*), and HA Manager.

## 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]` section 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.

## 4.6.3 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 *IBM solidDB Administration Guide*.

## 4.7 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.

### 4.7.1 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 4.2. Partial `solid.ini` Files

Node 1

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

Node 2

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

```
;The server connects to the opposite 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.

## 4.7.2 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 "ping" messages the server sends to indicate that it is healthy.
- 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 "I'm alive" message sent by one database server to another. (Some networking software also has a "ping" operation, but the IBM solidDB *PingTimeout* configuration parameter in the [*HotStandby*] section applies only to IBM solidDB server pings, not general network pings.) In other words, this refers to a passive heartbeat system. When this parameter is set, both the Primary and Secondary HotStandby servers send "ping" messages to each other at regular intervals. See also Section 2.1.3, "Heartbeat".

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 send ping messages to 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 IBM solidDB server is set up to respond quickly to missing ping messages. 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 fails, 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 message 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
```

## 4.7.3 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.

## 4.7.4 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 HAC, 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.

## 4.8 Performance Tuning

### 4.8.1 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”.

## 4.8.2 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 6.3.5, “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 *BackupBlockSize* 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.

## 4.9 Configuring HA Controller and HA Manager

High Availability Controller (HAC) is deployed on each of the HotStandby server nodes. The HAC configuration file `solidhac.ini` must be located in the HAC working directory. The mandatory parameters are:

Parameters in the `solidhac.ini` configuration file are grouped under the following sections:

- `HACController`
- `LocalDB`
- `RemoteDB`
- `ERE`

The parameters in different sections of `solidhac.ini` are described in the following chapters. All the configuration parameters are also shown in the `solidhac.ini` example file in Section 4.10.2, “The `solidhac.ini` Configuration File”.

### 4.9.1 Section

The parameters in the `HACController` section are explained below:

- `Listen`

The value of the `Listen` parameter specifies the protocol, and the port the HAC uses for client communications. HAC starts listen to that port during startup. If listening cannot be started, for example, if the port is used by another process, the information is written to log file (`hacmsg.out`) followed by the termination of HAC. The only supported protocol is TCP/IP ('tcp'). The HAC listening port is used by HA Manager to communicate with HAC. Also, the HAC shutdown command is sent to this port.

*Listen* is a mandatory parameter.

- *StartInAutomaticMode*

Once the HAC is running, it can be in one of the following modes: AUTOMATIC, or ADMINISTRATIVE. In the AUTOMATIC mode, HAC automatically tries to maximize the availability by changing the HSB states of the server, and restarting the server processes when necessary. In contrast, in the ADMINISTRATIVE mode, HAC only monitors the health of the servers. *StartInAutomaticMode* specifies the initial mode for HAC.

The default value for *StartInAutomaticMode* is 'Yes'.

- *EnableDBProcessControl*

Setting *EnableDBProcessControl=Yes* allows HAC to manage local server process by automatically starting the server, and by providing the user with commands to shutdown and restart the database process. Setting *EnableDBProcessControl=Yes* makes the *StartScript* parameter mandatory.

The default value is 'No'.

- *EnableAutoNetcopy*

Setting *EnableAutoNetcopy=Yes* allows HAC to initiate netcopy when a HSB link cannot be established with the **hsb connect** command.

The default value for *EnableAutoNetcopy* is 'Yes'.

- *RequiredConnectFailures*

When a server state is unknown, or HAC needs for some other reason to ensure the state of the server, the non-blocking **SQLConnect (check)** command is used. If the execution of non-blocking **SQLConnects** in such a case fails, it is repeated multiple times before the server in question is considered as non-responsive. The maximum number of consecutive connect attempts equals to the value of *RequiredConnectFailures*.

The default value for *RequiredConnectFailures* is 2.

- *CheckTimeout*

Timeout in milliseconds for consecutive non-blocking **SQLConnect** commands. A too short value can cause 'false positives'. That is, a server seems to be failed, although it is running, but was not able to respond within the timeout period.

The default value for *CheckTimeout* is 150 (milliseconds).

- *CheckInterval*

The interval between consecutive non-blocking **SQLConnect** commands. This value does not affect the failover time. Checking (polling) takes place typically after failure, or during system startup.

The default value for *CheckInterval* is 1000 (milliseconds).

- *Username*

The HAC username.

*Username* is a mandatory parameter.

- *Password*

The password of the user identified by using the *Username*.

*Password* is a mandatory parameter.

- *DBUsername*

The username for HAC to be used to connect to local HSB server. The database user should have either `SYS_ADMIN_ROLE`, or `SYS_CONSOLE_ROLE`.

*DBUsername* is a mandatory parameter.

- *DBPassword*

The password of the user identified by using the *DBUsername*.

*DBPassword* is a mandatory parameter.

## 4.9.2 LocalDB Section

The parameters in the LocalDB section are explained below:

- *Connect*

Connect information of the local database server. HAC uses this information when it connects to the local server.

*Connect* is a mandatory parameter.

- *StartScript*

*StartScript* is mandatory if HAC is configured to control the database process (*EnableDBProcessControl=Yes*). The value is not effective if *EnableDBProcessControl=No*.

- *PreferredPrimary*

Setting *PreferredPrimary=Yes* moves the local HSB server to the Primary in the case where either of the servers could start as Primary. If both servers have the same value in *PreferredPrimary*, the 'first' server becomes the new Primary.

The default value for *PreferredPrimary* is 'No'.

## 4.9.3 RemoteDB Section

The parameters in the RemoteDB section are explained below:

- *Connect*

Connect information of the remote database server. HAC uses this information when it connects to the server in the remote node.

*Connect* is a mandatory parameter.

## 4.9.4 ERE Section

The parameters in the ERE section are explained below:

- *EREIP*

IP address of an ERE.

- *RequiredPingFailures*

The number of consecutive ping calls that must fail before HAC concludes that the server is disconnected from ERE.

The default value for *RequiredPingFailures* is 3.

## 4.10 Configuration File Examples

The following sections give examples of different configuration files related to IBM solidDB.



## 4.10.1 The solid.ini Configuration File

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 188.177.166.12 1321
AutoPrimaryAlone=No
[Logging]
LogEnabled=yes
```

Below is a sample excerpt of the IBM solidDB 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 188.177.166.11 1320
AutoPrimaryAlone=No
[Logging]
LogEnabled=yes
```

## 4.10.2 The solidhac.ini Configuration File

Below is a sample excerpt of the High Availability Controller configuration file (`solidhac.ini`):

```
;=====
; NOTE : Copy this file as solidhac.ini
```

## 4.10.2 The solidhac.ini Configuration File

---

```
;          to solidhac working directory
;
; solidDB High Availability Controller inifile
;=====

[HAController]
;** HAC connect info
;** HAC clients, HA Manager, for example, use this information.
;** Mandatory
;** Listen=tcp 3135
Listen=

;** Setting StartInAutomaticMode=Yes starts HAC in AUTOMATIC mode.
;** In AUTOMATIC mode, solidhac automatically tries
;** to maximize the availability by changing the HSB states of the
;** server, and restarting the server processes when necessary.
;** In contrast, it can be in ADMINISTRATIVE mode
;** in which HAC only monitors the health of the servers.
;**
;** This is dynamically changeable parameter.
;** Optional
;** Values : Yes/No, default = Yes
StartInAutomaticMode=

;** Setting EnabledDBProcessControl=Yes allows solidhac
;** manage local db process by automatically starting
;** the db, and by providing the user with commands to
;** shutdown and restart db process.
;**
;** Optional
;** Effective only when HAC is in AUTOMATIC mode.
;** Values : Yes/No, default = No
EnabledDBProcessControl=

;** Setting EnableAutoNetcopy=Yes allows solidhac to initiate
;** netcopy when HSB link cannot be established with 'hsb connect'.
;**
;** Optional
;** Effective only when HAC is in AUTOMATIC mode.
;** Values : Yes/No, default = Yes
EnableAutoNetcopy=
```

```
;** When server state is unknown, or HAC needs, for some other reason, to
;** ensure the state of server, non-blocking SQLConnect command is used.
;** If the execution of non-blocking SQLConnect in such a case fails,
;** it is repeated multiple (RequiredConnectFailures) times before
;** the server in question is considered as non-responsive.
;**
;** Optional
;** Values : 1..n, default=2
RequiredConnectFailures=

;** Timeout in milliseconds for non-blocking SQLConnect commands.
;** Too short interval can cause 'false positives', server seems
;** to be failed because it wasn't able to respond within the timeout period.
;**
;** Optional
;** Values : 1..n, default=150 (milliseconds)
CheckTimeout=

;** Interval between consecutive non-blocking SQLConnect commands.
;** The value doesn't affect on failover time. Checking (polling)
;** takes place typically after failure, or during system startup.
;**
;** Optional
;** default = 1000 (milliseconds)
CheckInterval=

;** HAC user identification
;** Mandatory
Username=
Password=

;** HSB server user identification
;** Mandatory
DBUsername=
DBPassword=

[LocalDB]
;** soliddb connect info
;** Mandatory
;** Connect=tcp 2125
Connect=
```

## 4.10.2 The solidhac.ini Configuration File

---

```
;** The name of the script, which is used to initiate the db process.
;**
;** Optional, except if HAC controls db process (EnableDBProcessControl=Yes).
;** Value is not effective if EnableHACActions=No or EnableDBProcessControl=No
;** StartScript=/home/solid/start_solid.sh
StartScript=

;** Setting PreferredPrimary=Yes moves local HSB server to as Primary
;** in the case where either of the servers could start as Primary.
;** If both servers have PreferredPrimary=No, or no value, first
;** (new Primary) server wins.
;**
;** Optional
;** Value is not effective if EnableHACActions=No.
;** Values : Yes/No, default no
PreferredPrimary=

[RemoteDB]
;** soliddb connect info
;** Mandatory
;** Connect=tcp 192.168.3.123 2125
Connect=

[ERE]
;** IP address of an ERE
;** Optional
;** Connect=192.168.3.1
EREIP=

;** The number of consecutive ping calls that must
;** fail before HAC concludes that the server is
;** disconnected from ERE.
;**
;** Optional
;** Values : 1..n, default=3
```

```
;** RequiredPingFailures=10
RequiredPingFailures=
```

### 4.10.3 The HAManager.ini Configuration File

Below is a sample excerpt of the High Availability Manager configuration file (HAManager.ini):

```
=====
; IBM solidDB for MySQL High-Availability Manager
; Configuration file HAManager.ini
; V. 0.3
; 2008-21-01
=====
;** HA Controller connect info, e.g.
;Server1_name = Server 1
;Server1_host = node1.acme.com
;Server1_port = 2220
;Server2_name = Server 2
;Server2_host = node2.acme.com
;Server2_port = 2220

; All the following lines are mandatory.
Window_title = HA Manager
Header_text  = SolidDB HA Manager

; Display names, host addresses and port numbers
; of the SolidHAC (HA Controllers) instances
;Server 1 HA Controller
;-----
Server1_name = Server 1
Server1_host = localhost
Server1_port = 1234
Server1_user = foo
Server1_pass = bar
;
;Server 2 HA Controller
;-----
Server2_name = Server 2
Server2_host = 192.168.0.1
Server2_port = 1234
```

```
Server2_user = foo  
Server2_pass = bar
```

---

# 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 IBM solidDB 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

IBM solidDB 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 IBM solidDB 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.

## Executing Procedures under Load Balancing

All SQL stored procedures are executed in the Primary unless they are specified as read-only procedures by way of the SQL standard clause *SQL Data Access Indication*, in the procedure declaration.

```
<SQL-data-access-indication> ::=  
    NO SQL |  
    READS SQL DATA |  
    CONTAINS SQL |  
    MODIFIES SQL DATA
```

To avoid unnecessary handovers of read-only procedures and functions, one of the following values can be declared:

- NO SQL
- READS SQL DATA
- CONTAINS SQL

Only MODIFIES SQL DATA (which is the default) inflicts transaction handover.

The clause comes between the (optional) RETURNS clause and the procedure body. For example:

```
"CREATE PROCEDURE PHONEBOOK_SEARCH  
(IN FIRST_NAME VARCHAR, LAST_NAME VARCHAR)  
RETURNS (PHONE_NR NUMERIC, CITY VARCHAR)  
READS SQL DATA  
BEGIN  
-- procedure_body  
END";
```

