solid.

www.solidDB.com

solidDB AcceleratorLib™ Guide

Version 6.0 | June 2007

solidDB 6.0

solidDB AcceleratorLib User Guide

Document number: SAC-6.0
Product version: 06.00.1018
Date: 19.06.2007

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

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

Table of Contents

List of Figures

List of Tables

List of Examples

Chapter 1. Welcome

The solidDB Accelerator Library ("AcceleratorLib") is a higher performance version of Solid's data management solution. To avoid network delays, the solidDB executable and the user application are linked in the same program space to produce a single executable. By replacing the network connection and Remote Procedure Calls (RPCs) with local function calls, performance is improved significantly.

1.1 About This Guide

This guide contains information specific to the AcceleratorLib.

This guide supplements the information contained in the *solidDB Administration Guide*, which contains details on administration and maintenance of solidDB.

1.1.1 Organization

AcceleratorLib has some usage differences from standard solidDB. This manual highlights the main areas of difference and includes the following chapters:

- Chapter 2, *Introducing AcceleratorLib*, familiarizes you with the background, concepts, components, and physical configuration options for using the feature.
- Chapter 3, *Creating and Running an AcceleratorLib Application*, covers the steps required to implement AcceleratorLib at your solidDB site.
- Chapter 4, *Description of Control API*, describes how to use Solid SSC API, a low level C-language client library, to start and stop the server, and to perform operations such as setting the priority level for various server internal tasks. The AcceleratorLib supports the use of Solid SSC API for local connections.
- Chapter 5, *Using the Diskless Capability*, explains how to use the `SSCStartDisklessServer()` function call in the AcceleratorLib library to start a server that will run without a disk drive.
- Chapter 6, *Using solidDB AcceleratorLib With Java*, describes how to use Solid AcceleratorLib from a Java program.
- Appendix A, *AcceleratorLib Parameters*, describes configuration parameters that are specific to the AcceleratorLib. These parameters, like other Solid configuration parameters, are set in the `solid.ini` file.

1.1.2 Audience

This guide assumes a working knowledge of the C programming language, general DBMS knowledge, familiarity with SQL, and knowledge of a Solid data management product, such as solidDB, Solid In-memory Engine, or Solid Disk-based Engine. If you are going to work with the Solid Java Accelerator, then this manual also assumes a working knowledge of Java.

1.2 Conventions

1.2.1 About solidDB

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

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

1.2.2 Typographic Conventions

This manual uses the following typographic conventions:

Table 1.1. Typographic Conventions

Format	Used for
Database table	This font is used for all ordinary text.
NOT NULL	Uppercase letters on this font indicate SQL keywords and macro names.
`solid.ini`	These fonts indicate file names and path expressions.
`SET SYNC MASTER YES;` `COMMIT WORK;`	This font is used for program code and program output. Example SQL statements also use this font.
run.sh	This font is used for sample command lines.
`TRIG_COUNT()`	This font is used for function names.
`java.sql.Connection`	This font is used for interface names.

Format	Used for
LockHashSize	This font is used for parameter names, function arguments, and Windows registry entries.
argument	Words emphasised like this indicate information that the user or the application must provide.
solidDB Administration Guide	This style is used for references to other documents, or chapters in the same document. New terms and emphasised issues are also written like this.
File path presentation	File paths are presented in the Unix format. The slash (/) character represents the installation root directory.
Operating systems	If documentation contains differences between operating systems, the Unix format is mentioned first. The Microsoft Windows format is mentioned in parentheses after the Unix format. Other operating systems are separately mentioned.

1.2.3 Syntax Notation

This manual uses the following syntax notation conventions:

Table 1.2. Syntax Notation Conventions

Format	Used for
`INSERT INTO` *table_name*	Syntax descriptions are on this font. Replaceable sections are on *this* font.
`solid.ini`	This font indicates file names and path expressions.
[ ]	Square brackets indicate optional items; if in bold text, brackets must be included in the syntax.
|	A vertical bar separates two mutually exclusive choices in a syntax line.
{ }	Curly brackets delimit a set of mutually exclusive choices in a syntax line; if in bold text, braces must be included in the syntax.
...	An ellipsis indicates that arguments can be repeated several times.

Format	Used for
. . .	A column of three dots indicates continuation of previous lines of code.

1.3 Solid Documentation

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

Electronic Documentation

- *Release Notes*. This file contains installation instructions and the most up-to-date information about the specific product version. This file (`releasenotes.txt`) is copied onto your system when you install the software.
- *solidDB Getting Started Guide*. This manual gives you an introduction to the solidDB.
- *solidDB SQL Guide*. This manual describes the SQL commands that solidDB supports. This manual also describes some of the system tables, system views, system stored procedures, etc. that the engine makes available to you. This manual contains some basic tutorial material on SQL for those readers who are not already familiar with SQL. Note that some specialized material is covered in other manuals. For example, the Solid "administrative commands" related to the High Availability (HotStandby) Option are described in the *solidDB High Availability User Guide*, not the solidDB SQL Guide.
- *solidDB Administration Guide*. This guide describes administrative procedures for solidDB servers. This manual includes configuration information. Note that some administrative commands use an SQL-like syntax and are documented in the *solidDB SQL Guide*.
- *solidDB Programmer Guide*. This guide explains in detail how to use features such as Solid Stored Procedure Language, triggers, events, and sequences. It also describes the interfaces (APIs and drivers) available for accessing solidDB and how to use them with a solidDB database.
- *solidDB In-Memory Database User Guide*. This manual describes how to use the in-memory database of solidDB In-memory Engine.
- *solidDB SmartFlow Data Replication Guide*. This guide describes how to use the Solid SmartFlow technology to synchronize data across multiple database servers.

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

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

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

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

Chapter 2. Introducing AcceleratorLib

The solidDB AcceleratorLib is a function library that provides the same functionality and interfaces available with the solidDB. A user application may be linked to this library. The linked application communicates with the server by using direct function calls, thus skipping the overhead required when the client and server communicate through network protocols such as TCP/IP. Linking the application and server into a single executable provides higher performance.

Your application does not have to be re-written to use the AcceleratorLib library. For example, you do not need to call proprietary functions (except a few to start and stop the database server). Instead, your application may continue to use the same ODBC function calls that it has always used. When the AcceleratorLib library is linked to your application, these ODBC function calls go directly to the server, bypassing the network.

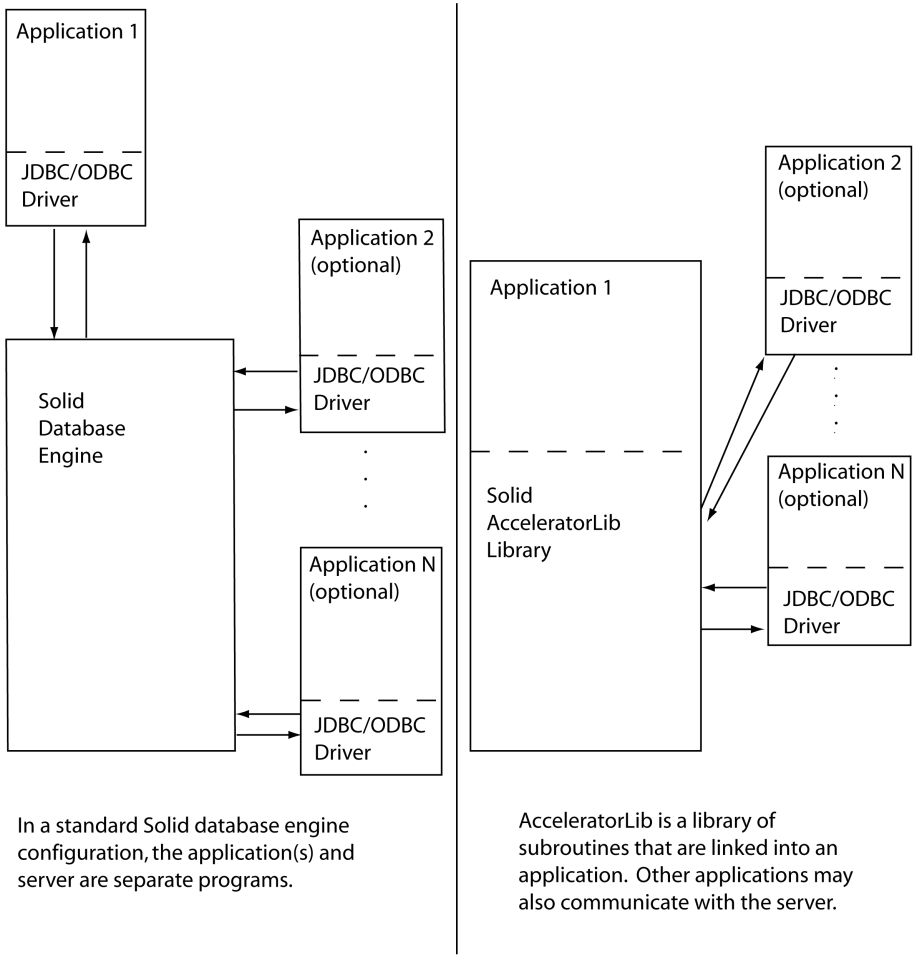
Your application also has access to some additional AcceleratorLib function calls to do things such as scheduling tasks within the server. However, you are not required to use these function calls unless you want to.

The fact that your server is linked to your application does not mean that your linked application is the only client that can use the server. A solidDB server that is executing as an AcceleratorLib function library is accessible not only to the "local" client application (the application that is linked directly to the library), but also to "remote" client applications (which connect to the server through communications protocols such as TCP/IP). Your remote clients see the AcceleratorLib server as similar to any other solidDB server, while your local client sees a faster, more precisely controllable version of the solidDB server.

Note

Although "remote" applications usually run on a different computer from the one that the server is running on, an application is also considered "remote" if it uses a network communication protocol to communicate with the server, even if that client runs on the same computer as the database server runs on.

Figure 2.1. solidDB with AcceleratorLib



The figure above shows a sample solidDB that uses the AcceleratorLib library.

Note

Local application requests are handled through Solid SA API or ODBC API direct function calls. For the local application, AcceleratorLib also provides a Control API which handles local requests for controlling Solid background processes and client tasks. You may also use JDBC calls with the AcceleratorLib. See Chapter 6, *Using solidDB AcceleratorLib With Java*.

As you can see in the illustration, remote clients communicate through an ODBC or JDBC driver that is linked to the client application, while the local client application does not need any remote communication driver.

2.1 AcceleratorLib Library

In a standard (non-AcceleratorLib) solidDB configuration, the application (the "client") and the database engine (the 'server") are separate processes that communicate through a network protocol. The client must link to a communications driver (such as an ODBC or JDBC driver) that communicates with the database server through the network.

With the AcceleratorLib, an application links to a static library (for example, .lib or .a for UNIX) that contains the full database server functionality. This means solidDB runs in the same executable with the application, eliminating the need to transfer data through the network. The application that links to the AcceleratorLib library can also have multiple connections, using both ODBC API and SA API. Both of these APIs are reentrant, allowing simultaneous connections from separate threads.

A user application that links directly to the AcceleratorLib library can also create remote connections to other database servers. The connect string that is passed to the ODBC API or SA API connect function defines whether the connection type is local or remote.

For details on linking an application, read Section 3.2, “Linking Applications for the AcceleratorLib”.

When you start your application, only the code in your application starts running automatically. The server code is largely independent of your application code, and you must explicitly start the server by calling a function. (In most or all implementations, the server runs on threads that are separate from the thread(s) used by the application. Calling the function to start a server will perform any initialization steps required by the server code, create the appropriate additional threads if necessary, and start the server running on those threads.)

2.1.1 Disk-based vs. Diskless Servers

The accelerator library contains two different function calls to start the server. One of the function calls starts a normal (that is, disk-based) server, while the other starts a server that does not use the disk drive. For more information, see Chapter 5, *Using the Diskless Capability* and the descriptions of the `SSCStartServer` and `SSCStartDisklessServer` functions.

2.1.2 Library Contents

The accelerator library includes functions for three separate APIs:

- Solid Control API (SSC API) library that contains functions to control task scheduling.

- Solid ODBC Driver functions that allows for direct communication with the server library, without going through the network.