---

## 5.2.3 Syntax of the TC Info

When using IBM solidDB 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 IBM solidDB TC Info is the following:

```
<solidDB 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 IBM solidDB 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_ONLY
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**

| <b>PREFERRED_ACCESS:</b> | <b>TF_LEVEL: Not specified or NONE</b>   | <b>TF_LEVEL: CONNECTION</b>   | <b>CONNECTION</b>   | <b>TF_LEVEL: SESSION</b>  |
|--------------------------|--|---|---|---|
| Not specified            | <ul style="list-style-type: none"> <li>No failover or switchover support</li> <li>No load balancing</li> </ul> (Basic Connectivity)  | <ul style="list-style-type: none"> <li>Transparent Failover (session state not preserved)</li> <li>Transparent Switchover</li> <li>Workload in Primary only</li> <li>No load balancing</li> </ul> | <ul style="list-style-type: none"> <li>Transparent Failover (session state preserved)</li> <li>Transparent Switchover</li> <li>Workload in Primary only</li> <li>No load balancing</li> </ul> | <ul style="list-style-type: none"> <li>Transparent Failover (session state preserved)</li> <li>Transparent Switchover</li> <li>Workload in Primary only</li> <li>No load balancing</li> </ul> |
| WRITE_MOSTLY (default)   | <ul style="list-style-type: none"> <li>No transparent failover support</li> <li>Transparent Switchover</li> <li>Workload in Primary only</li> <li>No load balancing</li> </ul> | <ul style="list-style-type: none"> <li>Transparent Failover (session state not preserved)</li> <li>Transparent Switchover</li> <li>Workload in Primary only</li> <li>No load balancing</li> </ul> | <ul style="list-style-type: none"> <li>Transparent Failover (session state preserved)</li> <li>Transparent Switchover</li> <li>Workload in Primary only</li> <li>No load balancing</li> </ul> | <ul style="list-style-type: none"> <li>Transparent Failover (session state preserved)</li> <li>Transparent Switchover</li> <li>Workload in Primary only</li> <li>No load balancing</li> </ul> |

| <b>PREFERRED_ACCESS:</b> | <b>TF_LEVEL: Not specified or NONE</b>   | <b>TF_LEVEL: CONNECTION</b>   | <b>CONNECTION</b> | <b>TF_LEVEL: SESSION</b>  |
|--------------------------|--|---|-------------------|---|
| READ_MOSTLY              | <ul style="list-style-type: none"> <li>No Transparent Failover support</li> <li>Transparent Switchover</li> <li>Workload in Secondary and Primary</li> <li>Load balancing</li> </ul> | <ul style="list-style-type: none"> <li>Transparent Failover (session state not preserved)</li> <li>Transparent Switchover</li> <li>Workload in Secondary and Primary</li> <li>Load balancing</li> </ul> |                   | <ul style="list-style-type: none"> <li>Transparent Failover (session state preserved)</li> <li>Transparent Switchover</li> <li>Workload in Secondary and Primary</li> <li>Load balancing</li> </ul> |

## 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. You can give the value as a mnemonic (such as `NONE`) or as a number (0 for `NONE`). Use primarily the mnemonics. The value must be given as a string. Three levels are available:

- `NONE`. The numerical value for `NONE` is 0.
- `CONNECTION`. The numerical value for `CONNECTION` is 1.
- `SESSION`. The numerical value for `SESSION` is 3.

For more information on the values, see Section 5.2.1, “Failure Transparency in TC”

The preferred access attribute indicates whether the read-only load is distributed or not. The preferred access attribute is enabled with the `solid_preferred_access` connection property. You can give the value



as a mnemonic or as a number. Use primarily the mnemonics. The value must be given as a string. Two levels are available:

1. `WRITE_MOSTLY`. `WRITE_MOSTLY` also sets the connection to the `WRITE MOSTLY` mode. It is not possible to do that by specifying a numeric value. The numeric value for `WRITE_MOSTLY` is 0.
2. `READ_MOSTLY`. The numeric value for `READ_MOSTLY` is 1.

There is also TC/TF specific property `solid_tf1_reconnect_timeout` to specify the connection reconnect timeout in milliseconds. The default value is 10 000 milliseconds (10 seconds).

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.setProperty("solid_tf_level", "CONNECTION");
...
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       | <p>Client unable to establish a connection</p> <p>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 the following reasons:</p> <ul style="list-style-type: none"> <li>• No host listed in the address list was found</li> <li>• A host was found but the login timed out</li> <li>• A host was found but the login was rejected</li> <li>• Hosts found but not in the PRIMARY/STANDALONE state</li> </ul> |
| HY000    | 21307       | <p>Invalid connect info...</p> <p>Description: a syntax error is found in an elementary connect string or in the TC connect info (data source info).</p>   |
| HY000    | 21300       | <p>Protocol ... not supported.</p> <p>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).</p>   |

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

**Table 5.5. Warnings**

| SQLSTATE | Native code | Message text and description   |
|----------|-------------|--|
| 0100     | 25218       | <p>Connected to Standalone or Primary Alone server.</p> <p>An effort has been made to set any non-default value of TF_LEVEL or PREFERRED_ACCESS, and there is only one server available. On this case, neither failure transparency nor load balancing is available.</p> |

## 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       | <p>Connection switch, some session context may be lost</p> <p>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</p> |

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       | <p>Communication link failure</p> <p>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.</p> |

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    | <p>Prepared statements</p> <ul style="list-style-type: none"> <li>• The prepared states are preserved.</li> </ul> <p>The effects of the following statements are preserved:</p> <ul style="list-style-type: none"> <li>• SET CATALOG</li> <li>• SET SQL INFO</li> <li>• SET SQL SORTARRAYSIZE</li> <li>• SET SQL CONVERTORSTOUNIONS</li> <li>• SET SQL SORTEDGROUPBY</li> <li>• SET SQL { OPTIMIZEROWS   OPTIMISEROWS }</li> <li>• SET SIMPLEOPTIMIZERRULES</li> <li>• SET LOCK TIMEOUT &lt;seconds&gt;</li> <li>• SET OPTIMISTIC LOCK TIMEOUT</li> <li>• LOCK_TIMEOUT</li> <li>• SET IDLE TIMEOUT</li> <li>• SET STATEMENT MAXTIME</li> <li>• SET ISOLATION LEVEL</li> <li>• SET DURABILITY</li> <li>• SET SAFENESS</li> <li>• SET SCHEMA</li> </ul> |

| TF_LEVEL | Preserved state   |
|----------|---|
|          | <ul style="list-style-type: none"> <li>• SET SQL JOINPATHSPAN</li> <li>• SET SYNC USER</li> <li>• SET SYNC MODE</li> </ul> <p>The following standard ODBC attributes are preserved</p> <ul style="list-style-type: none"> <li>• SQL_ATTR_ACCESS_MODE<br/>(SET READ ONLY, SET READ WRITE)</li> <li>• SQL_ATTR_CURRENT_CATALOG<br/>(duplicates SET CATALOG above)</li> <li>• SQL_ATTR_AUTOCOMMIT</li> </ul> |

### 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 server 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 (`solidimpodbc.a.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 fails, 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 component, 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 IBM solidDB 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 (IBM solidDB main memory engine 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 6.6, “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 Advanced Replication Requirements

Any node of a advanced replication system can be made highly available with the IBM solidDB HotStandby component.

When the master and replica databases of a advanced replication 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”.

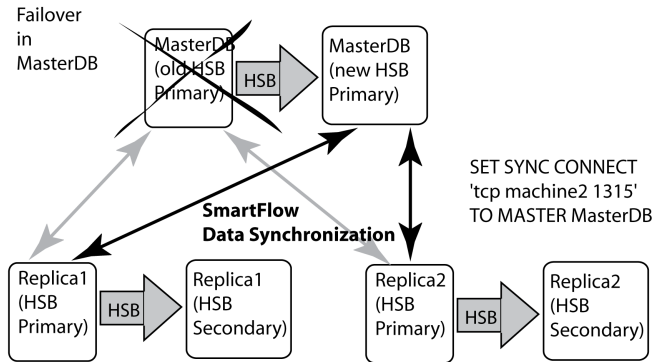
A database server may fail over to its Secondary server at any point of time, including when the database server is synchronizing data with another server using advanced replication. If the failover occurs during synchronization, executing the synchronization message stops and the process must be resumed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

If a server containing a advanced replication master database is made fault tolerant with IBM solidDB HotStandby, 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**



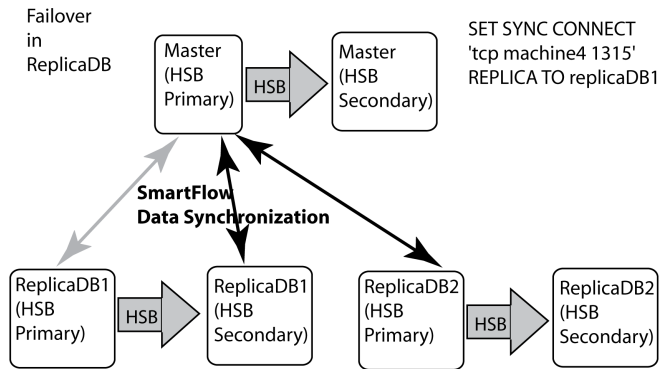
If the server using IBM solidDB HotStandby is a server containing a replica database AND if the master server uses remote procedure calls (CALL procedure AT node\_name) to run procedures at the replica, for example to initiate the synchronization, the master server must be informed about the connect 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. Using HotStandby API Commands

In any software managing high-availability of IBM solidDB HotStandby, the HotStandby API (HSB API) is used to monitor and control the server processes. HA Controller included with the product is an example of such a program. Another example is the Watchdog sample program. The HSB API is provided as a syntax extension to SQL, taking the shape of a non-standard statement in the form:

**ADMIN COMMAND hotstandby *hsb-command options***

or

**ADMIN COMMAND hsb *hsb-command options***

The HSB commands may be issued via any SQL-capable interactive tool (like solsql), or programmatically, through ODBC or JDBC. The commands are documented in Appendix C, *Summary of HotStandby Administrative Commands*.

This chapter can be used to program your own application to manage high availability in IBM solidDB. This may be necessary, for example, to implement an integration to an external cluster management software. The topics included in this chapter are:

- Switching server states
- Shutting off HotStandby operations
- Synchronizing Primary and Secondary servers
- Connecting HotStandby servers
- Checking HotStandby status
- Verifying HotStandby server states
- Choosing which server to make primary
- Changing a HotStandby server to a non-HotStandby server

## 6.1 Switching Server States

The HotStandby component requires that the server state is switched, when necessary, either automatically or manually by a user. In production use, the server state is chosen by using automatic state switching, that is, by performing failovers. This is at the responsibility of an automatic high availability control implemented in IBM solidDB. In IBM solidDB, the automatic high availability control is handled by the High Availability Controller (HAC).

### 6.1.1 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.

### 6.1.2 Performing Switchovers

HAC can reverse the roles of the servers by issuing the following command at the Secondary:

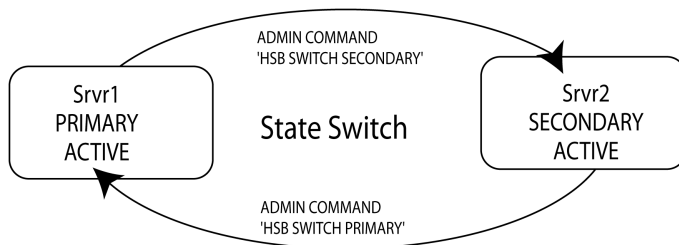
**ADMIN COMMAND 'hotstandby switch primary';**

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 6.1. 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**.)

## Switching Secondary to Primary

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 6.1.3, “Verifying the Switch”.

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 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 the 14525 error occurs, the database states do not change; both servers remain in the same state they were in before the command was issued.

### 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 6.5.1, “Displaying Switch Status Information”.

### 6.1.3 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.

### 6.1.4 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.

### 6.1.5 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 HAC, 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

2. Shutdown the new Secondary
3. The new Primary will automatically switch to the PRIMARY ALONE state

### 6.1.6 Bringing the Secondary Server Back Online

To bring the Secondary server back online, connect the Primary with the Secondary server. For details, read Section 6.4, “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 6.3, “Synchronizing Primary and Secondary Servers”.

## 6.2 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 4.2.3, “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 6.8, “Changing a HotStandby Server to a Non-HotStandby Server”.

## 6.3 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: "catchup" and "full copy".

### 6.3.1 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 IBM 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 STAN-DALONE 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 6.1.6, "Bringing the Secondary Server Back Online".

## **6.3.2 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

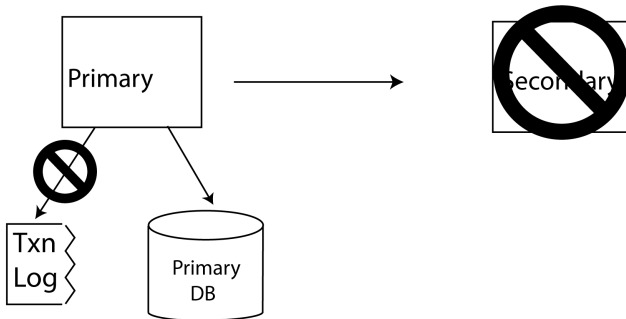
If the Secondary server has old database files, a full copy will write over those old files. If for any reason the files on the Secondary contain data that was not in the Primary (for example, if both servers were operating in PRIMARY ALONE state at the same time), then that data will be lost.

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:

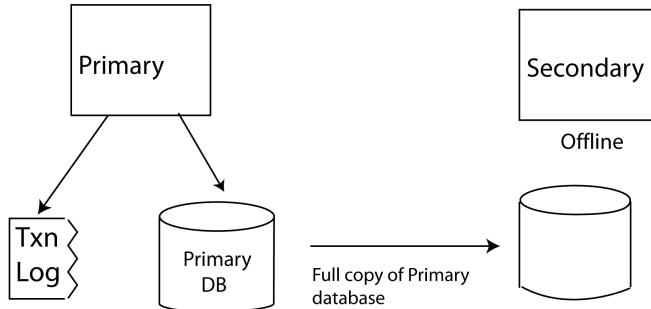
```
ADMIN COMMAND 'hotstandby netcopy';  
ADMIN COMMAND 'hotstandby copy [<directory_name>]';
```

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".

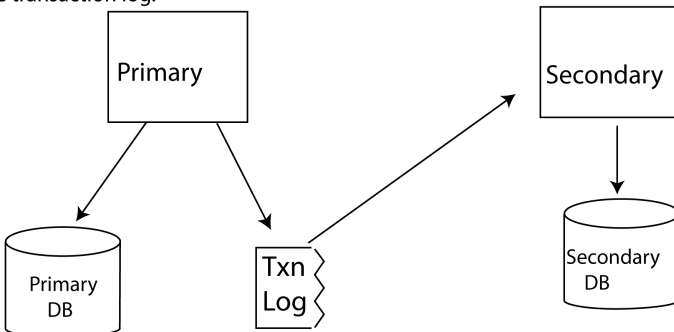
The **copy** and **netcopy** commands are described in Section 6.3.9, “Copying a Database File from Primary Server to a Specified Directory” and Section 6.3.5, “Copying a Primary Database to a Secondary Over the Network”.

**Figure 6.2. 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.

## 6.3.3 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.

## 6.3.4 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.

## 6.3.5 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 4.7.1, “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.

#### **Important**

To execute the command, the Primary server must 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 6.4, “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.

## 6.3.6 Creating a New Database for the Secondary Server

Normally, when you start the IBM solidDB server, it asks you if you want to create a new database (if there is not 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 4.7.1, "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 is not 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.

If HAC is used, the procedure to get the Primary server's database copied to the Secondary is as follows:

1. Ensure that the servers have proper connect parameters. For details on the *Connect* parameter, which defines the connect string, see Section 4.7.1, “Defining Primary and Secondary HotStandby Configuration”.
2. Ensure that the HAC in Primary node has the *PreferredPrimary=Yes* parameter set in *solid-hac.ini*, and that the HAC in the Secondary node does not. For further information of configuring HACs see Section 4.9, “Configuring HA Controller and HA Manager”.
3. Start the HAC instances, or optionally set HACs to the AUTOMATIC mode.



#### 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.

## 6.3.7 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.

## 6.3.8 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

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.

## 6.3.9 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.

#### 6.3.9.1 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 6.4, “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.

## 6.4 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 high availability control 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 6.5.2, “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”.

## 6.5 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 6.1. ADMIN COMMAND 'hotstandby status' Options**

| Option         | Description  |
|----------------|--|
| <b>catchup</b> | 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 <i>hsb status catchup</i> in Appendix C, <i>Summary of HotStandby Administrative Commands</i> . |
| <b>connect</b> | Shows whether the last attempt to connect the servers was successful. For more details, see <i>hsb status connect</i> explanation in Appendix C, <i>Summary of HotStandby Administrative Commands</i> .  |
| <b>copy</b>    | Shows whether the last attempt to <b>copy/netcopy</b> was successful. For more details, see <i>hsb status copy</i> in Appendix C, <i>Summary of HotStandby Administrative Commands</i> .   |
| <b>switch</b>  | Shows whether the last attempt to switch the server into PRIMARY ACTIVE or SECONDARY ACTIVE state was successful. For more details, see <i>hsb status switch</i> in Appendix C, <i>Summary of HotStandby Administrative Commands</i> .   |

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

*Example*

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

## 6.5.1 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.

## 6.5.2 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 6.2. 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. |

| Error Code | Text          | Description  |
|------------|---------------|--|
| 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.  |

## 6.5.3 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 IBM 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';**

## 6.5.4 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.

## 6.6 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';**

For descriptions of possible states the command returns, see Section 2.2, “Description of Server States”.



### 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 HotStandby, the following error message is returned:

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

If HAC, and HA Manager are used, HA Manager displays the HSB states of both servers. The information is queried periodically every second.

## 6.6.1 Server State Combinations

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 6.3. Server States**

| State of the Server | Possible state(s) of the Associated Server                              |
|---------------------|---|
| PRIMARY ACTIVE      | SECONDARY ACTIVE  |
| PRIMARY ALONE       | PRIMARY ALONE *<br>PRIMARY UNCERTAIN<br>SECONDARY ALONE<br>STANDALONE * |
| PRIMARY UNCERTAIN   | PRIMARY ALONE<br>PRIMARY UNCERTAIN<br>SECONDARY ALONE<br>STANDALONE     |
| SECONDARY ACTIVE    | PRIMARY ACTIVE  |
| SECONDARY ALONE     | PRIMARY ALONE<br>PRIMARY UNCERTAIN<br>SECONDARY ALONE<br>STANDALONE     |
| STANDALONE          | PRIMARY ALONE *<br>PRIMARY UNCERTAIN<br>SECONDARY ALONE                 |

| State of the Server | Possible state(s) of the Associated Server |
|---------------------|--|
|                     | 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.

## 6.7 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. If the logpos values of both servers are equal, the *PreferredPrimary* parameter defines whether the local server becomes 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 Administrative 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, the *PreferredPrimary* parameter defines whether The local server becomes the 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.

## 6.8 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 6.2, “Shutting Off HotStandby Operations”.

---

---

# Chapter 7. Behavior of High Availability Controller in Failure Cases

In this chapter, we explain possible failure scenarios. The High Availability Controller (HAC) implicitly handles failure scenarios. However, different failure or initialization scenarios (administrative scenarios, for short) can be handled by a human administrator, or any software program called a "watchdog". A watchdog is a separate program that monitors Primary and Secondary servers, and gives commands to change those servers' states when necessary. We recommend that you use the HAC 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.

The purpose of recovery is to bring the failed component back to operation. Occasionally, further failures happen during recovery. They usually lead to a situation where the system remains in a state of limited availability (only one server is up) awaiting human intervention. Typical recovery-time failures that are not automatically taken care of are:

- The failed database is corrupted to a point that it is impossible to restart it
- There is not enough free disk space to perform a catchup

## 7.1 Primary Database Fails

### Scenario

The primary database (in the PRIMARY ACTIVE state) on node 1 fails.

The secondary database (in the SECONDARY ACTIVE state) on node 2 encounters connection failure to the primary database on node 1.

### Recovery

In the recovery from the Primary database failure the Secondary server replaces the Primary server. The recovery proceeds automatically as follows:

1. Upon the connection failure, the Secondary database on node 2 moves automatically to the SECONDARY ALONE state.

2. The HAC instance on the Secondary database on node 2 concludes that the Primary database on node 1 has failed and sets the Secondary database on node 2 to the PRIMARY ALONE state.
3. In parallel with the above task, the HAC instance on the Primary database on node 1 restarts the Primary database, which enters the SECONDARY ALONE state.
4. The HAC instance on the Secondary database on node 2 initiates the process of connecting the Primary and Secondary database.
5. A catchup is made.

Optionally, the connection process includes a **netcopy** operation from the Secondary database on node 2 to the Primary database on node 1.

## 7.2 Secondary Database Fails

### Scenario

The secondary database (in the SECONDARY ACTIVE state) on node 2 fails.

The Primary database (in the PRIMARY ACTIVE state) on node 1 encounters connection failure to the Secondary database on node 2.

### Recovery

In the recovery from the Secondary database failure the Secondary server is re-started. The recovery proceeds automatically as follows:

1. Upon the connection failure, the Primary database on node 1 moves automatically to the PRIMARY UNCERTAIN state, or if the *AutoPrimaryAlone* parameter is enabled, to the PRIMARY ALONE state.
2. The HAC instance on the Primary database on node 1 concludes that the Secondary database on node 2 has failed.
3. If the Primary database was set to the PRIMARY UNCERTAIN state in step 1, HAC sets it now to the PRIMARY ALONE state.
4. In parallel with the above task, the HAC instance on the Secondary database on node 2 restarts the Secondary database, which enters the SECONDARY ALONE state.

5. The HAC instance on the Primary database on node 1 initiates the process of connecting the Primary and Secondary database.
6. A catchup is made.

Optionally, the connection process includes a **netcopy** operation from the Primary database to the Secondary database.

## 7.3 Primary Node Fails

### Scenario

The primary node (node 1) fails.

The Secondary database (in the SECONDARY ACTIVE state) on node 2 encounters connection failure to the Primary database (in the PRIMARY ACTIVE state) on node 1.

### Recovery

In the recovery from the Primary node failure the Primary server is re-started. The recovery proceeds automatically as follows:

1. Upon the connection failure, the Secondary database moves automatically to the SECONDARY ALONE state.
2. The HAC instance on the Secondary database on node 2 concludes that the Primary database on node 1 has failed.
3. The HAC instance on the Secondary database on node 2 sets the Secondary database to the PRIMARY ALONE state.
4. The Primary node (node 1) is re-started.
5. The HAC instance on the Primary database (node 1) is re-started.
6. The HAC instance on the Primary database (node 1) concludes that the Primary database is not running.
7. The HAC instance on the Primary database (node 1) restarts the Primary database and sets it to the SECONDARY ALONE state.
8. The HAC instance on the Secondary database on node 2 initiates the process of connecting the Primary and Secondary database.

9. A catchup is made.

Optionally, the connection process includes a **netcopy** operation from the Primary database to the Secondary database.

## 7.4 Secondary Node Fails

### Scenario

The secondary node (node 2) fails.

The Primary database (in the PRIMARY ACTIVE state) on node 1 encounters connection failure to the Secondary database (in the SECONDARY ACTIVE state) on node 2.

### Recovery

In the recovery from the Secondary node failure the Secondary server is re-started. The recovery proceeds automatically as follows:

1. Upon the connection failure, the Primary database on node 1 moves automatically to the PRIMARY UNCERTAIN state.
2. The HAC instance on the Primary database on node 1 concludes that the Secondary database on node 2 has failed.
3. The HAC instance on the Primary database on node 1 sets the Primary database to the PRIMARY ALONE state.
4. The Secondary node (node 2) is re-started.
5. The HAC instance on the Secondary database (node 2) is restarted.
6. The HAC instance on the Secondary database (node 2) concludes that the Secondary database is not running.
7. The HAC instance on the Secondary database (node 2) restarts the Secondary database and sets it to the SECONDARY ALONE state.
8. The HAC instance on the Primary database on node 1 initiates the process of connecting the Primary and Secondary database.
9. A catchup is made.

Optionally, the connection process includes a **netcopy** operation from the Primary database to the new Secondary database.

# 7.5 HotStandby Link Fails

## Scenario

The HotStandby link fails.

The Primary database (in the PRIMARY ACTIVE state) on node 1 and the the Secondary database (in the SECONDARY ACTIVE state) on node 2 both encounter connection failure to each other.

## Recovery

In the recovery from the HotStandby link failure the HAC instances Ping the External Reference Entity (ERE) to find out if it is the network or the opposite server that has failed. The recovery proceeds automatically as follows:

1. Upon the connection failure, both databases move automatically to the PRIMARY UNCERTAIN (node 1) and SECONDARY ALONE state (node 2), respectively.
2. The direct connection of both HAC instances to the remote server fail.
3. Both HAC instances Ping the ERE by using the Operating System's Ping utility.
4. The Secondary database on node 2 is re-started.
5. If the Ping fails, the local server is retained or set to the SECONDARY ALONE state.
6. If the Ping succeeds, the successful HAC tries to connect to the remote database server.
7. If the connect effort to the remote database server fails, the HAC concludes that its part of the network connection is operational and sets the local server to the PRIMARY ALONE state.
8. The HAC instance on the Primary database attempts to re-establish the connection to the Secondary database.
9. Once the network becomes operational and the connect succeeds, the Primary database and the Secondary database move automatically to the PRIMARY ACTIVE and SECONDARY ACTIVE state, respectively.





---

# 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 IBM solidDB 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.

IBM solidDB High Availability design allows you to upgrade your IBM 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 if you are using basic connectivity. If you are using the transparent connectivity, you can perform the hot migration transparently to users/applications.

## 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 IBM solidDB 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 IBM 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';<br><br>ADMIN COMMAND 'hsb status connect ping'; | Disconnect Primary from Secondary. |

### 8.3.1 Preparation Steps

|   |           | ADMIN COMMAND  | COMMENTS  |
|---|-----------|--|---|
| 2 | S 2<br>OS | ADMIN COMMAND 'shutdown force';  | Shut down server S2 (Secondary).  |
| 3 | S 1<br>OP | ADMIN COMMAND 'hsb set primary alone';<br><br>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.<br><br>Verify that server S1 (Primary) is in the PRIMARY ALONE state.   |
| 4 | S 2<br>OS | Upgrade server S2 (original Secondary).                                  | You should update the configuration parameters in the <code>solid.ini</code> file, as well as updating the software.  |
| 5 | S 2<br>OS | <b>solid -x migratehsbg2</b>   | Bring up server S2 (original Secondary) using the <b>-x migratehsbg2</b> command-line switch. The server should come up in Secondary Alone state.<br><br>This command-line switch has two effects. It instructs the server to accept and convert the existing database (the same effect as the <b>-x autoconvert</b> parameter). Also, it enables the new Secondary to communicate with the old Primary using the old replication protocol. |
| 6 | S 1<br>OP | ADMIN COMMAND 'hsb state';<br><br>ADMIN COMMAND 'hsb connect';           | Check server S1 (Primary) server to make sure that it is still in PRIMARY ALONE state.<br><br>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<br>OP | Wait for the "catchup" to complete before continuing.                    | If the catchup fails, then you will have to do the following: <ol style="list-style-type: none"> <li>1. Shut down server S2 (the Secondary).</li> <li>2. Do an "hsb copy" from S1 (the Primary) to copy the entire database to server S2.</li> <li>3. Recover the copy with the old version of the server (S2).</li> <li>4. Shut Down S2 (the Secondary).</li> </ol>  |

### 8.3.1 Preparation Steps

|    |           | <b>ADMIN COMMAND</b>   | <b>COMMENTS</b>  |
|----|-----------|--|--|
|    |           |  | <p>5. Go back to the previous step.</p> <p>(Note: In the second step, you must use "hsb copy" rather than "hsb netcopy" because "hsb netcopy" does not work between different server versions.)</p>  |
| 8  | S 1<br>OP | <p>After the servers are connected and caught up, perform:</p> <p><b>ADMIN COMMAND 'shutdown force';</b></p> | <p>After server S2 (Secondary) has caught up, make it the new Primary.</p> <p>Shut down server S1 (the former Primary) to upgrade it.</p> <p><b>CAUTION:</b> The <code>force</code> option will abort any open transactions.</p>   |
| 9  | S 2<br>OS | <p><b>ADMIN COMMAND 'hsb set primary alone';</b></p> <p><b>ADMIN COMMAND 'hsb state';</b></p>                | <p>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.</p> <p>Verify that server S2 is in the PRIMARY ALONE state.</p>   |
| 10 | S 1<br>OP | Upgrade server S1 (your original Primary server).  | You should update the configuration parameters in the <code>solid.ini</code> file, as well as updating the software.   |
| 11 | S 1<br>OP | <b>solid -x backupserver</b>   | Re-start the IBM solidDB server on server S1 in the OFFLINE state.   |
| 12 | S 2<br>OS | <p><b>ADMIN COMMAND 'hsb state';</b></p> <p><b>ADMIN COMMAND 'hsb netcopy';</b></p>                          | <p>Check the new Primary server S2 (old Secondary) to make sure that it is still in PRIMARY ALONE state.</p> <p>Netcopy the database from the new Primary (S2) to the new Secondary (S1).</p>  |
| 13 | S 2<br>OS | <p><b>ADMIN COMMAND 'hsb status copy';</b></p> <p><b>ADMIN COMMAND 'hsb connect';</b></p>                    | <p>Verify that the netcopy succeed.</p> <p>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.</p> <p>If this step fails, then copy the entire database to the Secondary server (using "hsb copy") and then resume from step 10.</p> |

## 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 IBM solidDB.

---

# Appendix A. Configuration Parameters

This appendix discusses the server-side configuration parameters in the `solid.ini` configuration file. The server-side parameters consist of parameters in the `[Cluster]` and the `[HotStandby]` sections in the `solid.ini` configuration file.

Furthermore, this appendix discusses the High Availability Controller (HAC) configuration parameters in the `solidhac.ini` configuration file and High Availability Manager configuration parameters in the `HAManager.ini` configuration file. These parameters are specific to IBM solidDB HotStandby. For a discussion of other `solid.ini` parameters, see *IBM solidDB Administration Guide*.

For information about how to format parameter names and section headings, see *IBM 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.

For information on parameters that can be used in the `[Watchdog]` section of the `solid.ini` file, see Appendix F, *Watchdog Sample*.

Note that some parameters in sections other than the `[HotStandby]` section also affect HotStandby functionality. These other parameters are documented in Section 4.7, “Configuring HotStandby-Specific Parameters” and in *IBM 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 *IBM 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 `temporary` option 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**

| <i>[Cluster]</i>                      | Description                                     | Factory Value | Access Mode |
|---------------------------------------|---|---------------|-------------|
| <i>ReadMostlyLoadPercentAtPrimary</i> | Percentage of read load directed to the Primary | 50            | RW/Startup  |

## A.7 HotStandby Section

**Table A.2. HotStandby Parameters**

| <i>[HotStandby]</i>   | Description   | Factory Value | Access Mode |
|-----------------------|---|---------------|-------------|
| <i>1SafeMaxDelay</i>  | In 1-Safe replication, the maximum delay before a committed transaction is sent to the Secondary (in milliseconds).   | 1000          | RW          |
| <i>2SafeAckPolicy</i> | <p>This specifies the timing of the Secondary's acknowledgement when it receives a transaction from the Primary.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>• 1 = 2-safe received. The Secondary server acknowledges when it receives the data.</li> <li>• 2 = 2-safe visible. The Secondary server acknowledges when the data is "visible", i.e. when the Secondary has executed the transaction.</li> <li>• 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.</li> </ul> <p>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.</p> <p>This parameter applies only if the server is using 2-safe replication.</p> <p>NOTE! Although this parameter controls the Secondary server's behavior, this parameter is set on the Primary. The value in the Secondary's <code>solid.ini</code> value is ignored.</p> <p>(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 <i>IBM solidDB High Availability User Guide</i>.)</p> | 1             | RW          |

## A.7 HotStandby Section

| <b>[HotStandby]</b>     | <b>Description</b>  | <b>Factory Value</b>             | <b>Access Mode</b> |
|-------------------------|---|----------------------------------|--------------------|
| <i>AutoPrimaryAlone</i> | 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 <i>IBM solidDB High Availability User Guide</i> .  | No                               | RW                 |
| <i>CatchupSpeedRate</i> | <p>While the server is performing catchup, it also continues to service database requests from clients. You may use the <i>CatchupSpeedRate</i> parameter to give greater importance to responding to application requests and lower priority to catchup, or vice versa.</p> <p>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.</p>   | 70                               | RW                 |
| <i>Connect</i>          | <p>The <i>Connect</i> parameter indicates the "address" of the other HotStandby server in the pair.</p> <p>The format of the <i>Connect</i> string in the HotStandby section is the same as the format of the <i>Listen</i> parameter in the <i>[Com]</i> section (see <i>IBM solidDB Administration Guide</i> for more details).</p> <p>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 <i>Connect</i> string, the server can only be in the states that do not involve a connection, i.e. PRIMARY ALONE, SECONDARY ALONE, and STANDALONE.</p> <p>The <i>Connect</i> parameter is ignored unless the <i>HSBEnabled</i> parameter is set to "yes".</p> | No factory value.                | RW                 |
| <i>ConnectTimeout</i>   | 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.  | 0 (no timeout)<br><br>Unit: 1 ms | RW                 |

| <i>[HotStandby]</i>  | Description  | Factory Value    | Access Mode |
|----------------------|--|------------------|-------------|
|                      | <p>The <i>ConnectTimeout</i> parameter (which is useful only on certain platforms) is only used with certain administration commands. These are:</p> <p><b>hotstandby connect</b></p> <p><b>hotstandby switch primary</b></p> <p><b>hotstandby switch secondary</b></p> <p>For example, to set the timeout to 30 seconds (30000 milliseconds)</p> <pre>[HotStandby] ConnectTimeout=30000</pre> <p>See also <i>PingTimeout</i>.</p>   |                  |             |
| <i>CopyDirectory</i> | <p>The <i>CopyDirectory</i> parameter in the <i>[HotStandby]</i> section defines a name and location for the HotStandby copy operation that is performed when the user executes the command:</p> <p><b>ADMIN COMMAND 'hotstandby copy';</b></p> <p>For example, the parameter may look like:</p> <pre>[HotStandby] CopyDirectory= C:\solidDB\secondary\dbfiles</pre> <p>If you provide a relative path for the <i>CopyDirectory</i> parameter, the path will be relative to the directory that holds the Primary server's <i>solid.ini</i> file.</p> | No factory value | RW          |

## A.7 HotStandby Section

| <i>[HotStandby]</i>      | Description  | Factory Value                          | Access Mode |
|--------------------------|--|--|-------------|
|                          | <p>This parameter has no factory value, so if the directory is not specified in the <code>solid.ini</code> file, it must be provided in the <code>copy</code> command.</p> <p>Please note that <b>ADMIN COMMAND 'hotstandby netcopy'</b> as the more flexible solution is the recommended way to copy the database.</p>  |  |             |
| <i>HSBEnabled</i>        | <p>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.</p> <p>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".</p> <p>To use HotStandby, you must also specify the <i>Connect</i> parameter, either by setting it in the <code>solid.ini</code> file or by using an ADMIN COMMAND to set it.</p> | no                                     | RO          |
| <i>MaxLogSize</i>        | Maximum size of the disk-based HSB log. The factory value: unlimited   | 0<br><br>Unit: 1 byte<br>k=KB<br>m=MB  |             |
| <i>MaxMemLogSize</i>     | When the file-based logging is disabled ( <i>Logging.LogEnabled=No</i> ), 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<br><br>Unit: 1 byte<br>k=KB<br>m=MB | RO          |
| <i>NetcopyRpcTimeout</i> | Data transmission acknowledgment timeout for netcopy operation (in milliseconds)   | 30000<br><br>Unit: 1 ms                | RW          |
| <i>PingInterval</i>      | The Primary and Secondary send "ping" messages to 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.)  | 1000<br><br>Unit: 1 ms                 | RW          |

| <i>[HotStandby]</i>  | <b>Description</b>  | <b>Factory Value</b>                       | <b>Access Mode</b> |
|----------------------|---|--|--------------------|
|                      | The value is equal to the interval (in milliseconds) between two consecutive pings sent by a server. The factory value is 1000 (one second).  |  |                    |
| <i>PingTimeout</i>   | <p>The parameter specifies how long a server should wait before concluding that the other server is down or inaccessible.</p> <p>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).</p> <p>See the "PingTimeout and PingInterval Parameters [HotStandby]" chapter in <i>IBM solidDB High Availability User Guide</i>.</p> <p>See also <i>ConnectTimeout</i>.</p>   | <p>4000</p> <p>Unit: 1 ms</p>              | RW                 |
| <i>PrimaryAlone</i>  | This parameter is deprecated. Use the <i>AutoPrimaryAlone</i> parameter.  | No   | RW                 |
| <i>SafenessLevel</i> | <p>This parameter sets the safeness level of the replication protocol.</p> <p>By using the "auto" value, you can allow the safeness level to dynamically change in relation to the durability level. If you set <i>SafenessLevel</i> to "auto" and set the durability to relaxed by using the SET DURABILITY command or the <i>DurabilityLevel</i> parameter, the safeness level is set to 1-safe, and when you set the durability level to strict, the safeness level is set to 2-safe. However, if <i>DurabilityLevel</i> is set to 2 (Adaptive Durability), the "auto" setting has no effect - the safeness level will always be 2-safe.</p> | Possible values are: 1safe, 2safe and auto | RW                 |

## A.8 High Availability Controller Configuration Parameters

The following tables explain the High Availability Controller configuration parameters in the `solidhac.ini` configuration file. The configuration file is divided into different sections; the tables follow this sectioning. The parameters are presented in the same order as they are in the configuration file.



**Table A.3. HAC Configuration Parameters: [*HACcontroller*] Section**

| [ <i>HACcontroller</i> ]        | Description   | Factory Value |
|---------------------------------|---|---------------|
| <i>Listen</i>                   | The protocol and the port of the HAC listening port. This port is used by High Availability Manager to send commands to HAC.  | tcp 3135      |
| <i>StartInAutomatic-Mode</i>    | <p>This parameter defines whether the HAC starts execution in the automatic mode. This is dynamically changeable parameter.</p> <p>When in the AUTOMATIC mode, HAC automatically tries to maintain the database availability by monitoring the states of the servers and executing the database related actions when necessary. In contrast, in the ADMINISTRATIVE mode, HAC only monitors the states of system components.</p> <p>See also the <i>PreferredPrimary</i> parameter.</p> <p>The possible values are "Yes" and "No".</p> | Yes           |
| <i>EnableDBProcessControl</i>   | <p>This parameter allows HAC to manage the local database process by automatically starting the database, and by providing the user with commands to shutdown and restart the database process.</p> <p>This value is only valid if the HAC is in the ACTIVE mode.</p> <p>The possible values are "Yes" and "No".</p> <p>This is an optional parameter.</p>  | No            |
| <i>EnableAutoNet-copy</i>       | <p>This parameter defines whether HAC can initiate a netcopy when necessary.</p> <p>The possible values are "Yes" and "No".</p> <p>This is an optional parameter.</p>   | No            |
| <i>RequiredConnect-Failures</i> | This parameter defines the number of consecutive failed connect attempts that are required before the server is considered to have failed.  | 1             |

| [HAController]       | Description   | Factory Value |
|----------------------|---|---------------|
|                      | The possible values are numerical values from 1 to unlimited.<br><br>This is an optional parameter.   |               |
| <i>CheckTimeout</i>  | This parameter defines the timeout in milliseconds for SQL commands in CHECK mode.<br><br>The possible values are milliseconds from 1 to unlimited.   | 50            |
| <i>CheckInterval</i> | This parameter defines the interval between consecutive server state checks in milliseconds.<br><br>The possible values are milliseconds from 1 to unlimited.<br><br>This is an optional parameter. | 1000          |
| <i>Username</i>      | Username for HAC.<br><br>This is a mandatory parameter.   |               |
| <i>Password</i>      | Password for HAC.<br><br>This is a mandatory parameter.   |               |
| <i>DBUsername</i>    | Username for the HotStandby server.<br><br>This is a mandatory parameter.   |               |
| <i>DBPassword</i>    | Password for the HotStandby server.<br><br>This is a mandatory parameter.   |               |

**Table A.4. HAC Configuration Parameters: [*LocalDB*] Section**

| [LocalDB]          | Description   | Factory Value               |
|--------------------|---|-----------------------------|
| <i>Connect</i>     | This parameter allows the HAC to connect to a local database. The local database is defined as a network name. The network name consists of a communication protocol and a server name.<br><br>This is a mandatory parameter. | tcp 2125                    |
| <i>StartScript</i> | This parameter declares the name of the script, which is used to initiate the database process. This parameter is   | /home/solid/start_sol-id.sh |

| [LocalDB]               | Description  | Factory Value |
|-------------------------|--|---------------|
|                         | <p>optional, but it is recommended to set it if the HAC controls the database process. HAC uses this the given script automatically if the HAC is in the ACTIVE mode and if the <i>EnableDBProcessControl</i> parameter is set to "Yes".</p> <p>This is an optional parameter.</p>   |               |
| <i>PreferredPrimary</i> | <p>This parameter defines whether the local server becomes the Primary if the logpos values of both servers are equal. If both servers have value "No" or have no value, the first server becomes the Primary.</p> <p>This value is only valid if the <i>StartInActiveMode</i> parameter has value "Yes".</p> <p>The possible values are "Yes" and "No".</p> <p>This is an optional parameter.</p> | No            |

**Table A.5. HAC Configuration Parameters: [RemoteDB] Section**

| [RemoteDB]     | Description   | Factory Value          |
|----------------|---|------------------------|
| <i>Connect</i> | <p>This parameter allows the HAC to connect to a remote database. The local database is defined by giving the communication protocol, the remote server IP address, and its port.</p> <p>This is a mandatory parameter.</p> | tcp 192.168.3.123 2125 |

**Table A.6. HAC Configuration Parameters: [ERE] Section**

| [ERE]        | Description   | Factory Value     |
|--------------|---|-------------------|
| <i>EREIP</i> | <p>This parameter identifies the IP address of an External Reference Entity.</p> <p>See also the <i>RequiredPingFailures</i> parameter.</p> <p>This is an optional parameter.</p> | EREIP=192.168.3.1 |

| [ERE]                       | Description   | Factory Value |
|-----------------------------|---|---------------|