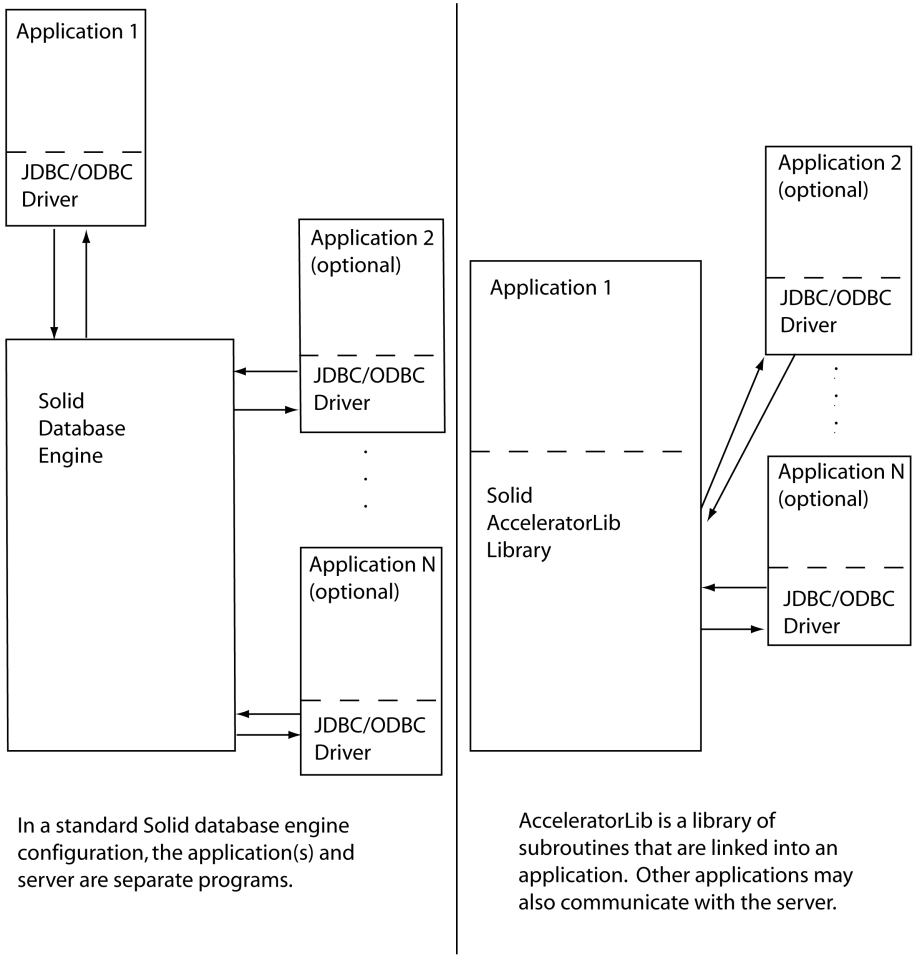
- Solid SA API library which may be required for additional functionality using the AcceleratorLib. For example, this library allows you to insert, delete, and select records from a table.

Because your application gets linked to a library with all three of these APIs (SSC, SA, and ODBC), your application program may call functions from any combination of these APIs. For details on each of these APIs, read Section 2.2, “solidDB Client APIs and Drivers for the AcceleratorLib”.

Note

Remote applications have access to the same three APIs (SSC, SA, and ODBC). However, the functions for these three APIs are not all in the same file for remote applications. For details on remote and dual role applications, read Section 2.1.3, “Application Types Used with AcceleratorLib”. For information on API files for remote applications, read Section 2.2, “solidDB Client APIs and Drivers for the AcceleratorLib”.

Figure 2.2. Linking to solidDB



2.1.3 Application Types Used with AcceleratorLib

The AcceleratorLib application is "local" to the server; the server and the application are combined into a single program. Calls to ODBC functions actually go directly to the server, rather than going through an ODBC driver and the communications protocol (such as TCP/IP).

In addition to handling requests from the local application that is linked to the AcceleratorLib library, the server also handles requests from remote applications.

A remote application is not linked to the AcceleratorLib library. It is a separate executable that must communicate with the server using a network connection (such as TCP/IP) or other connection (for example shared memory). Remote applications are usually, but not always, run on a different computer from the one that is running the server. However, a single computer can run an AcceleratorLib local application, while running one or more remote applications as separate processes.

Most applications are either local (that is linked to the AcceleratorLib library in a single executable) or remote (never linked to the AcceleratorLib library). However, it is also possible to write an application that can be either local and remote; it switches modes, depending upon how it is compiled and linked. Such a *dual mode application* uses, for example, the same C-language application code in either local or remote mode; but it is linked to a different library when in local mode than when in remote mode.

Using Dual-Mode Applications with the AcceleratorLib

In the case of AcceleratorLib, for example, a dual mode application must be linked to the local AcceleratorLib library when it is run locally. However, when it is run remotely, the dual mode application must be linked to the Accelerator Control API stub library (for example, `solidctrlstub.lib` in Windows), so that it can be compiled, linked, and executed without link-time errors.

The "Control API stub library" is required for remote applications because the AcceleratorLib's own Control API (which is provided in the local AcceleratorLib library) cannot be used with remote applications. For example, assume you have a local application (containing Control API functions) that links to a standard ODBC library. You want to run the same application remotely. By linking to the Control API stub library, you avoid having to remove the Control API function calls from your code. In this way, you can easily turn your AcceleratorLib local application into a normal remote client application.

Note

The Control API stub library contains "do-nothing" functions; if you call them in a remote application, they have no effect on the server.

A dual mode application is useful for other reasons as well:

- You may want to test your local application first before linking it with the AcceleratorLib library.
- You may want all users/processes to have the same application logic whether they are local or remote.

Example 2.1. Dual Mode Application

Assume there are two users who are running the same application. User1 runs the application locally (benefiting from higher performance). User2 runs the same application remotely.

User1 (local user) compiles and links with the server library (`solidac.a`, for example) and is responsible for starting and stopping the server and performing other scheduling tasks using the AcceleratorLib's Control API. User2 (remote user) runs the same application, but is not able to connect to the server until User1 has started the server. Thus, only User1 is able to control the tasking system.

2.2 solidDB Client APIs and Drivers for the AcceleratorLib

Below is a brief description of the APIs available for use with the AcceleratorLib.

Note

These descriptions use the term "local" and "remote" applications as defined in Section 2.1.3, “Application Types Used with AcceleratorLib”.

2.2.1 Solid SA API

SA API is a low-level proprietary C-language API to Solid data management services. It is included in the AcceleratorLib library (for example, `ssolidacxx.dll` for Windows or `solidac.a` for UNIX). The AcceleratorLib library includes the SA-API library that provides support for local applications using SA API function calls.

The SA API library is used internally in Solid products and provides access to data in solidDB database tables. The library contains 90 functions providing low-level mechanisms for connecting the database and running cursor-based operations. Solid SA API can enhance performance significantly. You can use SA API to optimize the performance of batch insert operations, for example.

For remote applications, the AcceleratorLib library also provides support for the SA API function calls. However, you must link to a separate SA API library file (for example, `solidimpsa.lib` for Windows).

For details on the Solid SA API, see *solidDB Programmer Guide*.

2.2.2 Solid ODBC API

Solid ODBC API provides a standards-compliant way to access data of a local or remote solidDB database through SQL. It provides functions for controlling database connections, executing SQL statements, retrieving result sets, committing transactions, and other data management functionality.

ODBC API, a Call Level Interface (CLI) for solidDB databases, is compliant with ANSI X3H2 SQL CLI, and is included in the AcceleratorLib library (for example, `ssolidacxx.dll` for Windows or `solidac.a` for UNIX).

AcceleratorLib supports the ODBC 3.51 standard. The AcceleratorLib library includes Solid ODBC 3.x, which provides support for local applications that require direct function calls to the server.

For local applications, the AcceleratorLib library provides support for ODBC function calls. For remote applications (or for a dual-mode application that is to be run remotely), you must link the ODBC Driver to get the same functionality.

If your application is a dual mode application (i.e. can be run either locally or remotely), and if it uses AcceleratorLib's Control API and ODBC, then you will need two different executables, one to be run locally and one to be run remotely. When you link your application to run it locally, you will link it to the AcceleratorLib library, which provides support for both the ODBC functions and the Control API library. When you link your application to run it remotely, you must link it to both the ODBC driver and to the Control API stub library (for example, `solidctrlstub.lib` for Windows). This stub library does not actually give your remote application any control over the server; it simply allows you to compile and link your program without getting errors about "unresolved symbols".

Note

When ODBC functions (in a dual mode application) are called remotely, then the calls go through the network from the client to the server. When ODBC functions are called locally (in accelerated applications), then the ODBC subroutine library bypasses the network and directly connects the local application to the server.

Read *solidDB Programmer Guide* for more details on ODBC API.

2.2.3 Solid JDBC API

JDBC API is used by remote applications only. As the core API for JDK 1.2, it defines Java classes to represent database connections, SQL statements, result sets, database metadata, etc. It allows you to issue SQL statements and process the results. JDBC is the primary API for database access in Java. AcceleratorLib supports both JDBC 1.x and 2.x. Read *solidDB Programmer Guide* for more details.

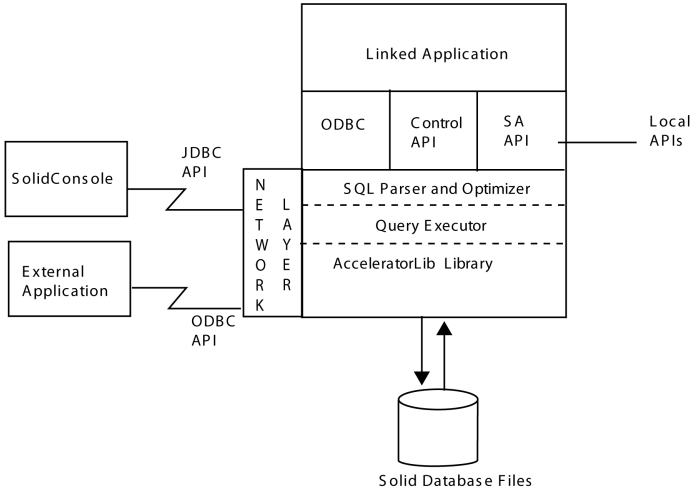
2.2.4 Solid Control API (SSC API)

Solid Control API (SSC API) is a C-language, thread-safe interface to control the server behavior in solidDB database products.

The Control API is included in the AcceleratorLib library (for example, `ssolidacxx.dll` for Windows or `solidac.a` for UNIX). The AcceleratorLib library provides support for local applications using Control API function calls and a separate library is available for remote-only applications.

If your application will run remotely and contains Control API function calls, then you must link the Control API Stub library (for example, `solidctrlstub.lib` for Windows). This library does not actually give your remote application control of the server; it merely allows you to compile and link your application as a remote application without getting link-time errors solidDB with AcceleratorLib.

Figure 2.3. solidDB with AccleratorLib - APIs



Chapter 3. Creating and Running an AcceleratorLib Application

This chapter describes how to create and run the AcceleratorLib application. It includes the following topics:

- Downloading the AcceleratorLib library
- Linking Applications to the AcceleratorLib library
- Creating or using an existing database
- Starting and stopping solidDB with the AcceleratorLib

Note

This chapter provides AcceleratorLib-specific additions, supplements, and AcceleratorLib usage differences from solidDB without the AcceleratorLib. For information on Solid SQL, Solid data management tools, general Solid administration and maintenance, and database error codes, refer to the *solidDB Administration Guide*. Read Chapter 4, *Description of Control API* and *solidDB Programmer Guide* for details on developing applications with an AcceleratorLib supported API.

3.1 Downloading the AcceleratorLib Library

The solidDB with AcceleratorLib is a library file that is included in the solidDB Development Kit. You may request the SDK from the Solid Website at:

http://www.solidDB.com

For example, if you are using solidDB with HP-UX, the AcceleratorLib library file is `solidac.a`. Refer to Section 3.2, “Linking Applications for the AcceleratorLib” for a list of platform-specific libraries.

The AcceleratorLib library for all platforms contains the following:

- Solid data management functionality
- SA API header (`sa.h`) for local user applications
- Solid Control API interface header (`sscapi.h`) for local user applications

For details on linking a user application to the AcceleratorLib library, read Section 3.2, “Linking Applications for the AcceleratorLib”.

3.1.1 Libraries for Remote Applications

For the purposes of this AcceleratorLib guide, a "remote" application is any application that is not linked to the server - that is, any application that is not using the AcceleratorLib library. Thus an application that is running on the same node as the database server, but that is not linked to it, is considered to be a "remote" application. A remote application communicates with the server through a network communications protocol such as TCP/IP. A "local" application, on the other hand, is linked to the AcceleratorLib library, and can call functions in that library directly, without going through a network protocol.