| <i>RequiredPingFailures</i> | <p>This parameter defines the maximum number of consecutive Ping calls that must fail before HAC concludes that the server is disconnected from the ERE and isolated from the client network.</p> <p>This parameter can only be used with ERE.</p> <p>The possible values are numerical values from 1 to unlimited.</p> <p>This is an optional parameter.</p> | 3             |

## A.9 High Availability Manager Configuration Parameters

High Availability Manager configuration parameters in the `HAManager.ini` configuration file.

**Table A.7. High Availability Manager Configuration Parameters**

| Parameter                                  | Description  |
|--|--|
| <i>Header_text</i>                         | <p>This parameter defines the header text for the HA manager. This value is shown in the user interface.</p> <p>This is a mandatory parameter.</p> |
| <i>Server1_host</i><br><i>Server2_host</i> | These two parameters define the host names of the HAC instances.   |
| <i>Server1_name</i><br><i>Server2_name</i> | These two parameters define the names of the HAC instances.  |
| <i>Server1_pass</i><br><i>Server2_pass</i> | These two parameters define the passwords of the HAC instances.  |
| <i>Server1_port</i><br><i>Server2_port</i> | These two parameters define the ports of the HAC instances.  |
| <i>Server1_user</i>                        | These two parameters define the usernames of the HAC instances.  |

| <b>Parameter</b>    | <b>Description</b>   |
|---------------------|--|
| <i>Server2_user</i> |  |
| <i>Window_title</i> | This parameter defines the window title for the HA manager. This value is shown in the user interface.<br><br>This is a mandatory parameter. |

---

---

# Appendix B. Error Codes

This appendix documents error codes that are related to HotStandby. Most other server error codes are documented in *IBM 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 "IBM solidDB HotStandby Errors" and "IBM solidDB High Availability Controller Errors" are ADMIN COMMAND result set values, whereas all "IBM solidDB Communication Errors" are returned by the driver.

The error categories covered in the tables contained in this appendix are:

- IBM solidDB HotStandby Errors (14009 - 147xx, 307xx)

These errors occur when using specific HotStandby commands, which are IBM solidDB SQL extensions.

- IBM solidDB High Availability Controller Errors (17xxx)

These errors occur when using specific High Availability Controller commands.

- IBM solidDB Database Errors (10002 - 10050)

These errors are detected by IBM solidDB and are sent to the client application. They may demand administrative actions.

- IBM solidDB Database Table Errors (13xxx)

These errors are caused by erroneous SQL statements and are detected by IBM solidDB. Administrative actions are not needed.

- IBM solidDB 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      |

| Error/Status code | Description  |
|-------------------|--|
|                   | <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hotstandby status switch'</b></p> <p><b>ADMIN COMMAND 'hotstandby status catchup'</b></p> <p><b>ADMIN COMMAND 'hotstandby status copy'</b></p> <p>Meaning: The switch process, catchup process, or copy/netcopy process is still active.</p>  |
| 14007             | <p>CONNECTING</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hotstandby status connect'</b></p> <p>Meaning: The Primary and Secondary servers are in the process of connecting.</p>  |
| 14008             | <p>CATCHUP</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hotstandby status connect'</b></p> <p>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.</p>  |
| 14009             | <p>No server switch occurred before.</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hotstandby status switch'</b></p> <p>Meaning: The switch process has never happened between the servers.</p>   |
| 14501             | <p>Operation failed.</p> <p>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.</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hotstandby switch primary'</b></p> |




| Error/Status code | Description   |
|-------------------|---|
|                   | <p><b>ADMIN COMMAND 'hotstandby switch secondary'</b></p> <p><b>ADMIN COMMAND 'hotstandby cominfo'</b></p> <p><b>ADMIN COMMAND 'hotstandby connect'</b></p> <p><b>ADMIN COMMAND 'hotstandby status switch'</b></p> <p><b>ADMIN COMMAND 'hotstandby set standalone'</b></p> <p><b>ADMIN COMMAND 'hotstandby copy'</b></p> <p><b>ADMIN COMMAND 'hotstandby netcopy'</b></p> |
| 14502             | <p>RPC parameter is invalid</p> <p>Meaning: some of the connection info provided in the HSB connect string is erroneous and the connection to another server failed.</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hotstandby status connect'</b></p>  |
| 14503             | <p>Communication error, connection lost.</p> <p>Meaning: There was a communication error and the other server was not found. There was a failure to connect to the other server.</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hotstandby status connect'</b></p>  |
| 14520             | <p>Server is HotStandby secondary server, no connections are allowed.</p>   |
| 14522             | <p>HotStandby copy directory not specified.</p> <p>Meaning: No copy directory is specified.</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hotstandby copy'</b></p> <p>To solve this problem, either specify the directory as part of the command, e.g.</p>   |

| Error/Status code | Description  |
|-------------------|--|
|                   | <p><b>ADMIN COMMAND 'hotstandby copy \Secondary\dbfiles\'</b></p> <p>or else set the <i>CopyDirectory</i> parameter in the <i>solid.ini</i> configuration file.</p>  |
| 14523             | <p>Switch process is already active.</p> <p>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.</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hotstandby switch primary'</b></p> <p><b>ADMIN COMMAND 'hotstandby switch secondary'</b></p> <p><b>ADMIN COMMAND 'hotstandby status switch'</b></p> |
| 14524             | <p>HotStandby databases have a different base database, database time stamps are different.</p> <p>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.</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hotstandby connect'</b></p> <p><b>ADMIN COMMAND 'hotstandby status switch'</b></p>  |
| 14525             | <p>HotStandby databases are not properly synchronized.</p> <p>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 <b>-x backupserver</b> and then netcopy the Primary's database to the Secondary.</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hotstandby connect'</b></p> <p><b>ADMIN COMMAND 'hotstandby status switch'</b></p>   |
| 14526             | <p>Invalid argument.</p>   |

| Error/Status code | Description  |
|-------------------|--|
|                   | <p>Meaning: An argument used in the HotStandby ADMIN COMMAND is unknown or invalid.</p> <p>All HotStandby commands can return this error in the result set of the ADMIN COMMAND.</p> <p>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:</p> <p><b>ADMIN COMMAND 'hotstandby switch primary'</b></p> <p><b>ADMIN COMMAND 'hotstandby switch secondary'</b></p>   |
| 14527             | <p>This is a non-HotStandby server.</p> <p>Meaning: The command was executed on a server that is not configured for HotStandby.</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hotstandby connect'</b></p> <p><b>ADMIN COMMAND 'hotstandby status switch'</b></p> <p><b>ADMIN COMMAND 'hotstandby switch primary'</b></p> <p><b>ADMIN COMMAND 'hotstandby switch secondary'</b></p> <p><b>ADMIN COMMAND 'hotstandby state'</b></p>   |
| 14528             | <p>Both HotStandby databases are primary databases.</p> <p>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.</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hotstandby connect'</b></p> <p><b>ADMIN COMMAND 'hotstandby status switch'</b></p> |
| 14535             | <p>Server is already a primary server.</p>   |

| Error/Status code | Description  |
|-------------------|--|
|                   | <p>Meaning: The server you are trying to switch to Primary is already in one of the PRIMARY states.</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hotstandby switch primary'</b></p>  |
| 14536             | <p>Server is already a secondary server.</p> <p>Meaning: The server you are trying to switch to Secondary is already in one of the SECONDARY states.</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hotstandby switch secondary'</b></p>   |
| 14537             | <p>HotStandby connection is broken.</p> <p>Meaning: This command is returned from both the Primary and Secondary server.</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hotstandby status connect'</b></p> <p>One possible cause of this problem is an incorrect Connect string in the Secondary's <code>solid.ini</code> 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.)</p> |
| 14538             | <p>Server is not HotStandby primary server.</p> <p>Meaning: To issue this command, the server must be a HotStandby Primary server.</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hotstandby copy copy_directory'</b></p> <p><b>ADMIN COMMAND 'hotstandby netcopy'</b></p> <p><b>ADMIN COMMAND 'hotstandby connect'</b></p> <p><b>ADMIN COMMAND 'hotstandby set primary alone'</b></p>   |

| Error/Status code | Description  |
|-------------------|--|
|                   | <b>ADMIN COMMAND 'hotstandby set standalone'</b>   |
| 14539             | <p>Operation Refused.</p> <p>This error code is given when one of the following situations occurs:</p> <ul style="list-style-type: none"> <li>The user issued a <b>netcopy</b> 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.)</li> </ul> <p>To solve the problem, re-start the "Secondary" with the <b>-x backupserver</b> command-line option, then try again to issue the <b>netcopy</b> command to the Primary.</p> <p> <b>Warning</b></p> <p>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 <b>netcopy</b> to put them back in sync would result in writing over some transactions that have already been committed in the "Secondary" server.</p> <ul style="list-style-type: none"> <li>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.</li> </ul> <p>When you use linked library access, you can provide "callback" functions by using the <code>SSCSetNotifier</code> 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.</p> <p>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).</p> |
| 14540             | Server is already a non-HotStandby server.   |
| 14541             | HotStandby configuration in <code>solid.ini</code> conflicts with <b>ADMIN COMMAND 'HSB SET STANDALONE'</b> .  |
| 14542             | Server in backupserver mode. Operation refused.  |

## B.1 HotStandby Errors and Status Codes

| Error/Status code | Description   |
|-------------------|---|
| 14543             | Invalid command. The database is a hotstandby database but, hotstandby section not found in <code>solid.ini</code> configuration file.  |
| 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             | <p>Switch failed. The server or the remote server cannot switch from primary alone to secondary server. Catchup should be done first before switch.</p> <p>Meaning: This command is returned when a state switch to <code>SECONDARY</code> is executed from a local or remote Primary server that is in the <code>PRIMARY ALONE</code> 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.</p> <p>HotStandby commands that return this error:</p> <p><b>ADMIN COMMAND 'hotstandby switch secondary'</b></p> |
| 14547             | The value for the <code>-R</code> option (Read Timeout) was missing or invalid.   |
| 14548             | <p>Switch failed. The server in Standalone cannot be switched to a secondary.</p> <p>Meaning: This command is returned when a state switch to <code>SECONDARY</code> is executed from a local or remote Primary server that is in the <code>STANDALONE</code> 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.</p> <p>HotStandby commands that return this error:</p> <p><b>ADMIN COMMAND 'hotstandby switch secondary'</b></p>  |
| 14549             | <p>HotStandby transaction is active.</p> <p>Meaning: If the HotStandby connection is broken, Primary server must be set to alone mode or switched to secondary mode before shutdown.</p>  |
| 14550             | Hotstandby connect parameter can be changed only when the primary is not connected to secondary.  |
| 14551             | Maximum number of <code>START AFTER COMMIT</code> statements reached.   |
| 14552             | Server is in backup server mode, no connections are allowed.  |

## B.1 HotStandby Errors and Status Codes

| Error/Status code | Description   |
|-------------------|---|
| 14700             | <p>Rejected connection, both servers in PRIMARY role.</p> <p>Meaning: Command '<b>hsb connect</b>' returns this error if both nodes are in same role.</p>   |
| 14701             | <p>14701:Rejected connection, both servers in SECONDARY role.</p> <p>Meaning: Command '<b>hsb connect</b>' returns this error if both nodes are in same role.</p>   |
| 14702             | <p>14702:Operation failed, catchup is active.</p> <p>Meaning:</p> <p>While the servers are performing catchup, you will get this error if you issue any of the following commands on the Primary: '<b>hsb switch secondary</b>', '<b>hsb set secondary alone</b>', '<b>hsb set standalone</b>', '<b>hsb connect</b>', '<b>hsb copy</b>' or '<b>hsb netcopy</b>'.</p> <p>While the servers are performing catchup, you will get this error if you issue any of the following commands on the Secondary: '<b>hsb switch primary</b>', '<b>hsb set secondary alone</b>', '<b>hsb set primary alone</b>', '<b>hsb set standalone</b>', or '<b>hsb connect</b>'.</p> |
| 14703             | <p>Operation failed, copy is active.</p> <p>Meaning:</p> <p>While the Primary is doing copy or netcopy, the following commands returns this error: '<b>hsb switch secondary</b>', '<b>hsb set secondary alone</b>', '<b>hsb set standalone</b>', '<b>hsb connect</b>', '<b>hsb disconnect</b>', '<b>hsb copy</b>' or '<b>hsb netcopy</b>'.</p>  |
| 14704             | <p>HotStandby copy or netcopy is only allowed when primary is in alone state.</p> <p>Meaning:</p> <p>This error is returned if the server is in PRIMARY ACTIVE state and the command '<b>hsb copy</b>' or '<b>hsb netcopy</b>' is issued.</p>   |
| 14705             | <p>Setting to STANDALONE is not allowed in this state.</p> <p>Meaning:</p> <p>If the server is in PRIMARY ACTIVE state and you issue the command '<b>hsb set standalone</b>', then you will get this message.</p>   |
| 14706             | <p>Invalid read thread mode for HotStandby, only mode 2 is supported.</p>   |

| <b>Error/Status code</b> | <b>Description</b>  |
|--------------------------|---|
| 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.  |
| 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<br><br>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<br><br>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><br><br>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 High Availability Controller Errors and Status Codes

**Table B.2. High Availability Controller Errors and Status Codes**

| <b>Error/Status code</b> | <b>Description</b>   |
|--------------------------|--|
| 17501                    | HAC is shutting down.<br><br>ADMIN COMMANDs that may return this status in the result set of the command:<br><br><b>ADMIN COMMAND 'hac shutdown'</b> |



| Error/Status code | Description   |
|-------------------|---|
|                   | Meaning: The High Availability Controller is shutting down.   |
| 17502             | <p>Command failed, HAC is suspended.</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hac suspend'</b></p> <p><b>ADMIN COMMAND 'hac gethsbdstate'</b></p> <p><b>ADMIN COMMAND 'hac getdbstate'</b></p> <p><b>ADMIN COMMAND 'hac shutdowndb'</b></p> <p><b>ADMIN COMMAND 'hac restartdb'</b></p> <p><b>ADMIN COMMAND 'hac switchdb'</b></p> <p><b>ADMIN COMMAND 'hac statemachinestate'</b></p> <p><b>ADMIN COMMAND 'hac getereip'</b></p> <p><b>ADMIN COMMAND 'hac pingere'</b></p> <p>Meaning: The command execution failed and the High Availability Controller is suspended.</p> |
| 17503             | <p>Unknown command. Enter 'hac commands' for help.</p> <p>Meaning: Incorrect admin command syntax.</p>  |
| 17504             | <p>HAC is already running.</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hac resume'</b></p>   |
| 17506             | <p>HSB state does not allow for switchover.</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p> <p><b>ADMIN COMMAND 'hac switchdb'</b></p>  |
| 17507             | <p>Cannot execute command.</p> <p>ADMIN COMMANDs that may return this status in the result set of the command:</p>  |

| Error/Status code | Description   |
|-------------------|---|
|                   | <b>ADMIN COMMAND 'hac shutdowndb'</b>   |
| 17509             | Restarting database server failed, see <code>solmsg.out</code> for details.<br><br>ADMIN COMMANDs that may return this status in the result set of the command:<br><br><b>ADMIN COMMAND 'hac restartdb'</b>                 |
| 17510             | Cannot connect to database server.<br><br>ADMIN COMMANDs that may return this status in the result set of the command:<br><br><b>ADMIN COMMAND 'hac switchdb'</b>   |
| 17511             | Database server was not shut down.<br><br>ADMIN COMMANDs that may return this status in the result set of the command:<br><br><b>ADMIN COMMAND 'hac restartdb'</b>  |
| 17513             | Switchover failed.<br><br>ADMIN COMMANDs that may return this status in the result set of the command:<br><br><b>ADMIN COMMAND 'hac switchdb'</b>   |
| 17514             | ERE IP is not specified in the configuration file.<br><br>ADMIN COMMANDs that may return this status in the result set of the command:<br><br><b>ADMIN COMMAND 'hac getereip'</b><br><br><b>ADMIN COMMAND 'hac pingere'</b> |
| 17516             | HAC is already active.<br><br>ADMIN COMMANDs that may return this status in the result set of the command:<br><br><b>ADMIN COMMAND 'hac setactive'</b><br><br><b>ADMIN COMMAND 'hac pingere'</b>                            |
| 17517             | HAC is already passive.<br><br>ADMIN COMMANDs that may return this status in the result set of the command:   |

| Error/Status code | Description                    |
|-------------------|--------------------------------|
|                   | ADMIN COMMAND 'hac setpassive' |

## B.3 IBM solidDB Database Errors

**Table B.3. IBM solidDB Database Errors**

| Error code           | Description   |
|----------------------|---|
| Database Error 10002 | Operation failed.<br><br>Meaning: The connect operation failed with an unexpected error. Most likely, the servers are not properly synchronized.  |
| Database Error 10013 | Transaction is read-only.<br><br>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.<br><br>Meaning: You have tried to start a backup or copy when one is already in progress.   |
| Database Error 10024 | Illegal backup directory " <i>directory_name</i> ".<br><br>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 ' <i>directory_name</i> ' does not exist.<br><br>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.<br><br>Meaning: This operation cannot be executed on a HotStandby Secondary server.<br><br>In order for the requested operation to succeed, the server must be a Primary.   |
| Database Error 10046 | Operation failed, data dictionary operation is active.<br><br>Meaning: A data dictionary operation is currently in progress.  |
| Database Error 10047 | Replicated transaction is aborted.  |

### B.3 IBM solidDB Database Errors

| Error code           | Description  |
|----------------------|--|
|                      | <p>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.</p>  |
| Database Error 10048 | <p>Replicated transaction contains data dictionary changes, normal update operations are not allowed.</p> <p>Meaning: HotStandby mode restricts data dictionary operations; for example, CREATE TABLE cannot be mixed with normal update operations.</p> <p>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.</p> |
| Database Error 10049 | <p>The remote server is not a secondary server.</p> <p>Meaning: The server that you specified in the command is not in a SECONDARY state.</p>  |
| Database Error 10050 | <p>Replicated operation updated BLOB columns.</p> <p>Meaning: BLOB columns cannot be replicated to the Secondary server.</p>   |
| Database error 10078 | User rolled back the transaction.  |
| Database error 10079 | Cannot remove filespec. File is already in use.  |
| Database error 10080 | <p>HotStandby Secondary server can not execute operation received from Primary server.</p> <p>Meaning: A possible cause for this error is that the database did not originate from the Primary server using HotStandby copy or netcopy command.</p>  |
| Database error 10081 | <p>The database file is incomplete or corrupt.</p> <p>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.</p>   |
| 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.4 IBM solidDB Table Errors

**Table B.4. IBM solidDB Table Errors**

| Error code        | Description   |
|-------------------|---|
| Table Error 13068 | <p>Server shutdown in progress</p> <p>Meaning: You are unable to complete this operation because server shutdown is in progress.</p>  |
| Table Error 13123 | <p>Table '<i>table_name</i>' is not empty</p> <p>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.</p>                                     |
| Table Error 13167 | <p>Only M-tables can be transient</p> <p>Meaning: You cannot create a transient table that is disk-based. For example, the following SQL statement will get this error message:</p> <pre data-bbox="396 874 1108 899">CREATE TRANSIENT TABLE t1 (i INT) STORE DISK;</pre>   |
| Table Error 13170 | <p>Only M-tables can be temporary</p> <p>Meaning: You cannot create a temporary table that is disk-based. For example, the following SQL statement will get this error message:</p> <pre data-bbox="396 1159 1108 1183">CREATE TEMPORARY TABLE t1 (i INT) STORE DISK;</pre> |

## B.5 IBM solidDB Communication Errors

**Table B.5. IBM solidDB Communication Errors**

| Error code                   | Description   |
|------------------------------|---|
| Communication Error<br>21306 | Server " <i>server_name</i> " not found, connection failed.<br><br>Meaning: The Secondary server was not found. <ul style="list-style-type: none"><li>• Check that the server is running.</li><li>• Check that the network name is valid.</li><li>• Check that the server is listening to the specified network name.</li></ul> |
| Communication Error<br>21308 | "Connection is broken (%s '%s' operation failed with code %d)".<br><br>For example,<br><br>"Connection is broken (TCP/IP 'Write' operation failed with code 7)."<br><br>See the recommended actions for error 21306.  |

---

# Appendix C. Summary of HotStandby Administrative Commands

This chapter summarizes the administrative commands available with HotStandby.

The tool you are using (for example IBM solidDB SQL Editor (solsql) or IBM solidDB Remote Control (solcon)) affects how you must enter the command.

Using IBM solidDB SQL Editor (teletype), you enter the command using the syntax shown below:

```
ADMIN COMMAND 'hotstandby switch primary';
```

If you are entering these commands in IBM solidDB 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 tables below, we use the shortest possible form; we omit the "ADMIN COMMAND" and the quotes, and we also use the abbreviations 'hsb' and 'hac'.

For more information about IBM solidDB Remote Control (teletype) or IBM solidDB SQL Editor, refer to "Using IBM solidDB Data Management Tools" in *IBM 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.

## C.1 HotStandby Commands

**Table C.1. HotStandby Commands**

| Command                                   | Explanation  |
|---|--|
| <b>hsb cominfo</b>                        | <p>Returns the communication information (connect string) used to connect to the other server. This is usually the value of the <i>Connect</i> parameter in the <i>HotStandby</i> section of the <i>solid.ini</i> configuration file, but might also have been set with the command:</p> <p><b>ADMIN COMMAND 'hsb parameter connect &lt;connect_string&gt;';</b></p> <p>You can use this information in an application to connect to other servers.</p>  |
| <b>hsb connect</b>                        | <p>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.</p> <p>This command can be executed on either the Primary or the Secondary.</p> <p>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</p>  |
| <b>hsb copy [ <i>directory_name</i> ]</b> | <p>Note: This command is deprecated. It is recommended that you use the <b>hsb netcopy</b> command instead.</p> <p>You can use the <b>hsb copy</b> 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.</p> <p>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 6.3, “Synchronizing Primary and Secondary Servers”.</p> |




| Command               | Explanation  |
|-----------------------|--|
|                       | <p>If the optional <i>directory_name</i> is specified, the database files are copied to that directory; otherwise it is copied to the directory specified with the <i>copydirectory</i> parameter in the <i>[Hotstandby]</i> section of the <i>solid.ini</i> configuration file. Because the <b>hsb copy</b> command does not copy the <i>solid.ini</i> configuration file or log files, it is recommended that you make this directory different from the normal backup directory.</p> <p>The Primary can execute the <b>hsb copy</b> command only if the Primary is in PRIMARY ALONE state. During and after the command, the server remains in PRIMARY ALONE state. After the command has been completed, you may start the Secondary server and then connect the two servers.</p>  |
| <b>hsb disconnect</b> | <p>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.)</p> <p>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.</p> <p>This command is valid on both the Primary and the Secondary.</p> <p>Note that using the shutdown command</p> <p><b>ADMIN COMMAND 'shutdown';</b></p> <p>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.</p> |
| <b>hsb logpos</b>     | <p>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.</p>   |

| Command            | Explanation   |
|--------------------|---|
|                    | <p>For a detailed explanation of how to use this command, see Section 6.7, “Choosing Which Server to Make Primary”.</p> <p>A typical output is shown below:</p> <pre data-bbox="396 430 888 548">ADMIN COMMAND 'hsb logpos'; RC TEXT -- ---- 0 00000000000000000000871:PRIMARY</pre> <p>The output consists of the log operation ID and the server's previous role, which in this example was Primary.</p>  |
| <b>hsb netcopy</b> | <p>This command is used to copy the Primary database or diskless in-memory data to a Secondary server using the <i>Connect</i> parameter specified in the <i>[HotStandby]</i> section of <i>solid.ini</i>. Once the connect string is used to connect to the Secondary server, the database files are copied through the network link.</p> <p>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 6.3, “Synchronizing Primary and Secondary Servers”.</p> <p>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 6.3.5, “Copying a Primary Database to a Secondary Over the Network” for details.</p> <p>The Primary server must be in PRIMARY ALONE state to issue this command.</p> <p>After the command has completed (successfully or unsuccessfully), the Primary server remains in the same state.</p> <p>If the copy is completed successfully, then the Secondary server is automatically switched to SECONDARY ALONE state.</p> <p>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.</p> |

| Command              | Explanation   |
|----------------------|---|
| <b>hsb parameter</b> | <p>(Deprecated) This command allows you to set HSB-specific parameters such as <i>AutoPrimaryAlone</i>, <i>Connect</i>, and <i>PingTimeout</i>. For a complete description of each of these parameters, see Appendix A, <i>Configuration Parameters</i>.</p> <p>Note that when you set the value of one of some parameters, the command takes effect immediately, but is not written to the <code>solid.ini</code> configuration file before a shut-down is executed.</p> <p>The syntax for this command is:</p> <pre>ADMIN COMMAND 'hsb parameter param_name param_value';</pre> <p>Note that this command does not use an equals sign. Thus it differs from the otherwise similar command (recommended):</p> <pre>ADMIN COMMAND 'parameter hotstandby.param_name = param_value';</pre>  |
| <b>hsb role</b>      | <p>Note: This command is deprecated. Please use <b>hsb state</b> instead.</p> <p>Returns one of the following roles in the result set:</p> <ul style="list-style-type: none"> <li>• 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.</li> <li>• 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.</li> <li>• 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.</li> </ul> |

| Command                      | Explanation  |
|------------------------------|--|
|                              | <ul style="list-style-type: none"> <li>• 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.</li> <li>• 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.</li> <li>• SECONDARY, if the connected server is a normal Secondary server. This means the server receives and applies transactions from the Primary.</li> <li>• SECONDARY BROKEN, if the Secondary server has a broken connection to the Primary server.</li> <li>• 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.</li> </ul> <p>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.</p> <p>This command returns the same information as the SQL function: HOTSTANDBY_ROLE.</p> |
| <b>hsb set primary alone</b> | <p>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.</p> <p>This command can be used to implement fast failovers. When the 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 HAC before the Secondary has detected the failure (the delay is dictated by the <i>PingTimeout</i> and <i>PingInterval</i> parameters).</p>  |



| Command                        | Explanation   |
|--------------------------------|---|
|                                | <p>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.</p> <p> <b>Note</b></p> <p>The alternative way of executing failovers is to use the <b>hsb switch primary</b> command.</p> <p>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, a connection is successfully made to the Secondary server, or the server runs out of space for the transaction log.</p> <p>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.</p> <p>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 4.2.2, "Network Partitions and Dual Primaries").</p> <p>See also the command '<b>hsb switch primary</b>'.</p> |
| <b>hsb set secondary alone</b> | 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.   |
| <b>hsb set standalone</b>      | 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.  |
| <b>hsb state</b>               | Returns one of the following states in the result set:  |

| Command                  | Explanation  |
|--------------------------|--|
|                          | <ul style="list-style-type: none"> <li>• 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.</li> <li>• 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.</li> <li>• 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.</li> <li>• 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.</li> <li>• SECONDARY ACTIVE, if the connected server is a normal Secondary server. This means the server receives and applies transactions from the Primary.</li> <li>• SECONDARY ALONE, if the Secondary server has a broken connection to the Primary server.</li> </ul> <p>If <b>ADMIN COMMAND 'hsb state'</b> is issued on a server that is not configured for HotStandby, the following error message is returned: 14527: This is a non-HotStandby Server.</p> <p>This command returns the same information as the SQL function: HOT-STANDBY_STATE. Read section <i>Using Function HOTSTANDBY_STATE</i> in the section called “Detecting HotStandby Server Failure in Client Applications” for details on this function.</p> <p>Refer to Appendix D, <i>Server State Transitions</i>, for an overview of HotStandby state transitions that occur while performing administrative and troubleshooting operations.</p> |
| <b>hsb status option</b> | <p>Returns HotStandby status information. The option may be any of the following:</p> <ul style="list-style-type: none"> <li>• catchup</li> <li>• connect</li> <li>• copy</li> </ul>   |

| Command                   | Explanation  |
|---------------------------|--|
|                           | <ul style="list-style-type: none"> <li>• switch</li> </ul> <p>For more details, see the descriptions of the individual commands/options below, e.g. '<b>hsb status catchup</b>'.</p> <p>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.</p>   |
| <b>hsb status catchup</b> | <p>Indicates whether or not the server is doing catchup, i.e. when the Secondary reads the Primary's transaction log and applies the changes.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>• ACTIVE</li> <li>• NOT ACTIVE</li> </ul>   |
| <b>hsb status connect</b> | <p>Status information returned:</p> <ul style="list-style-type: none"> <li>• CONNECTED - Connect active. This information is returned from both the Primary and Secondary servers.</li> <li>• CONNECTING - The Primary server and Secondary server are connecting to each other. This information is returned from both the Primary and Secondary servers.</li> <li>• 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.</li> <li>• BROKEN - Connection between the Primary and Secondary server is broken. This information is returned from both the Primary and Secondary servers.</li> </ul> <p>NOTE: This command returns the same information as the SQL function <code>HOTSTANDBY_CONNECTSTATUS</code>. Read section <i>Using Function HOTSTANDBY_CONNECTSTATUS</i> in the section called "Switching the Application to the New Primary" for details on this function.</p> |

| Command                   | Explanation  |
|---------------------------|--|
| <b>hsb status copy</b>    | <p>This command allows you to check the result of the last <b>hsb copy</b> or <b>hsb netcopy</b> command. Note that this status command always uses the keyword "copy", even if you are checking the result of a <b>netcopy</b> rather than a <b>copy</b>.</p> <p>Status information returned:</p> <ul style="list-style-type: none"> <li>• SUCCESS - Copy completed successfully.</li> <li>• ACTIVE - Copy process is still active.</li> <li>• ERROR <i>number</i> - Copy failed with error code <i>number</i>.</li> </ul>  |
| <b>hsb status switch</b>  | <p>Returns HotStandby switch status information. Status information returned:</p> <ul style="list-style-type: none"> <li>• ACTIVE - Copy process is still active.</li> <li>• SUCCESS - Copy completed successfully.</li> <li>• ERROR <i>number</i> - Copy failed with error code <i>number</i>.</li> <li>• NO SERVER SWITCH OCCURRED BEFORE - No switch has happened before.</li> </ul>  |
| <b>hsb switch primary</b> | <p>Switches the database server to PRIMARY. The command starts a switch process, which can be monitored using command hsb status switch.</p> <p>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.</p> <p>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.</p> <p>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 '<b>hsb set primary alone</b>' command. On the other hand, this method secures better against a possibility of "dual primaries".</p> |



| Command                     | Explanation  |
|-----------------------------|--|
|                             | See also the command ' <b>hsb set primary alone</b> '.   |