Because a remote application goes through the network communications protocol, the AcceleratorLib does not improve performance for remote applications. Only the local application (the one that is directly linked to the accelerator library) has higher performance.

In some cases, however, remote applications can benefit from improved performance by using the SA API, which allows low-level operations to read from and write to the database.

If you are using a remote application, you may need to link to the following libraries in the solidDB SDK into your application.

- Link to the Solid Control API stub library (`solidctrlstub.lib` for Windows platforms), when you have Control API function calls in your application and you want to run your application remotely. (Note that if your application is a local rather than remote application —i.e. if it is directly linked to the AcceleratorLib library - then you do not need `solidctrlstub.lib`.)

For more details on the Control API Stub library (`solidctrlstub.lib`), read the section called “Using Dual-Mode Applications with the AcceleratorLib”.

- Link to Solid SA API (`solidimpsa.lib` for Windows platforms) if you are running a remote Solid SA API application (without AcceleratorLib).

If you are using ODBC, SA API, or JDBC as remote applications only (that do not use Control API function calls), then you do not need to link to `solidctrlstub.lib`.

3.1.2 Sample C Applications

For Accelerator Control API usage samples (available in C programming language), refer to `samples/aclib`, `samples/aclib_smartflow` and `samples/control_api` under the installation directory. These C samples reflect linked applications that use ODBC API functions to connect to solidDB servers.

3.1.3 Using Data Synchronization

If you are new to solidDB data synchronization, *solidDB SmartFlow Data Replication Guide* contains sample scripts.

Before you run the sample C application `acsnet.c` (under directory `samples/aclib_smartflow`), it is recommended that you become familiar with solidDB functionality by doing at least one of the following:

- Using solidDB (without the AcceleratorLib) to run the SQL scripts contained in *solidDB SmartFlow Data Replication Guide*. These scripts are found in `samples/smartflow`.
- Running the SQL scripts locally, using the solidDB AcceleratorLib. As a prerequisite, you are required to set up an application to start the server according to the instructions in this chapter. For details, read Section 3.2, “Linking Applications for the AcceleratorLib” and Section 3.3, “Starting and Shutting Down solidDB AcceleratorLib”.

Note

You cannot use the SA API to run synchronization commands.

- Running the implementation sample file `aclibstandalone.c`, which with the AcceleratorLib library, emulates a normal server. The sample file is located in directory `samples/aclib`.

After using any of these methods, it is possible to run all the steps in *solidDB SmartFlow Data Replication Guide*'s chapter titled *Getting Started with Data Synchronization* using Solid SQL Editor (solsql) or SolidConsole.

Setting up Your ODBC Application with the SmartFlow Synchronization Scripts

You can build an ODBC application, similar to the sample C application `acsNet.c`, to execute all statements required to set up, configure, and run a synchronizing environment. You can find `acsNet.c` under directory samples/aclib_smartflow.

To set up sample databases for use with an ODBC client application, you can execute sample scripts `replica3.sql`, `replica4.sql`, `replica5.sql`, and `replica6.sql`, all of which you can find in the samples/smartflow/eval_setup directory. These sample scripts contain SQL statements that write new data to replica(s) and control the execution of synchronization messages. These scripts may be run independently through the Solid SQL Editor (solsql) or SolidConsole.

Alternatively, you can embed the SQL statements into a C/ODBC application, compile, and link it directly to the AcceleratorLib library. When linked with the AcceleratorLib, the sample scripts allow you to get the performance benefit inherent in AcceleratorLib's architecture.

The sample program `embed.c` in the samples/odbc directory illustrates how to set up databases with an ODBC client application using AcceleratorLib. You can insert the SQL commands from the sample scripts, such as `replica3.sql`, etc., into the `embed.c` application.

3.2 Linking Applications for the AcceleratorLib

The solidDB AcceleratorLib is a library that must be linked to a user application. As long as the application is running, local and remote application requests for Solid data management services are available through the library.

Note

If you are writing remote user applications that use Solid Control API, you will need to link your remote application to the Solid Control API stub library (for example, `solidctrlstub.lib` for Windows). If you are using Solid SA API remotely (without AcceleratorLib) then you need to link to a separate Solid API library (`solidimpsa.lib` for Windows). If you are only using ODBC, SA API, or JDBC remotely, without Control API, then there is no need to link to the Solid Control API stub library.

You link only one application directly to the AcceleratorLib library at one time. However, once the linked application is up and running, and the server started, any network client can connect to the server using any of the protocols supported by the server, which depends on the operating system. These are for example, TCP/IP, shared memory and named pipes. Remote clients cannot use direct function calls.

When linking an application to solidDB with the AcceleratorLib, use one of the following libraries required for your operating system. Refer to your operating system documentation.

Table 3.1. AcceleratorLib System Libraries

Platforms	solidDB with AcceleratorLib Library
Windows	`solidimpac.lib` (this is an import library file that gives you access to the real library file, which is `ssolidacxx.dll`)
Solaris	`solidac.a`
HP-UX	`solidac.a`
Linux	`solidac.a`
VxWorks	`solidac.a`

3.2.1 Preparing User Applications for the AcceleratorLib

To allow your application to use the solidDB with AcceleratorLib, be sure to:

- Link to the AcceleratorLib library instead of to the driver libraries.

  If you are using remote applications, you may need to link to other libraries. For details, read Section 3.1.1, “Libraries for Remote Applications”.

- Change the connect string to the local or remote server name. For details, read Section 3.2.2, “Establishing a Local or Remote Connection to solidDB with the AcceleratorLib”.

- If needed, add calls to `SSCStartServer` and `SSCStopServer` or other Control API calls. For details, read Section 4.5, “Control API Reference”.

Signal Handlers

Signal handlers are used to report the occurrence of an exceptional event to the application, for example division by zero. You must not set signal handlers in user applications because they would override the signal handlers that are set by the AcceleratorLib. For example, if the user application sets a signal handler for floating point exceptions, that setting overrides the handler set by the AcceleratorLib. Thus the server is unable to catch, for example, division by zero.

Dynamic Link Library

Solid provides both a "static" and a "dynamic" version of the AcceleratorLib library. The names of the dynamic link library files are shown below for some major platforms. (For the names of the static libraries, see Section 3.2, “Linking Applications for the AcceleratorLib”.

Table 3.2. Library Files

Platforms	**solidDB with AcceleratorLib Library**
Windows	`ssolidacxx.dll`
Solaris	`ssolidacxx.so`
HP-UX	`ssolidacxx.sl`
Linux	`ssolidacxx.so`

Both the static and dynamic library files contain a complete copy of the solidDB server, in library format. When you use a static library file (e.g. `lib/solidac.a`), you link your program directly to it, and of course both your code and the library code are written to the resulting executable file. If you link to a dynamic library file, the code from the library is not included in the output file that contains your executable program. Instead, the code is loaded from the dynamic link library separately when your program runs.

Other than changing the size of your executable, there is very little difference between linking to the static library file vs. the dynamic library file. The total amount of code in memory at any one time is, of course,

similar (assuming that you are executing a single client and a single server on your computer). Performance is also similar, although there is a slight amount of extra overhead if you use the dynamic library.

The main advantage of using the dynamic link library file is that you can save memory IF you execute more than one copy of the server in the same computer. For example, if you are doing development work on a single computer and you want to have both a SmartFlow Master and SmartFlow Replica on the computer at the same time, or you'd like to have a HotStandby Primary and a HotStandby Secondary at the same time, then you may prefer to use the dynamic library so that you don't have multiple copies of the AcceleratorLib library in memory at the same time.

On Microsoft Windows, the solidDB AcceleratorLib includes the additional file `lib/solidimpac.lib`. On Microsoft Windows, if you want to use a dynamic link library, you do not link directly to the `ssolidacxx.dll` dynamic link library itself; instead you link to `solidimpac.lib`, which is an import library. This links only a small amount of code to your client executable. At the time that your client program actually executes, the `ssolidacxx.dll` file will automatically be loaded by the Microsoft Windows operating system, and your client will be able to call the usual AcceleratorLib functions in that .dll file. The .dll file must be in your load path when you run the program that references that .dll.

Note

Using the dynamic link library file does not mean that you can have multiple "local" clients linked to the solidDB server. Even with the dynamic library approach, you are still limited to a single local client; all other clients must be remote clients, which means that they will communicate with the solidDB server by using TCP or some other network protocol, rather than the direct function calls available to the local client.

MakeFile Examples

Following are examples for providing the library name in Windows and Vxworks.

Example 3.1. Microsoft Windows MakeFile Example

For the Microsoft Windows makefile example below, the solidDB library name for the AcceleratorLib is used, `solidimpac.lib`.

```
# compiler
CC        = cl
# compiler flags
CFLAGS        = -I. -DSS_WINDOWS -DSS_WINNT
# linker flags and directives
SYSLIBS = libcmt.lib kernel32.lib advapi32.lib netapi32.lib wsock32.lib
```

```
user32.lib oldnames.lib gdi32.lib
LFLAGS  = ..\solidimpac.lib
OUTFILE  = -Fe

# MyApp building
all: myapp

myapp: myapp.c
  $(CC) $(CFLAGS)  $(OUTFILE)myapp myapp.c /link$(LFLAGS)
/NODEFAULTLIB:libc.lib
```

Example 3.2. VxWorks MakeFile Example

For the VxWorks makefile example below, the solidDB library name for the AcceleratorLib is used, `solidac.a`. Note that the example uses backslashes. If your makefile program does not support backslashes in pathnames, then change the backslashes to slashes.

```
CC      = ccppc
CFLAGS  = -DSS_UNIX -DSS_VXW -I. -I..\..\include -I$(WIND_BASE)
\target\h \
          -DCPU=PPC603 -DMV2600
LFLAGS  = -nostartfiles -s -r ..\..\lib\solidac.a
OUTFILE = -o

# solidDB with AcceleratorLib samples building

all:    acsNet acsrv

acsNet: acsNet.c
        $(CC) $(CFLAGS) $(OUTFILE)acsNet acsNet.c $(LFLAGS)

acsrv:  acsrv.c
        $(CC) $(CFLAGS) $(OUTFILE)acsrv acsrv.c $(LFLAGS)
```

3.2.2 Establishing a Local or Remote Connection to solidDB with the AcceleratorLib

Once an application is linked to the AcceleratorLib library, it can use ODBC API or SA API to establish a local or remote connection directly to the local server. An application can also establish remote connections to other solidDB servers, including others using the AcceleratorLib.

Establishing a Local Connection

When you establish a local connection, the client's calls to the server are direct function calls to the AcceleratorLib library; they do not go through the network.

In the ODBC API, to establish a connection to a local server (i.e. to the server that was linked to the application), the user application calls the `SQLConnect` function with the literal string "localserver". Note that for the local server connection you can also specify an empty source name "". You can also specify a local server name, but this will cause AcceleratorLib to use a "remote" connection (to go through the network rather than to use the direct function calls to the AcceleratorLib library).

The following ODBC API code examples connect directly to a local solidDB server with username dba and password dba :

```
rc = SQLConnect(hdbc, "localserver", (SWORD)SQL_NTS, "dba", 3, "dba", 3);
```

or

```
rc = SQLConnect(hdbc, "", (SWORD)SQL_NTS, "dba", 3, "dba", 3);
```

In the SA API, to establish a connection, the user application calls the `SaConnect` function with the literal string "localserver" (not the server name). Note that for the local server connection you can also specify an empty source name "". You can also specify a local server name, but this will cause AcceleratorLib to use a "remote" connection (to go through the network rather than to use the direct function calls to the AcceleratorLib library).

The following SA API example code connects directly to a solidDB server with username dba and password dba :

```
SaConnectT* sc = SaConnect("localserver", "dba", "dba");
```

or

```
SaConnectT* sc = SaConnect("", "dba", "dba");
```

Establishing a Remote Connection

When you establish a remote connection, the client's calls to the server will go through the network rather than use the direct function calls to the AcceleratorLib library.

In the ODBC API, to establish a remote connection, the user application calls the `SQLConnect` function with the name of the remote server. The following ODBC API code example connects to a remote solidDB server with username dba and password dba. In this example, the network protocol that the client and server use is "tcp" (TCP/IP). The server is named "remote_server1" and the port that it listens on is 1313.

```
rc = SQLConnect(hdbc, "tcp remote_server1 1313",
(SWORD)SQL_NTS, "dba", 3, "dba", 3);
```

In the SA API, to establish a remote connection, the user application calls the `SaConnect` function with the name of the remote server. In this example, the network protocol that the client and server use is "tcp" (TCP/IP). The server is named "remote_server1" and the port that it listens on is 1313.

```
SaConnectT* sc = SaConnect("tcp remote_server1 1313", "dba", "dba");
```

3.3 Starting and Shutting Down solidDB AcceleratorLib

You can start up, restart, and shut down the solidDB server from the following APIs:

- Explicitly, from local (linked) user application by calling the Control API function `SSCStartServer` to start solidDB and `SSCStopServer` to shut it down.

  When you start a new solidDB server that does not already have a database, you must explicitly specify that solidDB create a new database with the function `SSCStartServer()` with the

  ```
  -Uusername
  -Ppassword
  -Ccatalogname (the default database catalog name)
  ```

  parameters. For details, read Section 3.3.1, “Explicit Start up with the Control API Function SSCStartServer”.

- Implicitly, when connecting locally to solidDB for the first time, either using ODBC API function `SQLConnect` or SA API function `SaConnect`. In this case, shut down occurs when the last local connection disconnects from solidDB using either function `SQLDisconnect` or `SaDisconnect`.

  When solidDB engine/server is started implicitly from the application, it checks if a database already exists in the solidDB directory. If a database file is found, solidDB will automatically open that database. If a database file is not found, then solidDB will give an error. (solidDB will not create a new database during

implicit startup. To create a new database, you must use an explicit startup function, such as `SSCStartServer`, and pass the appropriate parameters.)

For details, read Section 3.3.2, “Implicit Start Up with ODBC API Function Call `SQLConnect`” and Section 3.3.3, “Implicit Start Up with SA API Function Call `SaConnect`”.

Note

1. At server start up, recovery is performed if needed before control returns to the application. Therefore, if the server is successfully started, it is ready to serve application requests. For the duration of the application process, the server can be started or stopped as needed.

2. If you want to start a diskless server, you must start the server with Control API function `SSCStartDisklessServer`.

3.3.1 Explicit Start up with the Control API Function SSCStartServer

To start solidDB explicitly, have the user application call the following Control API function:

```
SSCStartServer (int argc, char* argv [ ],
SscServerT* h, SscStateT runflags)
```

where parameters are:

Table 3.3. `SSCStartServer` Parameters

Parameter	Description
argc	The number of command line arguments.
argv	Array of command line arguments that are used during the function call. The argument `argv[0]` is reserved for the path and filename of the user application only and must be present. For valid options, see `SSCStartServer` options below.
h	Each server has a "handle" (a pointer to a data structure) that identifies that server and indicates where information about that server is stored. This handle is required when referencing the server with other Control API functions. The handle of the server is provided to you when you call the `SSCStartServer` function. To get the handle of the server, you create a variable that is of type pointer-to-server-handle (i.e. you create an *`SSCServerT *`*, which is a pointer to a handle — es-

Parameter	Description
	sentially a pointer to a pointer) and you pass that when you call `SSCStartServer`. If the server is created successfully, then the `SSCStartServer` function will write the handle (pointer) of the new server into the variable whose address you passed.
`runflags`	The options for this parameter are *`SSC_STATE_OPEN`* (remote connections are allowed) and *`SSC_STATE_PREFETCH`* (the server performs a prefetch if needed). Prefetch refers to the memory and/or disk cache that provides read-ahead capability for table content. See below for a runflags parameter entry: `runflags = SSC_STATE_OPEN | SSC_STATE_PREFETCH;`

When you start the server for the first time, solidDB creates a new database only if you have specified the database administrator's username, password, and a name for the default database catalog. For details on the database catalog, read the section "Managing Database Objects" in chapter "Using Solid SQL for Data Management" in *solidDB Administration Guide*.

For example:

```
SscServerT h; char* argv[4];
argv[0] = "appname"; /* path and filename of the user app. */
argv[1] = "-UDBA"; /* user name */
argv[2] = "-PDBA"; /* user's password */
argv[3] = "-CDBA"; /* catalog name */
/* Start the server */
rc = SSCStartServer(argc, argv, &h, run_flags);
```

If you start the server without an existing database and do not specify a database catalog name, solidDB returns an error that the database is not found.

Note

If you already have an existing database, you do not need to specify the username and password, or the catalog name.

By default, the database will be created as one file (with the default name, `solid.db`, or the name you specified in the `solid.ini` file) in the solidDB working directory, where the current working directory is

located. An empty database containing only the system tables and views uses approximately 850 KB of disk space. The time it takes to create the database depends on the hardware platform you are using.

After the database has been created, solidDB starts listening to the network for remote client connection requests.

SSCStartServer argv Parameter Options

Following are the command line options for the *argv* parameter. Note that all options are case sensitive.

Table 3.4. SSCStartServer *argv* Options

Option	Description
-c dir	Changes working directory.
-m	Monitors users' messages and SQL statements.
-n name	Set server name.
-U username	Specifies the username of the administrator for the database being created. The username is case insensitive. The username requires at least two characters. For username, the maximum number of characters is 80. A user name must begin with a letter or an underscore. Use lower case letters from a to z, upper case letters from A to Z, the underscore character '_', and digits from 0 to 9.  **Caution** You must remember your username to be able to connect to solidDB. There are no default usernames; the username you enter when creating the database is the only username available for connecting to the new database.
-P password	Specifies the password of the administrator for the database being created. The password is case insensitive. The password requires at least three characters. Passwords can begin with a letter, an underscore, or a digit. Use lower case letters from a to z, upper case letters from A to Z, the underscore character '_', and digits from 0 to 9.
-C catalogname	Specifies the name of the default catalog of the database, which is required if you are starting the server for the first time. For details on catalogs, read the section "Managing Database Objects" in chapter "Using Solid SQL for Data Management" in *solidDB Administration Guide*.
-xautoconvert	Converts database format to current version and starts server process.
-xforcerecovery	Does a forced roll-forward recovery.

Option	Description
-xignoreerrors	Ignores index errors.
-xtestblocks	Tests database blocks.
-xtestindex	Tests database index.

Example 3.3. Starting up `SSCStartServer`

Start up `SSCStartServer` with the servername, the catalog name, and the administrator's username and password:

```
SscStateT runflags = SSC_STATE_OPEN; SscServerT h; char* argv[5];
argv[0] = "appname"; /* path and filename of the user app. */
argv[1] = "-nsolid1"; argv[2] = "-UDBA" argv[3] = "-PDBA";
argv[4] = "-CDBA"; /* Start the server */ rc =
SSCStartServer(argc, argv, &h, run_flags);
```

Note

If you already have an existing database, you do not need to specify the username and password, or the catalog name.

Shut Down with SSCStopServer

If the server is started by `SSCStartServer`, then it must be shut down with the following function call in the embedded application:

```
SSCStopServer()
```

For example:

```
/* Stop the server * /
SSCStopServer (h, TRUE);
```

3.3.2 Implicit Start Up with ODBC API Function Call `SQLConnect`

When function `SQLConnect` is called for the first time, the server is implicitly started. The server is shut down implicitly when the user application calls function `SQLDisconnect` and this is the last open local connection. Note that the server will shut down regardless of currently existing remote connections.

Note

When you start the server for the first time, you must create a solidDB database by using function `SSCStartServer()` and specifying the default database catalog, along with the administrator's username and password. For a description and example, read Section 3.3.1, “Explicit Start up with the Control API Function SSCStartServer”.

Following is an example of implicit start up and shut down with `SQLConnect` and `SQLDisconnect`:

```
/* Connection #1 */
rc = SQLConnect (hdbc1, "", SQL_NTS, "dba", SQL_NTS, "dba",
SQL_NTS); //Server Started Here
... odbc calls

/* Disconnect #1 */
SQLDisconnect (hdbc1); //Server Shut Down Here

/* Connection #2 */
rc = SQLConnect (hdbc2, "", SQL_NTS, "dba", SQL_NTS, "dba",
SQL_NTS); //Server Started Here
... odbc calls

/* Disconnect #2 * /
SQLDisconnect (hdbc2); //Server Shut Down Here
```

OR

```
/* Connection #1*/
rc = SQLConnect (hdbc1, "", SQL_NTS, "dba",
SQL_NTS, "dba", SQL_NTS); // Server Started Here

/* Connection #2*/
rc = SQLConnect (hdbc2, "", SQL_NTS, "dba", SQL_NTS, "dba", SQL_NTS);

... odbc calls

/* Disconnect #1 */
SQLDisconnect (hdbc1);
```

```
/* Disconnect #2 * /
SQLDisconnect (hdbc2); // Server Shut Down Here
```

Note

If the server is started with an `SSCStartServer` function call, then `SQLDisconnect` does not do implicit shut down. The server must be shut down explicitly, either by `SSCStopServer`, **ADMIN COMMAND 'shutdown'**, or other explicit shutdown methods.

```
SscStateT runflags = SSC_STATE_OPEN;
SscServerT server;
SQLHDBC hdbc;
SQLHENV henv;
SQLHSTMT hstmt;

/* Start the server */
SSCStartServer (argc, argv, &server, runflags); // Server Started Here

/* Alloc environment */
rc = SQLAllocEnv (&henv);

/* Connect to the database */
rc = SQLAllocConnect (henv, &hdbc);
rc = SQLConnect (hdbc, "", SQL_NTS, "dba", SQL_NTS, "dba", SQL_NTS);

/* Delete all the rows from table foo */
rc = SQLAllocStmt (hdbc, &hstmt):
rc = SQLExecDirect (hsmt, (SQLCHAR *) "DELETE FROM FOO", SQL_NTS);

/* Commit */
rc = SQLTransact (henv, hdbc, SQL_COMMIT);
rc = SQLFreeStmt (hstmt, SQL_DROP);

/* Disconnect */
SQLDisconnect (hdbc);
SQLFreeConnect (hdbc);

/* Free the environment */
SQLFreeEnv(henv);
```

```
/* Stop the server */
SSCStopServer (server, TRUE);  // Server Shut Down Here
```

3.3.3 Implicit Start Up with SA API Function Call `SaConnect`

When function `SaConnect` is called for the first time, the server is implicitly started. The server is shut down implicitly when the user application calls function `SaDisconnect` and there are no more subsequent connections.

Note

When you start the server for the first time, you must create a solidDB database by using function `SSCStartServer()` and specifying the default database catalog, along with the username and password. For a description and example, read Section 3.3.1, “Explicit Start up with the Control API Function SSCStartServer”.

Following is an example of implicit start up and shut down with `SaConnect` and `SaDisconnect`:

```
/* Open Connection */
SaConnect(...);

Server Started Here
... sa calls

/* Close Connection */
SaDisconnect(...);

Server Shut Down Here
```

Note

If the server is started with an `SSCStartServer` function call, then it must be shut down only with an `SSCStopServer` function call.

3.3.4 Shutting Down solidDB AcceleratorLib

From solidDB client interfaces and even from another remote solidDB connection, you can shut down the solidDB server as long as you have SYS_ADMIN_ROLE privileges.

Programmatically, you can perform the shut down from an application such as SolidConsole (Query window or command line), Solid SQL Editor (solsql), or Solid Remote Control[1] .

To do this, perform the following steps:

1. To prevent new connections to solidDB, close the database(s) by entering the following command:

   ADMIN COMMAND 'close'

2. Exit all solidDB users by entering the following command:

   ADMIN COMMAND 'throwout all'

3. Stop solidDB by entering the following command:

   ADMIN COMMAND 'shutdown'

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

Note

You can use explicit methods (e.g. `SSCStopServer`) to shut down a server that was started with implicit methods (e.g. `SQLConnect`). The converse is not true; for example, you cannot use `SQLDisconnect` to stop a server that was started with `SSCStartServer`.

3.3.5 Implicit Start Configuration Parameter

solidDB implicitly starts up the server only when a local connection is established. In the *`Accelerator`* section of the `solid.ini` configuration file, the parameter *`ImplicitStart`*, by default, is set to Yes. This default setting starts the server automatically when you use the function `SQLConnect` which is required for any ODBC connection. The function `SaConnect` behaves similarly. When it is called for the first time, the server is implicitly started.

[1]When using Solid Remote Control for steps 1-3, you enter the command name only without quotes (for example, `close`).

Chapter 4. Description of Control API

The Control API (also called the SSC API) is a set of functions that provide a simple and efficient means to control the tasking system of a solidDB.