| <b>hsb switch secondary</b> | <p>Switches the database to Secondary state. All active write transactions are terminated.</p> <p> <b>Note</b></p> <p>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 <b>hsb switch primary</b> command).</p> <p> <b>Note</b></p> <p>If the <b>switch</b> 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: Administrative commands do force the start of a new transaction if one is not already open, however.)</p> |

## C.2 High Availability Controller Commands

Table C.2. High Availability Controller Commands

| Command  | Explanation                              |
|--|--|
| <b>hac shutdown</b><br>Abbreviation: <b>hac sd</b> | This command terminates the HAC process. |

---

---

# 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.




**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 <b>ADMIN COMMAND 'hotstandby netcopy'</b> 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 <i>AutoPrimaryAlone = Yes</i> .<br><br>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 <i>AutoPrimaryAlone = No</i> .<br><br>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 | <b>ADMIN COMMAND 'hotstandby set standalone'</b> at the Primary  | STANDALONE                   | Unchanged  |
| PRIMARY ACTIVE | <b>ADMIN COMMAND 'hotstandby switch secondary'</b> at the Primary or <b>ADMIN COMMAND 'hotstandby switch primary'</b> at the Secondary.  | SECONDARY ACTIVE             | SECONDARY ALONE                                  |
| PRIMARY ACTIVE | <b>ADMIN COMMAND 'hotstandby disconnect'</b> at the Primary.   | PRIMARY ALONE                | PRIMARY ALONE                                    |
| PRIMARY ALONE  | <b>ADMIN COMMAND 'hotstandby copy'</b> or <b>ADMIN COMMAND 'hotstandby netcopy'</b> at the Primary.<br><br>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                                    |

D.1 HotStandby State Transition Diagram

| 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... |
|---------------------------------|---|--|--|
|                                 | <p>state to PRIMARY ACTIVE, you must issue the "connect" command: <b>ADMIN COMMAND 'hotstandby connect'</b>;</p> <p>NOTE: If you are using a diskless server without file access to the Secondary server, you must use netcopy, not copy.</p>   |  |  |
| PRIMARY ALONE                   | <p><b>ADMIN COMMAND 'hotstandby connect'</b> at the Primary</p> <p>NOTE: The above command is used to connect to the Secondary server, which is now fixed, or a server other than the failed Secondary.</p>   | PRIMARY ACTIVE<br>(after the catchup is completed) | Unchanged  |
| PRIMARY ALONE                   | <b>ADMIN COMMAND 'hotstandby set standalone'</b> at the Primary or the transaction log is full.   | STANDALONE   | Unchanged  |
| PRIMARY ALONE                   | <b>ADMIN COMMAND 'hotstandby set secondary alone'</b> or <b>ADMIN COMMAND 'hotstandby switch secondary'</b> at the Primary.   | SECONDARY ALONE                                    | SECONDARY ALONE                                  |
| PRIMARY UNCERTAIN               | <b>ADMIN COMMAND 'hotstandby set primary alone'</b> at the Primary server   | PRIMARY ALONE                                      | Unchanged  |
| PRIMARY UNCERTAIN               | <p><b>ADMIN COMMAND 'hotstandby connect'</b> at the Primary.</p> <p> <b>Note</b></p> <p>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.</p> | PRIMARY ACTIVE                                     | Unchanged  |
| PRIMARY UNCERTAIN (HSB timeout) | <b>ADMIN COMMAND 'hotstandby set standalone'</b> at the Primary   | STANDALONE   | Unchanged  |

## D.1 HotStandby State Transition Diagram

| 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                             | <b>ADMIN COMMAND 'hotstandby set secondary alone'</b> or <b>ADMIN COMMAND 'hotstandby switch secondary'</b> at the Primary.  | SECONDARY ALONE              | Unchanged  |
| SECONDARY ACTIVE                              | HotStandby timeout (automatic)<br><br> <b>Note</b><br><br>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                              | <b>ADMIN COMMAND 'hotstandby switch secondary'</b> at the Primary or <b>ADMIN COMMAND 'hotstandby switch primary'</b> at the Secondary.  | PRIMARY ACTIVE               | Unchanged  |
| SECONDARY ACTIVE                              | <b>ADMIN COMMAND 'hotstandby set primary alone'</b> at the Secondary.  | PRIMARY ALONE                | Unchanged  |
| SECONDARY ACTIVE                              | <b>ADMIN COMMAND 'hotstandby disconnect'</b> at the Secondary or Primary.  | SECONDARY ALONE              | SECONDARY ALONE                                  |
| SECONDARY ALONE                               | <b>ADMIN COMMAND 'hotstandby connect'</b> at the Secondary or Primary  | SECONDARY ACTIVE             | Unchanged  |
| SECONDARY ALONE                               | <b>ADMIN COMMAND 'hotstandby set standalone'</b> at the Secondary.   | STANDALONE                   | Unchanged  |
| SECONDARY ALONE                               | <b>ADMIN COMMAND 'hotstandby set primary alone'</b> or <b>ADMIN COMMAND 'hotstandby switch primary'</b> 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 IBM 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 IBM 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 *IBM solidDB SQL Guide*, and Appendix B, IBM solidDB SQL Syntax, also in *IBM 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 | <i>ENAME</i> <i>WVARCHAR</i> , <i>POSTSR-<br/>VTIME</i> <i>TIMESTAMP</i> , <i>UID</i> <i>IN-<br/>TEGER</i> , <i>NUMDATAINFO</i> <i>IN-<br/>TEGER</i> , <i>TEXTDATA</i> <i>WVARCHAR</i><br><br>For TEXTDATA, the possible valid values are<br><br>TEXTDATA = {<br>CONNECTED  <br>CONNECTING  <br>CATCHUP  <br>BROKEN} | Change in connect status between the Primary and Secondary server |
| SYS_EVENT_HSB-STATESWITCH   | <i>ENAME</i> <i>WVARCHAR</i> , <i>POSTSR-<br/>VTIME</i> <i>TIMESTAMP</i> , <i>UID</i> <i>IN-</i>   | Each state switch sends a state switch event.                     |

| HSB Event            | Event Parameters  | Cause of Event   |
|----------------------|---|--|
|                      | <p><i>TEGER, NUMDATAINFO INTEGER, TEXTDATA WVARCHAR</i></p> <p>For TEXTDATA, the possible valid values are</p> <pre> TEXTDATA = { PRIMARY ACTIVE   PRIMARY ALONE   PRIMARY UNCERTAIN   SECONDARY ACTIVE   SECONDARY ALONE   STANDALONE } </pre> |  |
| SYS_EVENT_NETCOPYEND | <p><i>ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER, TEXTDATA WVARCHAR</i></p> <p>None of the parameters are used.</p>  | <p>HotStandby NETCOPY operation ended.</p> <p>This event can be caught by the user <i>only</i> if the user is using linked library access.</p>   |
| SYS_EVENT_NETCOPYREQ | <p><i>ENAME WVARCHAR, POSTSRVTIME TIMESTAMP, UID INTEGER, NUMDATAINFO INTEGER, TEXTDATA WVARCHAR</i></p> <p>None of the parameters are used.</p>  | <p>A HotStandby NETCOPY was requested.</p> <p>If the user application's callback function returns non-zero, then netcopy is not performed.</p> <p>This event can be caught by the user <i>only</i> if the user is using linked library access.</p> |

---

# Appendix F. Watchdog Sample

This appendix discusses the Watchdog sample application available in the samples included in your IBM solidDB installation.

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

IBM solidDB provides a sample watchdog that you can use as a basis for building a custom watchdog that meets your needs. This sample application is called the Watchdog. Pay attention to the following features of the Watchdog sample before you start programming.

- The Watchdog is meant to be used as an example of a watchdog
- The Watchdog uses polling to keep itself up-to-date with server states
- The Watchdog is a one-thread program
- The Watchdog uses the HSB API through ODBC. This API implementation can be used as a model for your own watchdog application.
- The Watchdog has no user interface.

If you are using the Watchdog, you need to configure a *[WatchDog]* section in the IBM solidDB 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.

Furthermore, this appendix contains explanations of `solid.ini` configuration parameters that are specific to the Watchdog. These parameters are set in the *[WatchDog]* section of the `solid.ini` configuration file. If you write your own watchdog program, you do not need to use any of these parameters.

For a discussion of other `solid.ini` parameters, see *IBM solidDB Administration Guide*.

## F.1 HotStandby Configuration Using Watchdog

A HotStandby configuration allows for a Primary server, Secondary server, and the Watchdog to reside in different machines and use different operating systems and APIs as shown in the example in Figure F.1,

“Heterogeneous HotStandby Configuration with Watchdog”. For details on implementing heterogeneous configurations, read Section F.1.2, “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.

### Important

If you are running the Watchdog 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 4.2.2, “Network Partitions and Dual Primaries”.

## F.1.1 How the Watchdog Application Works

The Watchdog sample 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 (the `[Watchdog]` section in the *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.

### Failure Mode

When the Watchdog sample application 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, the Watchdog will take some course of action. If the action fails to connect the servers, the Watchdog goes into failure mode.

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 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 the section called “Coding the Watchdog for Multiple Failures”.

During failure mode, the Watchdog 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 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.

### Coding the Watchdog for Multiple Failures

There are two ways to handle multiple failures in the Watchdog. 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 Watchdog 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 F.1. Dual Primaries**

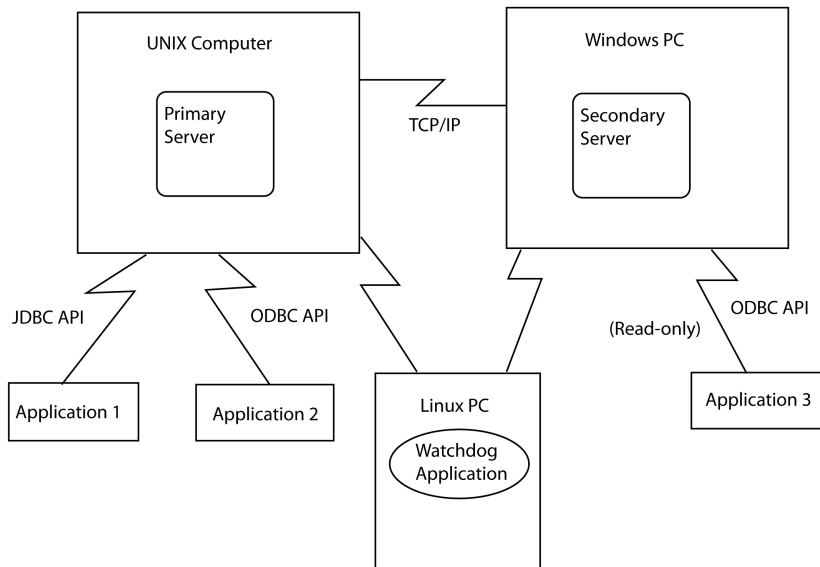
In this example, Server1 is initially the Primary and Server2 is initially the Secondary.

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.

## **F.1.2 System Design Issues**

How you configure HotStandby (locally or remotely, at one or more different locations, over the Internet, and with the 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 F.1. Heterogeneous HotStandby Configuration with Watchdog**

## F.1.3 Watchdog Configuration

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

If only two machines are available, making it impossible to run the 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 the Watchdog does not allow both servers to be put in the PRIMARY ALONE or STANDALONE state at the same time. See Section 4.2.2, "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 cannot 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.

## F.1.4 Using the Sample Watchdog Application

Initially, you should start the Watchdog after both servers are up and connected. To start the 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.



## **F.2 Failure Situations 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.

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 the Watchdog sample.

### **F.2.1 Primary is Down**

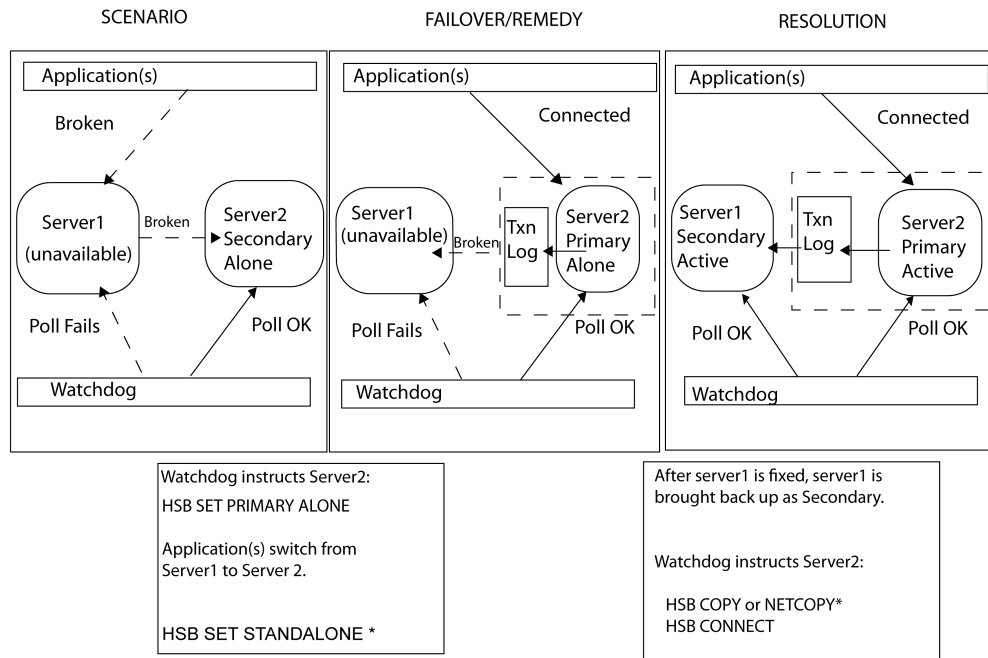
#### **Scenario**

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 F.2. Primary is Down Scenario and Remedy**



\* 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

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 6.3, “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.

## F.2.2 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.

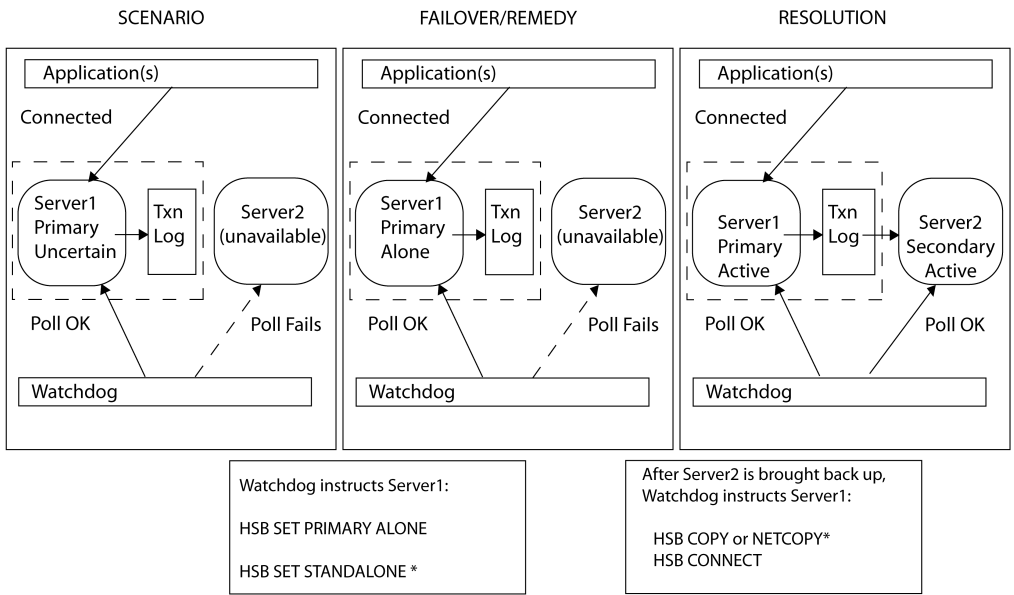
The Primary commits the open transactions after the Primary is set to PRIMARY ALONE state. To avoid the possibility that the Primary will commit the transactions when the Secondary hasn't, the transactions are kept in the transaction log, as though they had never been sent to the Secondary. When the 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 completely 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 F.3. Secondary is Down Scenario and Remedy**



\* 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 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 6.3, “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 F.2.1, “Primary is Down”.

## F.2.3 Watchdog Is Down

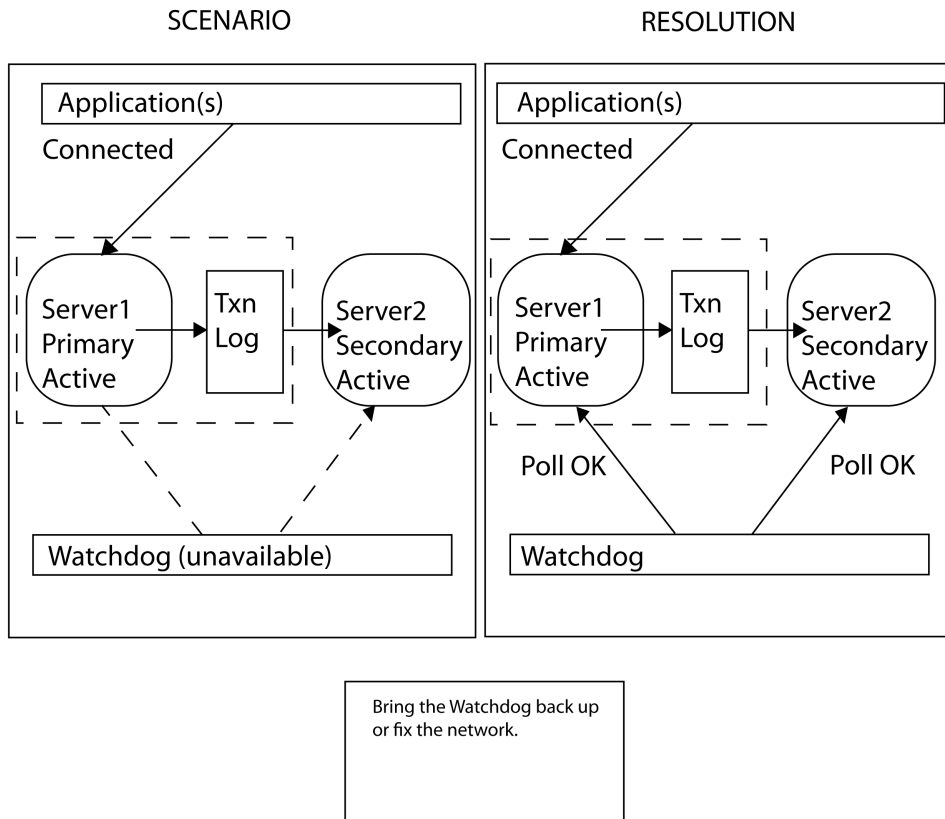
This chapter explains what happens if the Watchdog fails.