4.1 Retrieving Task Information

To retrieve a list of all active tasks, use the `SSCGetActiveTaskClass` function. To retrieve a list of all suspended tasks, use the `SSCGetSuspendedTaskClass` function. To get the priority of a task class, use the `SSCGetTaskClassPrio` function.

4.2 Notifying Functions of a Special Event

The AcceleratorLib provides fine tuning of priority tasks. You can use the `SSCSetNotifier()` function to establish that solidDB calls a specified user-defined function whenever a special event occurs. Special events that the function detects are:

- solidDB server shutdown
- Bonsai merge from the index to the storage tree
- Bonsai merge interval maximum
- Backup or checkpoint request
- Idle server state
- Netcopy request (which is a request to send a network copy of the Primary database to the Secondary server) received from the Primary server.
- Completion of a netcopy request, which occurs when the server is started up with the new database received through the network copy (netcopy).

4.2.1 Obtaining solidDB Status and Server Information

You can use the function `SSCGetStatusNum` to view current status information of the solidDB database server. The following information is displayed:

- Number of rows that are not merged from the Bonsai Tree to the Storage Tree
- Number of server threads

The `SSCGetServerHandle` function returns the solidDB server handle if the server is running.

You can also use the function `SSCIsRunning` to verify if the server is running and the function `SSCIsThisLocalServer` to verify whether an application is linked to the local AcceleratorLib server library (for example, `ssolidacxx.dll` for Windows platforms) or a "dummy" server library (for example, `solidctrlstub.lib` for Windows platforms) used to test remote applications that are using Control API.

4.3 Summary of Control API Functions

The following is a brief summary of Control API functions and where the function is described in the Control API Function Reference section.

Table 4.1. Summary of Control API Functions

Function	Description	For more details, see
`SSCSetCipher`	Sets an application-provided encryption library.	See Section 4.11, “SSCSetCipher”.
`SSCStartServer`	Starts a solidDB AcceleratorLib Server.	See Section 4.15, “SSCStartServer”.
`SSCStartDisklessServer`	Starts a solidDB AcceleratorLib diskless server.	See Section 4.14, “SSCStartDisklessServer”.
`SSCSetState`	Sets the state of a solidDB server (for example, SSC_STATE_OPEN indicates if subsequent connections are allowed). Setting the state to ~SSC_STATE_OPEN will block local, as well as remote, connections.	See Section 4.13, “SSCSetState”.
`SSCRegisterThread`	Registers an AcceleratorLib application thread for the server. Registration is required in every thread in the user application before any Accelerator API function can be called.	See Section 4.10, “SSCRegisterThread”.
`SSCUnregisterThread`	Unregisters an AcceleratorLib application thread for the server. Registration removal is required in every thread that is registered before terminating.	See Section 4.17, “SSCUnregisterThread”.
`SSCStopServer`	Stops solidDB server.	See Section 4.16, “SSCStopServer”.

Function	Description	For more details, see
SSCSetNotifier	Specifies a user-defined function which solidDB calls at a specified event, such as merge, backup, shutdown, etc.	See Section 4.12, “SSCSetNotifier”.
SSCIsRunning	Returns non-zero if the server is running.	See Section 4.8, “SSCIsRunning”.
SSCIsThisLocalServer	Indicates whether the application is linked to the solidDB server with the AcceleratorLib or the "dummy" (solidctrlstub) library to test Solid remote applications using the AcceleratorLib's Control API.	See Section 4.9, “SSCIsThisLocalServer”.
SSCGetServerHandle	Returns the solidDB server handle if the server is running.	See Section 4.6, “SSCGetServerHandle”.
SSCGetStatusNum	Gets solidDB status information.	See Section 4.7, “SSCGetStatusNum”.

4.4 Control API and Equivalent ADMIN COMMANDs

Control API functions have equivalent Solid SQL extension ADMIN COMMANDs. You can execute these commands from both remote and local sites through Solid tools, such as SolidConsole, Solid Remote Control (solcon), and Solid SQL Editor (solsql).

Refer to Appendix A, *AcceleratorLib Parameters* for details on Control API equivalent ADMIN Commands.

4.5 Control API Reference

The following pages describe each Control API function in alphabetic order. Each description includes the purpose, synopsis, parameters, return value, and comments.

4.5.1 Function Synopsis

The declaration synopsis for the function is:

```
ReturnType SSC_CALL function(modifier parameter[,...]);
```

The ReturnType varies, but is usually a value that indicates success or failure of the call. Return values are described in more detail later in this section.

SSC_CALL is required for portability. SSC_CALL specifies the calling convention of the function. It is defined appropriately for each platform in the `sscapi.h` file.

Parameters are in italics and are described below.

Parameter Description

In each function description, parameters are described in a table format. Included in the table is the general usage type of the parameter (described below), as well as the use of the parameter variable in the specific function.

Parameter Usage Type

The table below shows the possible usage type for Control API parameters. Note that if a parameter is used as a pointer, it contains a second category of usage to specify the ownership of the parameter variable after the call.

Table 4.2. Control API Parameter Usage Types

Usage Type	Meaning
in	Indicates the parameter is input.
output	Indicates the parameter is output.
in out	Indicates the parameter is input/output
use	Applies only to a pointer parameter. It means that the parameter is just used during the function call. The caller can do whatever it wants with the parameter after the function call. This is the most common type of parameter passing.
take	Applies only to a pointer parameter. It means that the parameter value is taken by the function. The caller cannot reference the parameter after the function call. The function or an object created in the function is responsible for releasing the parameter when it is no longer needed.
hold	Applies only to a pointer parameter. It means that the function holds the parameter value even after the function call. The caller can continue to reference the parameter value after the function call and is responsible for releasing the parameter.  **Warning** Because this parameter is shared by the user and the server, you must not release it until the server is finished with it. In general, you can free the held object after you free the object that is holding it. For example:

Usage Type	Meaning

	`conn = SaConnect("", "dba", "dba");`
	`/* Connection is held until cursor is freed */`
	`scur = SaCursorCreate(conn, "mytable");`
	`...`
	`SaCursorFree(scur);`
	`/* After we free the cursor, it is safe to free */`
	`/* the connection (or, as in this case, call a */`
	`/* server function that frees the connection). */`
	`SaDisconnect(conn);`

4.5.2 Return Value

Each function description indicates if the function returns a value and the type of value that is returned.

SscTaskSetT

When functions return a value of type *`SscTaskSetT`*, this definition is used as a bit mask. *`SScTaskSetT`* is defined in `sscapi.h` with the following possible values:

```
SSC_TASK_NONE
SSC_TASK_CHECKPOINT
SSC_TASK_BACKUP
SSC_TASK_MERGE
SSC_TASK_LOCALUSERS
SSC_TASK_REMOTEUSERS
SSC_TASK_SYNC_HISTCLEAN
SSC_TASK_SYNC_MESSAGE
SSC_TASK_HOTSTANDBY
SSC_TASK_HOTSTANDBY_CATCHUP
SSC_TASK_ALL (all of the above tasks)
```

Note that the HotStandby "netcopy" and HotStandby "copy" operations are performed by the task "SSC_TASK_BACKUP"; there is no separate task "SSC_TASK_NETCOPY".

4.5.3 Control API Error Codes and Messages

Control API functions may return the following error codes and messages:

Table 4.3. Error Codes and Messages for Control API Functions

Error Code/Message	Description
SSC_SUCCESS	Operation is successful.
SSC_ERROR	Generic error.
SSC_ABORT	Operation aborted.
SSC_FINISHED	SSCAdvanceTasks returns this message if all tasks are executed.
SSC_CONT	SSCAdvanceTasks returns this message if there are still more tasks to execute.
SSC_CONNECTIONS_EXIST	There are open connections.
SSC_UNFINISHED_TASKS	There are unfinished tasks.
SSC_INFO_SERVER_RUNNING	The server is already running.
SSC_INVALID_HANDLE	Invalid local server handle given. This server does not match the one started through `SSCStartServer`.
SSC_INVALID_LICENSE	No license or invalid license file found.
SSC_NODATABASEFILE	No database file found.
SSC_SERVER_NOTRUNNING	The server is not running.
SSC_SERVER_INNETCOPYMODE	The server is in netcopy mode (applies only with High Availability/HotStandby).

These constants (SSC_SUCCESS, etc.) are defined in the `sscapi.h` file.

4.6 SSCGetServerHandle

`SSCGetServerHandle` returns the solidDB server handle if the server is running.

Synopsis

```
SscServerT SSC_CALL SSCGetServerHandle(void)
```

Comments

This function has no corresponding Solid SQL extension ADMIN COMMAND.

Return value

- NULL if the server is not running.
- The server handle if the server is running.

4.7 SSCGetStatusNum

`SSCGetStatusNum` gets the status information of solidDB.

Synopsis

```
SscRetT SSC_CALL SSCGetStatusNum(SscServerT h, SscStatusT stat,
      long * num)
```

The `SSCGetStatusNum` function accepts the following parameters:

Table 4.4. `SSCGetStatusNum` Parameters

Parameters	Usage Type	Description
h	in, use	Handle to server.
stat	in	Specifies the status identifier for retrieval:
num	out	If the function was successful, then when it returns this parameter's value will be set to either the number of writes not merged, or the number of server threads, depending upon which information was requested.

Comments

This function has no corresponding Solid SQL extension ADMIN COMMAND.

If you call `SSCGetStatusNum` and pass it an unrecognized value for the *`stat`* parameter, then the function will return SSC_SUCCESS.

Return value

- SSC_SUCCESS - Operation is successful. This value is also returned if you pass an invalid value for the stat parameter.
- SSC_ERROR - Operation failed.
- SSC_SERVER_INNETCOPYMODE - The server is in netcopy mode (CarrierGrade/HotStandby option only)
- SSC_SERVER_NOTRUNNING - The server is not running.

4.8 SSCIsRunning

`SSCIsRunning` returns non-zero if the server is running.

Synopsis

```
int SSC_CALL SSCIsRunning(SscServerT h)
```

The `SSCIsRunning` function accepts the following parameters:

Table 4.5. `SSCIsRunning` Parameters

Parameters	Usage Type	Description
h	in, use	Handle to server

Return value

- 0 - The server is not running.
- nonzero - The server is running.

Comments

This function has no corresponding Solid SQL extension ADMIN COMMAND.

4.9 SSCIsThisLocalServer

`SSCIsThisLocalServer` indicates whether the application is linked to a solidDB server or the "dummy" (`solidctrlstub`) library. The `solidctrlstub` library allows developers to test Solid remote applications using Control API without linking the AcceleratorLib library and modifying the source code.

Synopsis

```
int SSC_CALL SSCIsThisLocalServer(void)
```

Return value

- 0 - The application is not linked to the solidDB server.
- 1 - The application is linked to the solidDB server.

Comments

This function has no corresponding Solid SQL extension ADMIN COMMAND.

4.10 SSCRegisterThread

`SSCRegisterThread` registers a solidDB application thread for the server. Every thread that uses Control API, ODBC API, or SA API must be registered. The `SSCRegisterThread` function must be called by the thread before any other AcceleratorLib API function can be used.

If the application has only one (main) thread, that is, if the application creates no threads itself, then registration is not required.

Before a thread terminates, it must unregister itself by calling the function `SSCUnregisterThread`.

Synopsis

```
SscRetT SSC_CALL SSCRegisterThread(SscServerT h)
```

The `SCCRegisterThread` function accepts the following parameters:

Table 4.6. `SCCRegisterThread` Parameters

Parameters	Usage Type	Description
h	In, Use	Handle to server

Return value

- SSC_SUCCESS
- SSC_INVALID_HANDLE

Comments

This function has no corresponding Solid SQL extension ADMIN COMMAND.

See also

`SSCUnregisterThread`

4.11 SSCSetCipher

`SSCSetCipher` function sets application-provided cipher and encryption/decryption functions. The provided cipher will be automatically used when the related database encryption command line arguments are used in `SSCStartServer`.

Synopsis

```
void SSC_CALL SSCSetCipher(
    void* cipher,
    char* (SSC_CALL *encrypt)(void *cipher, int page_no, char *page,
        int n, size_t pagesize),
    int (SSC_CALL *decrypt)(void *cipher, int page_no, char *page,
        int n, size_t pagesize));
```

Table 4.7. `SSCSetCipher` Parameters

Parameters	Usage Type	Description
cipher	in	A pointer to an application-specific cipher object, for example, an encryption password. solidDB server does not use or otherwise in-

Parameters	Usage Type	Description
		terpret this pointer. It only passes it to the application-provided encryption/decryption functions.
encrypt	in	A pointer to the application-provided encryption function. This function is called from the server when it must encrypt the database file or log file pages. The function parameters are: • *cipher* - A pointer to the application provided cipher object. • *page_no* - A server-provided page number. The encryption algorithm can safely ignore it, or use it as part of the encryption key. • *n* - The number of pages to encrypt. • *pagesize* - The size of page to encrypt. • *page* - A pointer to the data buffer to be encrypted. The size of the data buffer is: *n*pagesize* The function must return the pointer to the encrypted data buffer to be written to the file of size (*n*pagesize*). The server does not free the pointer to the encrypted data buffer. The data buffer passed to the function as 'page' parameter can be overwritten or manipulated by the encryption function in any way. For example, the encryption function can encrypt the data "in place" and return the 'page' pointer.
decrypt	in	A pointer to the application-provided decryption function. This function is called from the server when it has read part of the encrypted database or log file and has to decrypt it. The function parameters are: • *cipher* - A pointer to the application provided cipher object.

Parameters	Usage Type	Description
		• *page_no* - A server-provided page number. The decryption algorithm can safely ignore it, or use it as part of the decryption key. • *n* - The number of pages to decrypt. • *pagesize* - The size of page to decrypt. • *page* - A pointer to the data buffer to be decrypted. The size of the data buffer is: *n*pagesize*

Return value

The `SSCSetCipher` function does not return any value. It is supposed to be invoked before the AcceleratorLib server is started by using the `SSCStartServer` function.

Comments

The decryption function is supposed to return a non-zero value if it has successfully decrypted the pages and the 0 value if decryption has failed for one reason or another. In the latter case, the server makes an emergency shutdown since it is not able to continue. The function is supposed to return the encrypted data in the same buffer as given with the parameter 'page'.

Example 4.1. Using the AcceleratorKib Encryption API

The following code illustrates the usage of the AcceleratorLib encryption API. The encryption method is trivial, namely the XOR obfuscation.

```
char* SS_CALLBACK encrypt(void *cipher, int page_no, char *page, int np,
size_t pagesize)
{
        size_t n = np*pagesize;
        int *key = cipher;
        size_t i;
```

```
        for (i=0; i<n; i++) {
            page[i] ^= (i+*key);
        }
        return page;
}


bool SS_CALLBACK decrypt(void *cipher, int page_no, char *page, int np,
size_t pagesize)
{
        size_t n = np*pagesize;
        int *key = cipher;
        size_t i;

        for (i=0; i<n; i++) {
            page[i] ^= (i+*key);
        }

        return TRUE;
}
...

int main(int argc, char** argv)
{
    int key = 17;
...
        SSCSetCipher(&key, encrypt, decrypt);
        sscret = SSCStartServer(argsc, args, &h, SSC_STATE_OPEN);
        ....
        SSCStopServer(h, FALSE);
...

}
```

4.12 SSCSetNotifier

`SSCSetNotifier` sets the callback functions that an AcceleratorLib server calls when it is started or stopped. The function does not have a corresponding ADMIN COMMAND.

Synopsis

```
SscRetT SSC_CALL SSCSetNotifier(SscServerT h, SscNotFunT what,
    notify_fun handler, void* userdata
)
```

The `SSCSetNotifier` function accepts the following parameters:

Table 4.8. `SSCSetNotifier` Function Parameters

Parameters	Usage Type	Description
h	in	Handle to server.
what	in	Specifies event for notification. Options are: • SSC_NOTIFY_EMERGENCY_EXIT This function is called if a server crashes after it has been activated with `SSCStartServer()`. The notifier call `SSCSetNotifier()` has to be issued before `SSCStartServer()` • SSC_NOTIFY_SHUTDOWN Function is called at shutdown. • SSC_NOTIFY_SHUTDOWN_REQUEST Function is called when the server receives the shutdown request and may shut down if the user-defined function accepts the request. You can refuse the shut down by returning SSC_ABORT from the notified function, or proceed with the request by returning SSC_CONTINUE. • SSC_NOTIFY_ROWSTOMERGE Function is called when there is data in the bonsai index tree that needs to be merged to the storage server. • SSC_NOTIFY_MERGE_REQUEST

Parameters	Usage Type	Description
		Function is called when the *MergeInterval* parameter setting in the solid.ini configuration file is exceeded and the merge has to start. • SSC_NOTIFY_BACKUP_REQUEST Function is called when a backup is requested. You can refuse the backup by returning SSC_ABORT from the notified function. • SSC_NOTIFY_CHECKPOINT_REQUEST Function is called when a checkpoint is requested. You can refuse the checkpoint by returning SSC_ABORT from the notified function. • SSC_NOTIFY_IDLE Function is called when the server switches to the idle state. • SSC_NOTIFY_NETCOPY_REQUEST This callback function applies to the CarrierGrade/HotStandby option only. The function is called when a netcopy request (which is a request to send a network copy of the Primary database to the Secondary server) is received from the Primary server. For details on the netcopy command, refer to *solidDB High Availability User Guide*. • SSC_NOTIFY_NETCOPY_FINISHED This callback function applies to the CarrierGrade/HotStandby option only. The function is called when a netcopy request is finished. When finished, the server is started up with the new database received through the network copy (netcopy) and SSC_NOTIFY_FINISHED is called to inform the application that the server is again available.
notify_fun_handler	in, hold	User function to call.

Parameters	Usage Type	Description
userdata	in, hold	User data to be passed to the notify function. Be sure to read the warning on releasing a parameter of usage type *hold* under the section called “Parameter Description”.

Return value

- SSC_SUCCESS - Request from the server accepted.

  CarrierGrade (HotStandby) option only :

  If SSC_NOTIFY_NETCOPY_FINISHED returns SSC_SUCCESS, then all other application connections are terminated and the server is set to "netcopy listening mode". In this mode the server accepts the connection from the Primary server and the only possible operation for the Secondary server is to receive the data from the hotstandby netcopy command. For more details on "netcopy listening mode", read *solidDB High Availability User Guide*. (Note that in the past, "netcopy listening mode" was also called "backup listening mode".)

- SSC_ABORT - Request from the server denied.

  CarrierGrade (HotStandby) option only:

  If the SSC_NOTIFY_NETCOPY_REQUEST returns SSC_ABORT, then the netcopy is not started and an error code (SRV_ERR_OPERATIONREFUSED) is returned to the Primary server.

- SSC_INNETCOPYMODE - The server is in netcopy mode (CarrierGrade/HotStandby option only).

  SSC_SERVER_NOTRUNNING - The server is not running.

Comments

This function has no corresponding Solid SQL extension ADMIN COMMAND.

Releasing a parameter of usage type *hold* should be done with caution. Read the warning for hold the section called “Parameter Description”.

The user-defined notifier function should not call any SA, SSC, or ODBC function.

When creating a user-defined notifier function, you must conform to the following prototype:

```
int SSC_CALL mynotifyfun(SscServerT h, SscNotFunT what ,void* userdata);
```

Once you have used SSC_CALL to explicitly define the convention for your user function, then you use the `SSCSetNotifier` function to register the function so that it is called during the specified event; for example:

```
SscRetT SSCSetNotifier(h, SSC_NOTIFY_IDLE, mynotifyfun, NULL);
```

Example

Example 4.2. Calling a Function upon Shutdown

Assume a user creates the function `user_own_shutdownrequest`, which is called every time a shut down is requested:

```
int user_own_shutdownrequest(SscServerT * handle, int reason, void
    *udata);
{
  if (shutdown not needed) {
    return SSC_ABORT;
  }
  return SSC_SUCCESS; /*Proceed with shutdown*/
}
```

The `SSCSetNotifier` function can then be called as follows to specify that `user_own_shutdownrequest` gets called before the server is shut down.

```
SSCSetNotifier(handle, SSC_NOTIFY_SHUTDOWN, user_own_shutdownrequest, NULL);
```

Note

If function user_own_shutdownrequest returns SSC_ABORT, the shut down is not allowed and if the function returns SSC_SUCCESS, the shut down can proceed.

4.13 SSCSetState

`SSCSetState` sets the state of an AcceleratorLib server. This allows you to control whether the server accepts subsequent connections, and whether the server uses prefetch.

If the server is set to "open", then the server will accept connections. If the server is set to "closed", then it will not accept any further connections (this applies to both local connections and remote connections); however, any connections that have already been made are allowed to continue.

Turning on prefetch tells the server to "read ahead" to fetch data that is likely to be referenced soon. Prefetch requires more memory or disk cache space. When prefetch is on, performance is generally higher. When prefetch is off, less memory is required. Turning on prefetch is most useful if you have queries that involve large sequential scans of the server. For example, if you use reports or aggregate functions to get values for the entire database (or large portions of it), then prefetch may help. Prefetch is generally not useful if all your queries involve only one or a few records. Because prefetch uses up memory, prefetch may actually reduce performance in systems with little available memory.

The following guidelines may help you decide when to use prefetch.

DO use prefetch when: you have a lot of available memory (or disk cache space) and your queries require large sequential scans.

DO NOT use prefetch when: you have little available memory and your queries generally read unrelated records one at a time.

Synopsis

```
SscRetT SSC_CALL SSCSetState(ssc_serverhandle_t h,SscStateT runflags)
```

The `SSCSetState` function accepts the following parameters:

Table 4.9. `SSCSetState` Function Parameters

Parameter	Usage Type	Description
h	in, use	Handle to the server.
runflags	in	Options can be a combination of the flags SSC_STATE_OPEN, which means new remote connections are allowed and SSC_STATE_PREFETCH, which means the user allows the server to do a prefetch if needed. Following is an example of the possible combinations: • set server open: state = state | SSC_STATE_OPEN; • set server closed: state = state & ~SSC_STATE_OPEN; • set prefetch on: state = state | SSC_STATE_PREFETCH; • set prefetch off: state = state & ~SSC_STATE_PREFETCH;

Return value

- SSC_SUCCESS - Operation is successful.
- SSC_ERROR - Operation failed.
- SSC_SERVER_INNETCOPYMODE - The server is in netcopy mode (HotStandby only).
- SSC_SERVER_NOTRUNNING - The server is not running.

Comments

This function has a corresponding Solid SQL extension ADMIN COMMAND. The command is:

ADMIN COMMAND 'close';

4.14 SSCStartDisklessServer

`SSCStartDisklessServer` starts a diskless server using the AcceleratorLib.

Synopsis

```
SscRetT SSC_CALL SSCStartDisklessServer (int argc, char* argv[ ],
    SscServerT * h, SscStateT runflags, char* lic_string, char* ini_string);
```

The `SSCStartDisklessServer` function accepts the following parameters:

Table 4.10. `SSCStartDisklessServer` Parameters

Parameters	Usage Type	Description
argc	in	The number of command line arguments.
argv	in, use	Array of command line arguments that are used during the function call. The argument argv[0] is reserved only for the path and filename of the user application and must be present. For a list of valid arguments, refer to the `SSCStartDisklessServer` Parameter Options listed below.
h	out	Returns a handle to the started server. This handle is needed when referencing the server with other Control API functions.
runflags	in	The only option for this parameter is:

Parameters	Usage Type	Description
		SSC_STATE_OPEN - Remote connections are allowed. runflags = SSC_STATE_OPEN
lic_string	in	Specifies the string containing the Solid license file.
ini_string	in	Specifies the string containing the Solid configuration file.

SSCStartDisklessServer Parameter Options

Following are the command line options for the *argv* parameter.

Table 4.11. Command Line Options for the *argv* Parameter

Option	Description
-h	Displays help.
-n*name*	Sets server name.
-U*username*	Specifies the username for the data. The username is case insensitive. The username requires at least two characters. For username, the maximum number of characters is 80. A user name must begin with a letter or an underscore. Use lower case letters from a to z, upper case letters from A to Z and the underscore character '_', and digits from 0 to 9. **Note** You must remember your username to be able to connect to solidDB. There are no default usernames ; the username you enter when creating the database is the only username available for connecting to the new database.
-P*password*	Specifies the given password for the data. The password is case insensitive. The password requires at least three characters. Passwords can begin with a letter, underscore, or a number. Use lower case letters from a to z, upper case letters from A to Z and the underscore character '_', and digits from 0 to 9.
-C*catalogname*	Specifies the catalog name for the data, required if you are starting the server for the first time. When specifying this parameter, be sure to use uppercase C. For details on catalogs, read the section "Managing Database

Option	Description
	Objects" in chapter "Using Solid SQL for Data Management" in *solidDB Administration Guide*.
-x ignoreerrors	Ignores index errors.