### 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 F.4. 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.

## F.2.4 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.

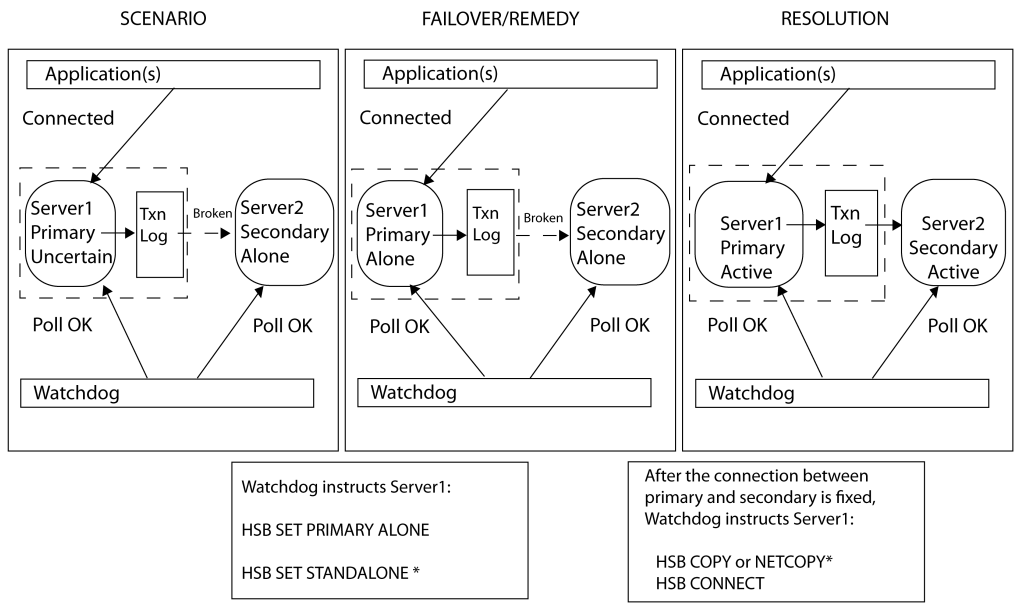
Until the Primary server receives a command from the Watchdog or the administrator, it no longer accepts transactions. At this stage, in order for the Primary server to continue operations, the Watchdog or administrator can set the Primary server to PRIMARY ALONE state.

### 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 F.5. Broken Link Between Primary and Secondary Scenario and Remedy**



\* 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 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 6.3, “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 F.2.1, “Primary is Down”.

## **F.2.5 Communication Link Between Watchdog and Primary Is Down**

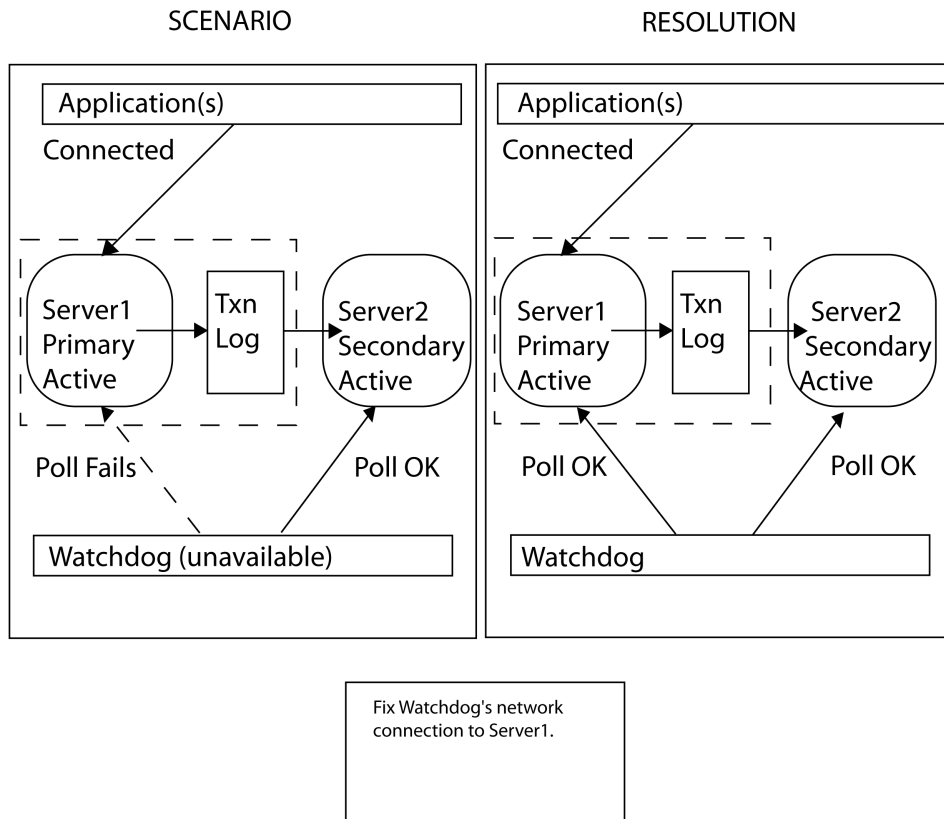
### **Scenario**

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 F.6. 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.

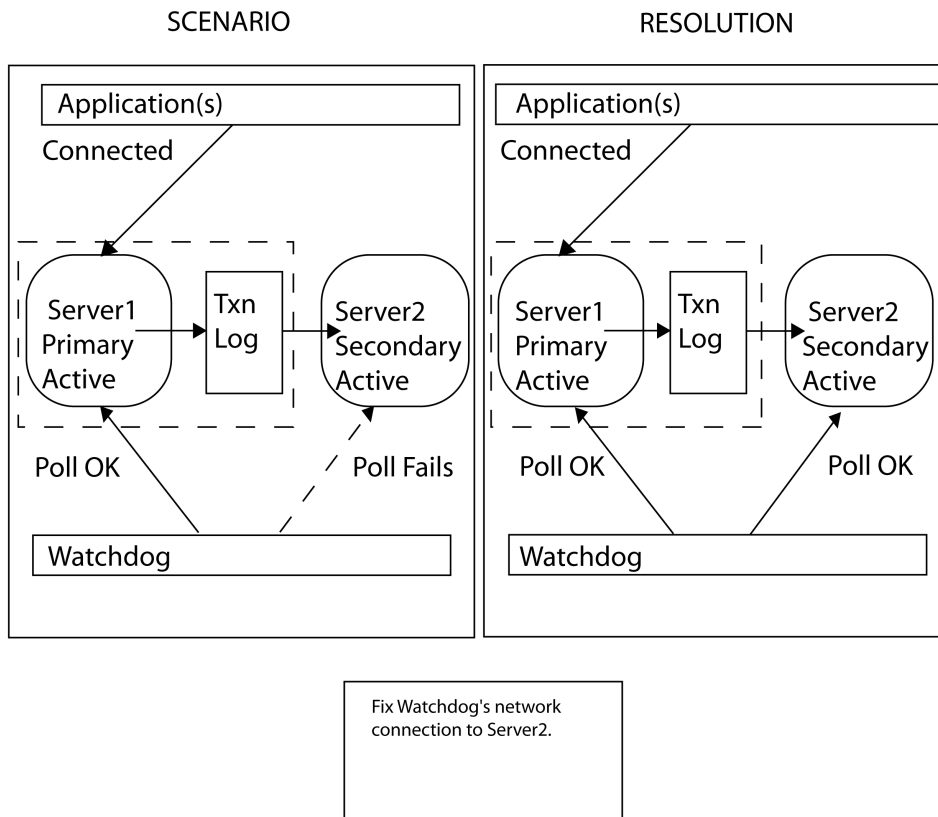
## **F.2.6 Communication Link Between Watchdog and Secondary Is Down**

### **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 F.7. 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.

## **F.2.7 Communication Links Between Watchdog and Primary, and Between Primary and Secondary, Are Down**

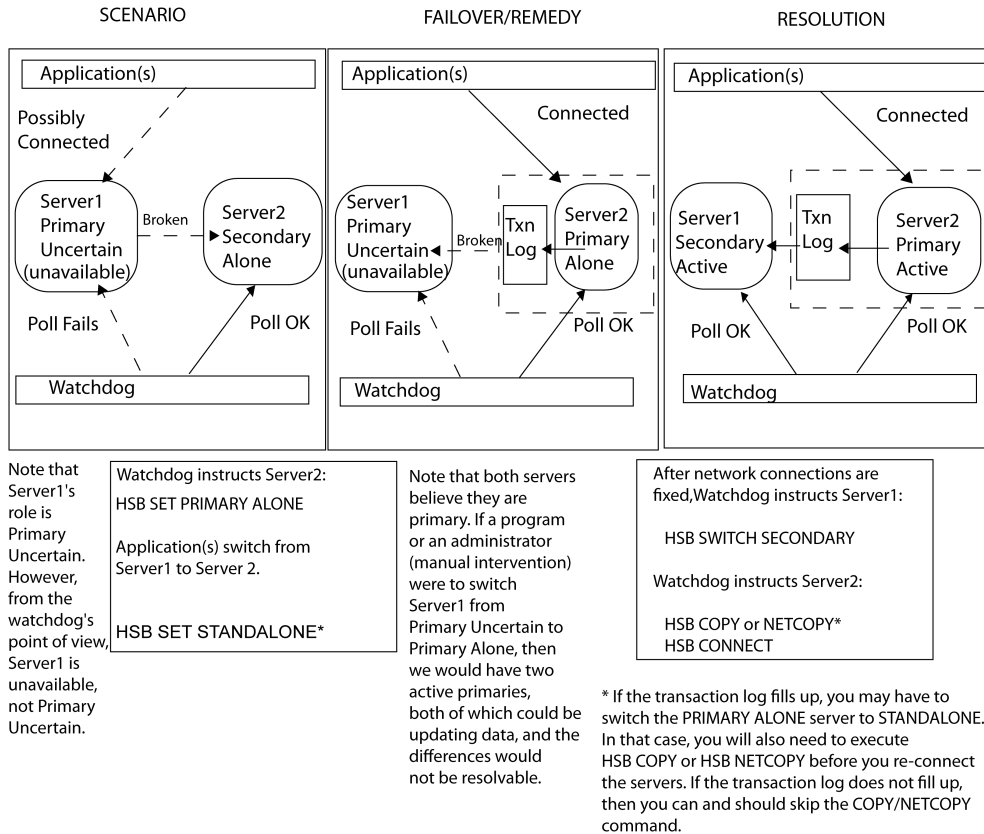
### **Scenario**

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 F.8. Broken Link Between Watchdog and Primary, and between Primary and Secondary, Scenario and Remedy**



## 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 6.3, “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.



## **F.2.8 Communication Links Between Watchdog and Secondary, and Between Primary and Secondary, Are Down**

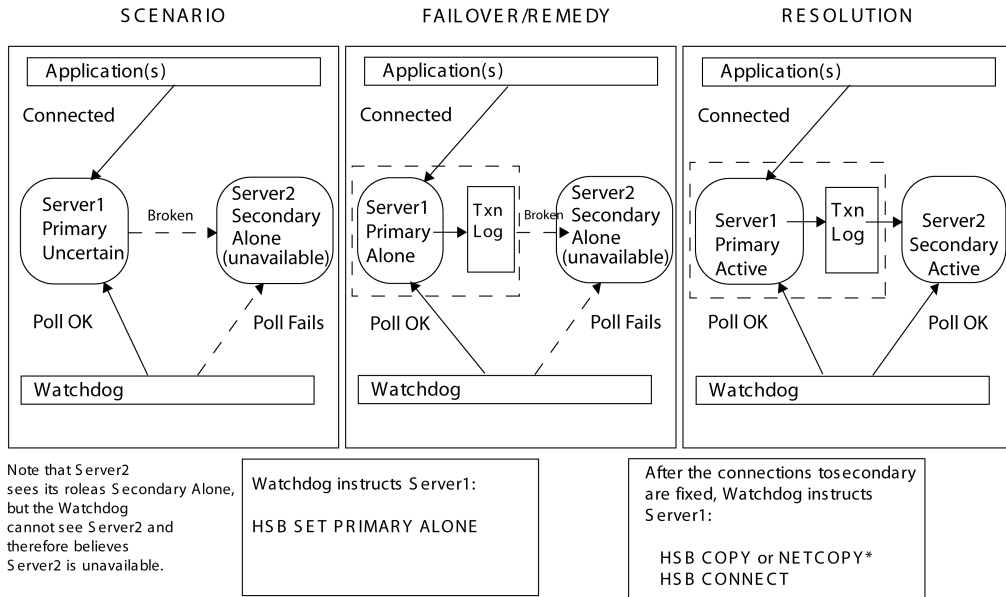
### **Scenario**

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 F.9. Broken Link between Watchdog and Secondary and between Primary and Secondary Scenario and Remedy**



\* 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 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 6.3, “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.

## F.3 Watchdog Section of solid.ini Configuration File

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



### Important

The parameters in the `[Watchdog]` section of the `solid.ini` file are NOT all pre-defined by IBM solidDB. Depending upon how you write your Watchdog and whether you want it to read parameter information from the `solid.ini` file, you can use any mix of the parameters defined here and parameters that you have defined. You can also ignore parameters. The parameters shown here are for the sample C-language Watchdog program that IBM solidDB provides.




### Note


Note that in IBM solidDB's sample Watchdog program, parameter names are not case-sensitive.

**Table F.1. Watchdog Parameters**

| <i>[Watchdog]</i> | Description   | Factory Value |
|-------------------|---|---------------|
| <i>AutoSwitch</i> | <p>If the <i>AutoSwitch</i> parameter is set to yes, the Watchdog automatically does the following:</p> <ol style="list-style-type: none"> <li>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.</li> <li>2) If the Primary server fails, then the Watchdog automatically sends the commands</li> </ol> <pre>'hsb switch primary' 'hsb set primary alone'</pre> <p>to switch the original Secondary to be the new Primary. For example:</p> <pre>[Watchdog] AutoSwitch = NO</pre> <p>This parameter is optional.</p> | Yes           |
| <i>Connect1</i>   | <p>The <i>Connect1</i> parameter in the <i>[Watchdog]</i> 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 <i>Connect1</i> server.</p> <p>For example:</p> <pre>connect1 = tcp primarymachine 1313</pre>  | None          |
| <i>Connect2</i>   | <p>The <i>Connect2</i> parameter in the <i>[Watchdog]</i> section enables the Watchdog application to connect to the Primary or Secondary server. This is a required parameter</p>  | None          |

| <b>[Watchdog]</b>                    | <b>Description</b>  | <b>Factory Value</b> |
|--------------------------------------|---|----------------------|
|                                      | <p>that defines the protocol and network address for the <i>Connect2</i> server.</p> <p>For example:</p> <pre>connect2 = tcp secondarymachine 1313</pre>  |                      |
| <i>DualSecAutoSwitch</i>             | <p>If <i>DualSecAutoSwitch</i> = <i>Yes</i> 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 <i>DualSecAutoSwitch</i> = <i>No</i> then the system administrator must switch one server to be the primary. Note that <i>DualSecAutoSwitch</i> applies whether the Watchdog is in "normal" mode or "failure" mode.</p>   | Yes                  |
| <i>NumRetry</i>                      | <p>The <i>NumRetry</i> parameter in the <i>[Watchdog]</i> 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:</p> <pre>[Watchdog] NumRetry = 3</pre> <p>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 <i>PingTimeout</i>) in between retries when there is a failure.</p> <p>This parameter is optional.</p> | 0                    |
| <i>Password1</i><br><i>Password2</i> | <p>See the description of the <i>Username1</i> and <i>Username2</i> parameters below.</p>   | No factory value.    |

| <b>[Watchdog]</b>   | <b>Description</b>   | <b>Factory Value</b> |
|---------------------|--|----------------------|
| <i>Pessimistic</i>  | <p>Setting this parameter to Yes can speed up Watchdog reactions.</p> <p>When <i>Pessimistic</i> = <i>No</i>, 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).</p> <p>When <i>Pessimistic</i> = <i>Yes</i>, the Watchdog acts as soon as the Watchdog itself loses contact with one of the 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.</p> <p>When <i>Pessimistic</i> = <i>Yes</i>, 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.</p> <p> <b>Caution</b></p> <p>Setting <i>Pessimistic</i> = <i>Yes</i> 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.</p> <p>You can also turn on Pessimistic behavior by using the optional command-line switch "<b>-p</b>".</p> | No                   |
| <i>PingInterval</i> | <p>The <i>PingInterval</i> parameter in the <i>[Watchdog]</i> 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 <i>PingInterval</i> milliseconds. For example:</p>   | 1000<br>(1 second)   |

| <i>[Watchdog]</i>                            | Description   | Factory Value            |
|--|---|--------------------------|
|  | <p>[Watchdog]<br/>PingInterval = 5000</p> <p>This parameter is optional.</p> <p>Note that the <i>PingInterval</i> parameter for the Watchdog is different from the <i>PingTimeout</i> parameter for the servers.</p> <p> <b>Caution</b></p> <p>Previous sample Watchdogs required that the <i>PingInterval</i> be specified in seconds, not milliseconds. If you are using an older <i>solid.ini</i> file, you should update it.</p> |                          |
| <p><i>Username1</i><br/><i>Username2</i></p> | <p>The <i>Username</i> and <i>Password</i> parameters in the <i>[Watchdog]</i> section are optional. They specify the username and password that are authorized for using the connect1 server. For example:</p> <p>[Watchdog]<br/>Username1 = Tom<br/>Password1 = dr17xy<br/>Username2 = Jerry<br/>Password2 = M89tvt</p> <p>If (for security reasons) these parameters are not specified in the <i>solid.ini</i> configuration file, the Watchdog will prompt for them when the Watchdog is started.</p>             | <p>No factory value.</p> |
| <p><i>WatchdogLog</i></p>                    | <p>The <i>WatchdogLog</i> parameter in the <i>[Watchdog]</i> 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.</p>  | <p>Watchdog.log</p>      |

| <i>[Watchdog]</i> | <b>Description</b>   | <b>Factory Value</b> |
|-------------------|--|----------------------|
|                   | <p>For example:</p> <pre>[Watchdog] WatchdogLog = Watchdog.log</pre> <p>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).</p> <p>This parameter is optional.</p> |                      |

 **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.



---

# 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 IBM solidDB 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 IBM 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 IBM solidDB, see "Binary Data Types" on page A-4 and "Character Data Types" on page A-1 in *IBM 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 *IBM 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, IBM 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 IBM solidDB advanced replication 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 IBM solidDB Linked Library Access 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, IBM solidDB's diskless capability is not optimized to enhance performance. If you want to minimize I/O to maximize performance, we recommend that you use IBM solidDB main 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 operational continuity during a specified measurement period. In IBM solidDB "HotStandby" is the name of the technique used by the HotStandby component. See also "HotStandby".

### HotStandby (HSB)

The IBM solidDB HotStandby component 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. IBM solidDB 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 IBM solidDB 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.

IBM solidDB 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 2.1.4, “The Transaction Log and HotStandby”.

### Two-Safe Algorithm

See 2-Safe Algorithm

## U

### UPS

Uninterruptible Power Supply

---

# Index

## Symbols

- x autoconvert, 132
- x backupserver (command line option), 111
- x backupserver (command), 109
- x migratehsbg2, 132
- 1-Safe Algorithm
  - defined, 217
- 1SafeMaxDelay (parameter), 140
- 2-Safe Algorithm
  - defined, 217
- 2SafeAckPolicy (parameter), 140
- =
  - use of the equals sign when setting parameter values, 41

## A

- Access Mode, 139
  - RO (read-only), 139
  - RW (read-write), 139
  - RW/Create, 139
  - RW/Startup, 139
- access rights, 42
- Adaptive Durability, 20
- ADMIN COMMAND 'hotstandby cominfo'
  - viewing connect settings, 114
- ADMIN COMMAND 'hotstandby connect'
  - connecting HotStandby servers, 114
- ADMIN COMMAND 'hotstandby copy'
  - copying database contents, 113
- ADMIN COMMAND 'hotstandby netcopy'
  - copying database contents, 111, 112
  - copying to secondary, 109
- ADMIN COMMAND 'hotstandby set primary alone'
  - Running the server in PRIMARY ALONE state, 101
- ADMIN COMMAND 'hotstandby set standalone'
  - shutting off HotStandby operations, 103

- ADMIN COMMAND 'hotstandby state'
  - verifying server states, 118
- ADMIN COMMAND 'hotstandby status connect'
  - displaying connect status information, 116
- ADMIN COMMAND 'hotstandby status copy'
  - verifying a copy procedure, 108, 112
- ADMIN COMMAND 'hotstandby status switch'
  - verifying the switch process, 101
- ADMIN COMMAND 'hotstandby status'
  - querying HotStandby status, 115
- ADMIN COMMAND 'hotstandby switch primary'
  - switching server states, 98
- ADMIN COMMAND 'hotstandby switch secondary'
  - switching server states, 98
- ADMIN COMMANDs
  - list of available commands, 168
  - list of High Availability Controller commands, 177
  - list of HotStandby commands, 168
- administering HotStandby, 116, 117
  - switching server states, 98
- administrative commands, 167
- advanced replication
  - using HotStandby with, 11
- applications
  - switching to the new primary, 90
  - using Basic Connectivity, 87
  - using Transparent Connectivity, 72
- autoconvert, 132
- AutoPrimaryAlone (parameter), 59, 102, 141, 191
  - and 'hotstandby switch' command, 99
- AutoSwitch (parameter), 212
- Availability
  - administrative commands, 167, 177

## B

- backup, 10
- backup listening mode, 109
  - (see also netcopy listening mode)
- BackupBlockSize (parameter), 60
- BackupDeleteLog (parameter), 10
- Basic Connectivity, 71

---

## BLOB

defined, 217

bringing a secondary back online, 103

## C

catchup, 13, 104

CatchupSpeedRate (parameter), 60, 141

CatchupStepsToSkip (parameter), 60

CheckInterval (parameter), 146

checkpoint, 10

CheckpointDeleteLog (parameter), 10

CheckTimeout (parameter), 146

Choosing which server to make primary, 120

CLUSTER, 77

Communication protocol

defined, 218

configuration

Secondary and Primary node configuration, 52

timeouts between applications and servers, 52

Configuration

HotStandby using Watchdog, 187

the Watchdog, 191

Configuration parameters, 135

Configuring HotStandby

netcopy performance, 60

Connect (parameter), 42, 55, 121, 141

Connect [LocalDB] (parameter), 146

Connect [RemoteDB] (parameter), 147

connect settings

viewing, 114

Connect1 (parameter), 212

Connect2 (parameter), 212

connection switch, 71

in Transparent Connectivity, 83

connectivity

basic, 71

choosing connectivity type, 72

Transparent Failover, 71

ConnectTimeout (parameter), 56, 57, 141

CopyDirectory (parameter), 58, 142

copying

database contents, 111, 112, 113

primary to local secondary, 112

verifying procedure, 108, 112

copying database contents, 111, 112, 113

Copying Primary database to Secondary server over the network, 108

Creating a New Database for the Secondary Server, 109

current value, 138

## D

Data management tools, 43

Database

In-Memory Tables, 44

database

copying contents, 111, 112, 113

verifying a copy procedure, 112

DBPassword (parameter), 146

DBUsername (parameter), 146

default value, 138

Defining Primary Server Behavior During a Secondary Failure, 59

Determining Whether the Primary's Settings Take Precedence Over the Secondary's, 137

displaying communication information, 117

displaying connect status information, 116

displaying switch status information, 116

Dual Primaries, 44

DualSecAutoSwitch (parameter), 213

DurabilityLevel (parameter), 54

## E

EnableAutoNetcopy (parameter), 145

EnableDBProcessControl (parameter), 145

Ensuring that Primary and Secondary Parameter Values Are Coordinated, 135

equals sign

use of when setting parameter values, 41

ERE (see External Reference Entity)

EREIP (parameter), 147

Error Codes, 151

---

events, 185

External Reference Entity

    configuring, 147, 148

    defined, 26

## F

factory value, 138

failover, 12

failure mode

    the Watchdog application, 189

Failure Transparency, 72

    choosing connectivity type

        CONNECTION, 72

        NONE, 72

        SESSION, 73

## G

GUI (see High Availability Manager)

## H

HA Controller, 97

HA Manager parameters

    Header\_text, 148

    Server1\_host, 148

    Server1\_name, 148

    Server1\_pass, 148

    Server1\_port, 148

    Server1\_user, 148

    Server2\_host, 148

    Server2\_name, 148

    Server2\_pass, 148

    Server2\_port, 148

    Server2\_user, 148

    Window\_title, 149

HAC (see defined)

HAC commands, 177

HAC failure scenarios

    communication link between primary and secondary is down, 200

    communication link between Watchdog and primary is down, 202

    communication link between Watchdog and secondary is down, 204

    communication links between Watchdog and primary, and between primary and secondary, are down, 206

    communication links between Watchdog and secondary, and between primary and secondary, are down, 209

    HotStandby link fails, 127

    primary database fails, 123

    primary node fails, 125

    secondary database fails, 124

    secondary node fails, 126

HAC parameters

    HAController, 61, 63, 64

Header\_text (parameter), 148

High Availability Control

    principles, 97

High Availability Controller, 9

    behaviour in failure cases, 123

    configuration, 34

    configuring, 61

    defined, 24

    logging, 30

    network considerations, 28

    sample, 31

    setup, 34

    solidhac.ini, 34

    starting, 34

    stopping, 34

High Availability Controller commands, 177

High Availability Manager

    configuring, 61

    defined, 28

    screenshot, 28

hotstandby copy, 182

HotStandby

    administering, 41

    administrative commands, 168

    configuration, 32

    configuring, 41, 51, 61

    defined, 219

---

- quick start, 32
- setup, 32
- shutting off operations, 103
- turning off, 121
- upgrading your server, 129
- HotStandby events
  - SYS\_EVENT\_HSBCONNECTSTATUS, 185
  - SYS\_EVENT\_HSBSTATESWITCH, 185
  - SYS\_EVENT\_NETCOPYEND, 186
  - SYS\_EVENT\_NETCOPYREQ, 186
- hotstandby netcopy, 182
- HotStandby recovery model
  - sending data, 15
  - synchronous hot standby, 14, 17
- HotStandby Status, 115
- HotStandby using the Watchdog
  - Configuration, 190
  - System Design Issues, 190
- HotStandby using Watchdog
  - Configuration, 187
- HotStandby with HAC
  - configuration, 35
  - quick start, 35
  - setup, 35
- HOTSTANDBY\_CONNECTSTATUS
  - SQL function, 116
- HOTSTANDBY\_CONNECTSTATUS (SQL function), 90
- HOTSTANDBY\_STATE (SQL function), 91
- HSB
  - defined, 219
- hsb status catchup, 175
- hsb status connect, 175
- hsb status copy, 176
- hsb status switch, 176
- HSBEnabled (parameter), 42, 55, 121, 143

## I

- IBM solidDB Data Dictionary
  - defined, 43
- IBM solidDB data management tools, 43

- IBM solidDB Export
  - defined, 43
- IBM solidDB SpeedLoader
  - defined, 43
- In-Memory Tables, 44
- installation, 31

## L

- Listen (parameter), 52, 145
- load balancing
  - dynamic control of, 74
  - with Transparent Connectivity, 73
- load balancing methods
  - PREFERRED\_ACCESS=READ\_MOSTLY, 73
  - PREFERRED\_ACCESS=WRITE\_MOSTLY, 73
- LogEnabled (parameter), 55
- Logging (see High Availability Controller)
- logpos (admin command), 169
- logpos (hotstandby command), 120

## M

- MaxLogSize (parameter), 143
- MaxMemLogSize (parameter), 143
- migratehsbg2, 132

## N

- netcopy, 60, 182
  - (see also netcopy listening mode)
  - Primary must be in PRIMARY ALONE state, 17
- netcopy listening mode, 109, 111
  - ADMIN COMMAND 'hotstandby netcopy', 109
  - tuning performance, 60
- NetcopyRpcTimeout (parameter), 143
- Network name
  - defined, 220
- network names, 52
- Network Partitions, 44
- Network Partitions and Dual Primaries, 44
- networking in HAC, 28
- NumRetry (parameter), 213



---

## O

OFFLINE (state), 182

One-Safe Algorithm  
defined, 217

## P

parameters

AutoPrimaryAlone, 59, 99, 102

AutoSwitch, 212

BackupBlockSize, 60

BackupDeleteLog, 10

CatchupSpeedRate, 60

CatchupStepsToSkip, 60

CheckInterval, 63, 146

CheckpointDeleteLog, 10

CheckTimeout, 62, 146

Connect, 42, 55, 63, 64, 121

Connect [LocalDB], 146

Connect [RemoteDB], 147

Connect1, 212

Connect2, 212

ConnectTimeout, 56, 57

CopyDirectory, 58

DBPassword, 63, 146

DBUsername, 63, 146

DualSecAutoSwitch, 213

DurabilityLevel, 54

EnableAutoNetcopy, 62, 145

EnableDBProcessControl, 62, 145

EREIP, 64, 147

Header\_text, 148

High Availability Controller

CheckInterval, 146

CheckTimeout, 146

Connect [LocalDB], 146

Connect [RemoteDB], 147

DBPassword, 146

DBUsername, 146

EnableAutoNetcopy, 145

EnableDBProcessControl, 145

EREIP, 147

Listen, 145

Password, 146

PreferredPrimary, 147

RequiredConnectFailures, 145

RequiredPingFailures, 148

StartInAutomaticMode, 145

StartScript, 146

Username, 146

HSBEnabled, 42, 55, 121

Listen, 52, 61

LogEnabled, 55

NumRetry, 213

Password, 63, 146

Password1, 213

Password2, 213

Pessimistic, 214

PingInterval, 56, 57, 214

PingTimeout, 56, 57

PreferredPrimary, 64, 147

ReadMostlyLoadPercentAtPrimary, 73

RequiredConnectFailures, 62, 145

RequiredPingFailures, 64, 148

SafenessLevel, 14

Server1\_host, 148

Server1\_name, 148

Server1\_pass, 148

Server1\_port, 148

Server1\_user, 148

Server2\_host, 148

Server2\_name, 148

Server2\_pass, 148

Server2\_port, 148

Server2\_user, 148

StartInAutomaticMode, 62, 145

StartScript, 63, 146

Username, 63, 146

Username1, 215

Username2, 215

WatchdogLog, 215

Window\_title, 149

Partition

network, 44

---

Password (parameter), 146  
Password1 (parameter), 213  
Password2 (parameter), 213  
Performing Read-Only Transactions on the Secondary,  
22  
performing recovery and maintenance, 47  
Pessimistic (parameter), 214  
ping, 57  
PingInterval (parameter), 56, 57, 143, 214  
PingTimeout (parameter), 56, 57, 144  
PreferredPrimary (parameter), 147  
PRIMARY ACTIVE (state), 182  
PRIMARY ALONE, 101  
PRIMARY ALONE (state), 182, 183  
PRIMARY ALONE state  
    running the new Primary in PRIMARY ALONE  
    state, 101  
PRIMARY UNCERTAIN (state), 182  
PrimaryAlone (parameter), 144

## R

READ COMMITTED  
    transaction isolation level, 44  
Read-Only Transactions on the Secondary, 22  
ReadMostlyLoadPercentAtPrimary (parameter), 73,  
139  
Reconnecting to Primary Servers from Applications,  
87  
recovery  
    maintenance, 47  
REPEATABLE READ  
    transaction isolation level, 44  
RequiredConnectFailures (parameter), 145  
RequiredPingFailures (parameter), 148  
rights  
    access, 42  
RO  
    Access Mode, 139  
Running out of space for transaction log, 45  
Running the server in PRIMARY ALONE state, 101  
RW

    Access Mode, 139  
RW/Create  
    Access Mode, 139  
RW/Startup  
    Access Mode, 139

## S

SafenessLevel (parameter), 14  
Samples  
    High Availability Controller, 31  
    Watchdog, 31  
secondary server  
    bringing back online, 103  
SERIALIZABLE  
    transaction isolation level, 44  
Server state  
    OFFLINE, 19  
    PRIMARY ACTIVE, 17  
    PRIMARY ALONE, 18, 101  
    PRIMARY UNCERTAIN, 18  
    SECONDARY ACTIVE, 19  
    SECONDARY ALONE, 19  
    STANDALONE, 19  
server state transitions, 179  
Server states  
    described, 17  
server states  
    switching server states, 98  
    verifying, 118  
Server1\_host (parameter), 148  
Server1\_name (parameter), 148  
Server1\_pass (parameter), 148  
Server1\_port (parameter), 148  
Server1\_user (parameter), 148  
Server2\_host (parameter), 148  
Server2\_name (parameter), 148  
Server2\_pass (parameter), 148  
Server2\_port (parameter), 148  
Server2\_user (parameter), 148  
servers  
    connecting, 114

---

SET TRANSACTION WRITE, 74  
SET WRITE, 74  
sever catchup, 13  
    (see also catchup)  
sever names, 52  
    (see also network names)  
Shutting Off HotStandby Operations, 103  
solidhac.ini, 34  
    ERE section, 64  
    HAController section, 61  
    LocalDB section, 63  
    RemoteDB section, 64  
SQL functions  
    HOTSTANDBY\_CONNECTSTATUS, 90, 116  
    HOTSTANDBY\_STATE, 91  
STANDALONE (state), 103, 121, 182  
start-up sequence, 37  
StartInAutomaticMode (parameter), 145  
StartScript (parameter), 146  
state  
    OFFLINE, 182  
    PRIMARY ACTIVE, 182  
    PRIMARY ALONE, 182, 183  
    PRIMARY UNCERTAIN, 182  
    STANDALONE, 182  
State switch  
    Secondary switch to new Primary, 12  
state switch  
    verifying, 101  
states  
    STANDALONE, 103, 121  
    transitions, 179  
    verifying server states, 118  
status, 115  
    displaying communication information, 117  
    displaying connect status information, 116  
    displaying switch status information, 116  
    HotStandby, 115  
    list of, 81, 83, 116  
Store Mode, 139  
switching  
    displaying connect status information, 116

    displaying switch status information, 116  
    switching states  
        verifying, 101  
synchronization  
    using HotStandby with advanced replication, 11  
synchronizing Primary and Secondary servers, 104  
SYS\_EVENT\_HSBCONNECTSTATUS (event), 185  
SYS\_EVENT\_HSBSTATESWITCH (event), 185  
SYS\_EVENT\_NETCOPYEND (event), 186  
SYS\_EVENT\_NETCOPYREQ (event), 186

## T

TC Connection, 71  
TC Info, 76  
    attribute combinations, 79  
    handling contradictions, 80  
    JDBC syntax, 80  
    proprietary ODBC attributes, 86  
    syntax, 76  
TF Connectivity, 71  
The Transaction Log and HotStandby, 9  
the Watchdog  
    using, 192  
the Watchdog application  
    failure mode, 189  
the Watchdog sample  
    described, 188  
Tools  
    IBM solidDB data management tools, 43  
transaction isolation level  
    in-memory tables, 44  
transaction log, 9, 15  
    running out of space for, 45  
Transaction log file  
    defined, 220  
transitions  
    server state transitions, 179  
Transparent Connectivity, 71  
Two-Safe Algorithm  
    defined, 217

---

## U

- upgrading your server, 129
  - cold and hot migration, 129
  - cold migration, 129
    - procedure, 131
  - hot migration, 129
    - procedure, 131
  - migration between hsb-compatible versions, 129
  - migration between hsb-incompatible versions, 130
  - preparing, 130
- UPS
  - defined, 220
- Username (parameter), 146
- Username1 (parameter), 215
- Username2 (parameter), 215

## V

- verifying a copy procedure, 112
- verifying connect status information
  - HOTSTANDBY\_CONNECTSTATUS, 90
- verifying the state switch, 101
- verifying the switch process
  - ADMIN COMMAND 'hotstandby status switch', 101
- viewing current connect settings, 114

## W

- Watchdog
  - configuration, 191
  - failure, 198
    - Watchdog is down, 198
- watchdog failure scenarios
  - primary is down, 193
  - secondary is down, 195
- Watchdog sample, 31, 187
- WatchdogLog (parameter), 215
- Window\_title (parameter), 149