Return value

- SSC_SUCCESS - The server is started.
- SSC_ERROR - The server failed to start.
- SSC_SERVER_INNETCOPYMODE - The server is netcopy mode (CarrierGrade/HotStandby option only).
- SSC_INFO_SERVER_RUNNING - The server is already running.
- SSC_INVALID_HANDLE - Invalid local server handle given.
- SSC_INVALID_LICENSE - No license or invalid license file found.

Comments

By default, the state is set to SSC_STATE_OPEN.

This function has no corresponding Solid SQL extension ADMIN COMMAND.

Example

Example 4.3. `SSCStartDisklessServer`

```
SscStateT runflags = SSC_STATE_OPEN;
SscServerT h;
char* argv[4]; /* pointers to four parameter strings */
int argc = 4;
char* lic = get_lic(); /* get the license */
char* ini = get_ini(); /* get the solid.ini */
SscRetT rc;
argv[0] = "appname"; /* path and filename of the user app. */
argv[1] = "-Udba"; /* user name */
argv[2] = "-Pdba"; /* user's password */
```

```
argv[3] = "-Cdba"; /* catalog name */
/* Start the diskless server */
rc = SSCStartDisklessServer(argc, argv, &h, runflags, lic, ini);
```

Note

In the example, `get_ini()` and `get_lic()` are functions that a user must write. Each must return a string that contains the `solid.ini` file text or the `solid.lic` license file.

If you do not specify a catalog name, solidDB returns an error.

See also

`SSCStopServer`

See also Chapter 5, *Using the Diskless Capability*.

4.15 SSCStartServer

`SSCStartServer` starts the AcceleratorLib. In multi-thread environments, the server runs in a separate thread(s) from the client. For the duration of the application, the application can start or stop the server sub-routines as needed.

Note that the third parameter is an "out" parameter. If the server is started successfully, then the `SSCStartServer` routine will set this parameter to point to the handle for this server.

Note

If you are starting a diskless server, you must start the server with Control API function `SSCStartDisklessServer`. Read Section 4.14, “SSCStartDisklessServer”.

Synopsis

```
SscRetT SSC_CALL SSCStartServer(int argc, char* argv[], SscServerT* h
    SscStateT runflags)
```

The `SSCStartServer` function accepts the following parameters:

Table 4.12. `SSCStartServer` Parameters

Parameters	Usage Type	Description
`argc`	in	Number of command line arguments.
`argv`	in, use	Array of command line arguments. For a list of valid arguments, refer to Section 4.15, “SSCStartServer”.
`h`	out	Returns a handle to the started server. This handle is needed when referencing the server with other Control API functions.
`runflags`	in	Options can be one or both of the following: • SSC_STATE_OPEN - Remote connections are allowed. • SSC_STATE_PREFETCH - Server will do a prefetch if needed. For example: `runflags = SSC_STATE_OPEN & SSC_STATE_PREFETCH`

Return value

- SSC_SUCCESS - The server started.
- SSC_ERROR - The server failed to start.
- SSC_ABORT
- SSC_BROKENNETCOPY - Database corrupted because of incomplete netcopy.
- SSC_FINISHED
- SSC_CONT
- SSC_CONNECTIONS_EXIST
- SSC_UNFINISHED_TASKS
- SSC_INVALID_HANDLE - Invalid local server handle given.

- SSC_INVALID_LICENSE - No license or invalid license file found.
- SSC_NODATABASEFILE - No database file found.
- SSC_SERVER_NOTRUNNING
- SSC_INFO_SERVER_RUNNING - The server is already running.
- SSC_SERVER_INNETCOPYMODE - The server is in netcopy mode (CarrierGrade/HotStandby option only).
- SSC_DBOPENFAIL - Failed to open database.
- SSC_DBCONNFAIL - Failed to connect to database.
- SSC_DBTESTFAIL - Database test failed.
- SSC_DBFIXFAIL - Database fix failed.
- SSC_MUSTCONVERT - Database must be converted.
- SSC_DBEXIST - Database exists.
- SSC_DBNOTCREATED - Database not created.
- SSC_DBCREATEFAIL - Database create failed.
- SSC_COMINITFAIL - Communication init failed.
- SSC_COMLISTENFAIL - Communication listen failed.
- SSC_SERVICEFAIL - Service operation failed.
- SSC_ILLARGUMENT - Illegal command line argument.
- SSC_CHDIRFAIL - Failed to change directory.
- SSC_INFILEOPENFAIL - Input file open failed.
- SSC_OUTFILEOPENFAIL - Output file open failed.
- SSC_SRVCONNFAIL - Server connect failed.
- SSC_INITERROR - Operation init failed.

- SSC_CORRUPTED_DBFILE - Assert or other fatal error.
- SSC_CORRUPTED_LOGFILE - Assert or other fatal error.

Comments

By default, the state is set to SSC_STATE_OPEN.

This function has no corresponding Solid SQL extension ADMIN COMMAND.

When you start a new solidDB server, you must explicitly specify that solidDB create a new database with the function `SSCStartServer()` with the **-U***username* **-P***password* **-C***catalogname* (the default database catalog name) parameters. For details, read Section 3.3.1, “Explicit Start up with the Control API Function SSCStartServer”.

If you are restarting a database server (i.e. a database already exists in the directory), then `SSCStartServer` will use the existing database.

The `SSCStartServer` function may spawn multiple threads to run the server tasks. The server tasks include processing local and remote client requests, as well as running various background tasks, such as checkpoints, merges, etc.

See also

`SSCStopServer`

4.16 SSCStopServer

`SSCStopServer` stops an AcceleratorLib server.

Note that you can use explicit methods (e.g. `SSCStopServer`) to shut down a server that was started with implicit methods (e.g. `SQLConnect`). The converse is not true; for example, you cannot use `SQLDisconnect` to stop a server that was started with `SSCStartServer`.

An application is not limited to starting and stopping the server once each time that the application is run. After the server has been stopped, the application can re-start the server by using `SSCStartServer`.

Synopsis

```
SscRetT SSC_CALL SSCStopServer(SscServerT h, bool force)
```

The `SSCStopServer` function accepts the following parameters:

Table 4.13. `SSCStopServer` Parameters

Parameter	Usage Type	Description
h	in, use	Handle to server
force	in	Options are: • TRUE - stop server in all cases. • FALSE - stop server if there are no open connections. Otherwise, stop fails.

Return value

- SSC_SUCCESS - The server is stopped.
- SSC_CONNECTIONS_EXIT - There are open connections.
- SSC_UNFINISHED_TASKS - Tasks that are executing.
- SSC_ABORT
- SSC_ERROR

Comments

Remote users can stop solidDB by using **ADMIN COMMAND ' shutdown'**. Refer to Appendix A, *AcceleratorLib Parameters* for details.

The FALSE option does not permit shut down if there are open connections to the database or existing users. This option is equivalent to Solid SQL extension **ADMIN COMMAND 'shutdown'**.

The `SSCSetState()` with the *`&~SSC_STATE_OPEN`* option prevents new connections to solidDB.

See also

`SSCStartServer`

`SSCSetState`

4.17 SSCUnregisterThread

`SSCUnregisterThread` unregisters a solidDB application thread for the server. The `SSCUnregisterThread` function must be called by every thread that has registered itself with the function `SSCRegisterThread`. The function is called before the thread terminates.

Synopsis

```
SscRetT SSC_CALL SSCUnregisterThread(SscServerT h)
```

The `SCCUnregisterThread` function accepts the following parameters:

Table 4.14. `SCCUnregisterThread` Parameters

Parameter	Usage Type	Description
h	in, use	Handle to server

Return value

- SSC_SUCCESS
- SSC_INVALID_HANDLE

Comments

SSC_CALL is required to explicitly define the calling convention of your user function. It is defined in the `sscapi.h` file appropriately for each platform.

This function has no corresponding Solid SQL extension ADMIN COMMAND.

See also

`SSCRegisterThread`

Chapter 5. Using the Diskless Capability

solidDB AcceleratorLib allows you to create a database engine that runs without any disk storage space. This is useful in embedded systems that do not have hard disks, such as line cards in a network router or switch.

There are two main ways to run a diskless server: alone, and as a replica in a SmartFlow system. In each case, you will start the server by using the AcceleratorLib function call `SSCStartDisklessServer()`.

Diskless Server Alone

If you run a diskless server alone, then of course it has no way to read data when it starts up and no way to write data when it shuts down. This means that each time the server starts, it starts without any previous data.

Furthermore, since the server has no way to write data to disk, if the server is shut down abnormally (due to a power failure, for example), then any data in the server is lost and cannot be recovered. You can reduce the risk of data loss by using the Solid CarrierGrade option to create a "hot standby" machine that contains a copy of the data. For more information about this hot standby capability, see *solidDB High Availability User Guide*.

Diskless Server as Part of a SmartFlow System

A diskless server may be a replica in a SmartFlow system. In this situation, the replica may send data to the master server and may download data from that master server. Thus, even though the replica has no disk storage or other permanent storage of its own, it may make some or all of its data persistent within the SmartFlow system.

5.1 Configuration Parameters for a Diskless Server

This section describes the parameter settings for implementing and maintaining a diskless server.

5.1.1 Parameters Used in Diskless Servers

The following sections of the configuration file contain parameters that have specific settings for diskless servers.

Index File Section

Following are the configuration parameters that affect the index file.

FileSpec_[1...N] parameter

The *FileSpec* parameter describes the name and the maximum size of the database file. To define the maximum size in bytes for the in-memory capacity of the diskless server, the *FileSpec* parameter accepts the following arguments:

- database file name - Since the diskless server does not create a physical database file, this parameter is not used; however, a dummy value must be provided for this argument.
- maximum file size - This setting is required. You need to specify the size in bytes that is large enough to store all the data in the diskless server. Note that the maximum file size must be smaller than the cache size, which is set with the *CacheSize* parameter.

The default value for the *FileSpec* parameter is `solidr.db`, *5000000 bytes*. For example:

```
FileSpec_1=SOLIDR.db 5000000
```

Note

If you specify multiple files, then the maximum file size setting must be the sum of all the *FileSpec* parameter settings.

Not surprisingly, the maximum size is limited by the physical memory available, since a diskless machine has no disk to use as swap space for virtual memory. Note that on some platforms, the amount of physical memory available to the applications may be less than the amount of physical memory in the machine. For example, in some versions of Linux on 32-bit systems, the amount of memory available to applications is limited to one half or one quarter of the theoretical address space (4GB) because Linux reserves the 1 or 2 most significant bits of the address for its own memory manager.

If the data in memory exceeds the maximum file size, the error message 11003 is displayed:

```
File write failed, configuration exceeded
```

CacheSize

The *CacheSize* parameter defines the amount of main memory in bytes that the server allocates for the buffer cache. For example:

```
CacheSize=10000000
```

The setting for this value depends on the following criteria for diskless servers:

- For disk based tables, the cache size (in bytes) should be at least 20% larger than the maximum file size (that is, the amount of data) set with the *FileSpec* parameter since this data is held in the buffer cache. The 20% overhead is an estimate that may vary depending on the usage of the database. For example:

  ```
  [IndexFile]
  FileSpec_1=solid.db 10MB
  CacheSize=12MB
  ```

- Even if no disk-based tables are used (the database is created by using in-memory tables), the cache is necessary to hold system tables. In that case, the minimum cache size is 1-2 MB. The space occupied by the system tables depends of the number and complexity of database objects and whether SmartFlow is used or not.

- The cache size must be less than the physical memory available for running the diskless server.

  Total memory used by the diskless server can be estimated as follows. (Note that the TOTAL of all of these must fit within the amount of physical memory available, which means that the cache size must in fact be significantly smaller than the amount of physical memory available to the server:)

  ```
  CacheSize
  + 5MB
  + (100K * number of users * number of active statements per user)
  + in-memory table space
  + (HSB operations to be sent to the Secondary) [1][2]
  ```

  [1] This term of the equation applies to HotStandby users only. An HSB Primary server needs some memory to store HotStandby operations that are to be sent to the Secondary server. During a temporary network failure between the Primary server and the Secondary diskless server, the Primary may continue to accept transactions from an application. When the network connection is restored between the servers, updates from the Primary server are sent to the Secondary server. (HotStandby uses the transaction log to store these operations. A diskless server cannot write the transaction log to disk, of course, so the information must be stored in memory.) This memory is separate from the Cache.

[2] For this term of the equation, the maximum limit is currently 1 MB or 512 operations, whichever is lower. Unlike on a disk-based server, the transaction log is not allowed to keep growing until it uses up all available space.

The exact amount required also depends on other factors, including the nature of the queries executed against the server. Naturally, the amount of memory available to the server is less than the total physical memory, since the operating system etc. will use up some of the physical memory.

Com Section

Following are the configuration parameters that affect communication between the master and the diskless replica server (if you are using the diskless server as a SmartFlow replica server).

Listen parameter [Com]

This is the protocol and name that the diskless server uses when it starts listening to the network. Its default is Operating System dependent. Refer to "Managing Network Connections" in *solidDB Administration Guide*.

5.1.2 Configuration Parameters that Do Not Apply to Diskless Engines

The following configuration file parameters (grouped by section) are disabled or inoperable for diskless servers. These parameters affect behaviors that do not apply to diskless engines.

Table 5.1. Configuration Parameters not Applicable to Diskless Engines

Parameter	Description
[General] Section	
CheckpointInterval	This parameter is disabled since checkpoints do not apply to diskless servers.
[IndexFile] Section	
ReadAhead	No physical read from the database file, so this parameter is inoperable
PreFlushPercent	No physical write to the database file, so this parameter is inoperable
[Logging] Section	
LogEnabled	This parameter is disabled since transaction logging is always disabled for diskless servers.

Parameter	Description
	 **Note** Diskless mode supports transaction rollback only. Transaction rollbacks are typically used when some failure interrupts a half-completed transaction. The diskless mode does not support roll-forward recovery.

Chapter 6. Using solidDB AcceleratorLib With Java

Note

This chapter assumes that you are already familiar with the material in the preceding chapters. If you jumped straight to this chapter because you are interested only in Java/JDBC, not C/ODBC, you will have missed too much material to understand this entire chapter.

6.1 Overview of solidDB JDBC Accelerator (SJA)

A Java/JDBC program, like a C/ODBC program, may use the solidDB AcceleratorLib to get higher performance and greater control over the server. SJA enables a Java application to start a local solidDB server, which will be loaded into the Java Virtual Machine context from a dynamic library called 'ssolidacxx'. The Java application will then be able to connect to the solidDB server and use the services solidDB DBMS provides through a standard JDBC API.

The client application program will get higher performance because it is directly linked to the server library, so calls to server functions do not have the overhead of network (RPC) calls. The application will have greater control because it can call functions (methods) in the Solid Server Control (SSC) library to do things such as assign priorities to certain types of tasks. For example, the application might give itself a high priority and might give remote client applications a low priority.

solidDB JDBC Accelerator (SJA) can only be used when the server and client are linked together; thus, if the Java application and the solidDB server are to be run in separate hosts, SJA cannot be used.

Not surprisingly, only the "local" client (the one that is linked to the AcceleratorLib library) can bypass the network and get the higher performance of the AcceleratorLib. Other client programs may also use the server, but they must connect through the network, and are treated as "remote" programs even if they are running on the same computer as the solidDB server. You may only have one "local" client; the rest are "remote". The remote programs may be a mix of C and Java programs.

The language in which the local client is written does not restrict which languages the remote clients can be written in. For example, if you use JDBC Accelerator, the remote client programs may use C, Java, or both.

6.2 How the Accelerator Works

As with C programs, Java/JDBC programs that want to use the AcceleratorLib must link to the solidDB AcceleratorLib library (ssolidacxx). This library contains the entire solidDB server, except that it is in the form of a callable library instead of a standalone executable program. The ssolidacxx used with Java/JDBC is the same as the ssolidacxx that was explained in previous chapters; there are not separate versions for Java and C clients. Linking to the library allows a client program to avoid the overhead of RPC (Remote Procedure Calls) through the network.

When you use the AcceleratorLib with Java/JDBC, you link the following into a single executable process:

- solidDB AcceleratorLib library,
- your Java-language client program, and
- the JVM.

The table below shows the different "layers" in the executable process.

Table 6.1. Layers in the Executable Process

Local Java/JDBC client application
JVM (Java Virtual Machine)
solidDB Accelerator Library (`ssolidacxx`)

Java commands in your client are executed by the JVM. If the command is a JDBC function call, then the JVM calls the appropriate function in `ssolidacxx`. The function call is "direct", rather than going through the network (through RPC). The calls are made using JNI (Java Native Interface). Note that you do not need to know about these low-level details. You do not need to write any JNI code yourself; you simply have to call the same JDBC functions that you would call if you were a remote client program.

Accessing a solidDB database from Java Accelerator is identical to accessing a solidDB database through RPC — with one exception: in order to access the database services, the application using Java Accelerator must first start the solidDB accelerator server. This is done with a proprietary API called `SolidServerControl` (SSC). SSC API calls are used to start, as well as to stop, the solidDB DBMS. The actual database connections are done with normal JDBC API. Both the `SolidServerControl` API and Solid's JDBC driver can be found in a .jar file named `SolidDriver2.0.jar`.

When the local solidDB server is started, it will be loaded into the Java Virtual Machine context from a dynamic library called `ssolidacxx`. The Java application will then be able to connect to the solidDB server and use the services solidDB DBMS provides through a standard JDBC API.

Every local client program that uses solidDB Java Accelerator follows the same basic three-step pattern:

1. Start the accelerator server with SolidServerControl
2. Access the database by using normal JDBC API
3. When database processing is done, stop the accelerator server again with SolidServerControl

The `SolidServerControl` classes for accessing solidDB accelerator server have been embedded inside Solid JDBC driver file, inside the `solid.ssc` package. The Solid JDBC driver jar file (`Solid-Driver2.0.jar`) contains the following packages:

- `solid.jdbc.*` Solid JDBC driver classes
- `solid.ssc.*` Solid Server Control classes (proprietary interface)

The classes inside the Solid Server Control (`solid.ssc`) package are:

- `SolidServerControl` (for starting and stopping solidDB server from Java)
- `SolidServerControlInitializationError` (for reporting errors)

For detailed information on `SolidServerControl` (SSC) class interface, see Section 6.6, “Solid Server Control (SSC) API”.

To start a solidDB server from a Java application, you must instantiate the class `SolidServerControl` in the beginning of your application and call the `startServer` method with correct parameters (examples are given below). After you've started the server, you should be ready to make a JDBC connection to the server.

6.3 System Requirements

You need the following to use the solidDB Java Accelerator:

- The solidDB AcceleratorLib library itself. This is a file named ssolidacxx. The filename extension varies depending upon the platform; some common names and platforms are listed below:
  - Microsoft Windows: `ssolidacxx.dll` and the import library `solidimpac.lib`
  - Solaris and Linux: `ssolidacxx.so`
  - HP-UX: `ssolidacxx.sl`
- A valid license file for using the solidDB server and the AcceleratorLib

- solidDB JDBC2 driver file (`SolidDriver2.0.jar`)
- Solid communication libraries for your platform (these are normally installed when you install the solidDB Development Kit).
- To compile the program, you must have JDK Version 1.3.1_03-b03 or later (JDK 1.4 or later on HP-UX), and an appropriate JDK/JRE to run the program. The JDK/JRE that you use MUST have a HotSpot runtime/compiler. SJA has been tested ONLY with HotSpot JREs.

6.4 Basic Usage

6.4.1 Installation

If you have installed a Java Development Kit (such as JDK 1.3), then you do not need to do any further installation. When solidDB is installed, it includes the library(s) that are needed when using the solidDB Java Accelerator.

Note

You may need to set PATH and CLASSPATH environment variables to appropriate values so that you can access the Java compiler, etc.

6.4.2 Compiling and Running a Program

In order for the server startup to succeed, you need to have at least a valid license for using solidDB and AcceleratorLib.

The `ssolidacxx` dynamic link library must be in the system search path. Proceed as follows:

1. Set the paths (examples from Microsoft Windows command prompt)

   set PATH=<path to your ssolidacxx DLL>;%PATH%

   Make sure you have the directory containing Solid communication libraries in your path too.

2. Set your path environment variable to include JDK's HOTSPOT runtime environment in (SJA has only been tested in hotspot JRE's). For example,

   set PATH=<your JDK directory>\jre\bin\hotspot;%PATH%

3. Save the example file included in the end of this chapter into a file named `SJASample.java` and compile it with the following command:

javac -classpath <Solid JDBC driver directory>/SolidDriver2.0.jar;. \ SJASample.java

4. Run your application with a command line resembling the next one:

   java -Djava.library.path=<path to ssolidacxx DLL> \ -classpath <Solid JDBC driver directory>/Solid-Driver2.0.jar;. \ <your application name>

   For example, on Microsoft Windows, if you installed the server to `C:\solid` and would like to run the SJASample program, then your command line would look like:

   java -Djava.library.path=C:\solid\bin -classpath C:\solid\jdbc\SolidDriver2.0.jar;. SJASample

   (On Microsoft Windows, the `ssolidacxx.dll` dynamic library is in the `bin` subdirectory of the solidDB root installation directory.)

   As in the example class SJASample, you must pass the solidDB server at least the following parameters with `SolidServerControl`'s `startServer` method:

```
-c<directory containing Solid license file>
-U<username>
-P<password>
-C<catalog>
```

   Note that upper and lower case "C" are both used, and they mean different things.

   Assuming you have all the necessary files (`ssolidacxx` library, communication libraries, JDBC driver and `solid.lic`) in your current working directory, you can start SJASample with a command line like the following one:

   java -Djava.library.path=. -classpath SolidDriver2.0.jar;. <your application>

   If all things went as they were supposed to go, you should now have a solidDB accelerator server up and running.

6.4.3 Making JDBC Connections

solidDB Java accelerator supports both local database connections as well as RPC based connections.

In order to make a local (non RPC-based) JDBC connection, you need to specify the JDBC driver that you are using 'localserver' at port 0. Thus, if you are making the database connection by using, for example, JDBC class DriverManager, connect by using the following statement (as also presented in the example code SJASample further below)

```
DriverManager.getConnection("jdbc:solid://localserver:0", myLogin, myPwd);
```

As you can see, the DriverManager uses the URL *"jdbc:solid://localserver:0"* for making a connection to the local server. If the `getConnection` subroutine is given another URL, the driver will probably try to connect with RPC.

So remember the URL -

```
jdbc:solid://localserver:0
```

when making java accelerator connections.

Note

Note! If you are using multiple threads (java.lang.Thread objects) that access solidDB Accelerator server inside your Java application, you must register each thread separately with the solidDB Accelerator server before you start any JDBC-related activities using that thread. The thread registration is done by calling `SolidServerControl` API's `registerThread` method in the thread's context. The thread registration must be done explicitly for each user thread (except the main thread) using Solid's JDBC driver.

The user must also explicitly unregister each thread that has been registered to the solidDB Accelerator server. To unregister a thread, call `SolidServerControl` API's 'unregisterThread'.

6.5 Limitations

Note

Solid 'admin commands' do not work in the Java accelerator context.

Caution

Java doesn't behave consistently if something fails outside the VM context (for example, inside a native method call). If something should assert (or even crash) in the solidDB server native code, Java either exits (when it notices an unexpected exception) or hangs up completely. In the latter case, you may have to kill the dangling java process manually.

6.6 Solid Server Control (SSC) API

Below is the complete public interface for the SolidServerControl class. For an example of a program that uses some of the methods in this class, see the file `samples/accelerator_java/SJASample.java`

```
        /**
         * See Solid AcceleratorLib User Guide
         * for the following constants
         */
        public final static int SSC_SUCCESS = 0;
        public final static int SSC_ERROR = 1;
        public final static int SSC_ABORT = 2;
        public final static int SSC_FINISHED = 3;
        public final static int SSC_CONT = 4;
        public final static int SSC_CONNECTIONS_EXIST = 5;
        public final static int SSC_UNFINISHED_TASKS = 6;
        public final static int SSC_INVALID_HANDLE = 7;
        public final static int SSC_INVALID_LICENSE = 8;
        public final static int SSC_NODATABASEFILE = 9;
        public final static int SSC_SERVER_NOTRUNNING = 10;
        public final static int SSC_INFO_SERVER_RUNNING = 11;
        public final static int SSC_SERVER_INNETCOPYMODE = 12;

        public final static int SSC_STATE_OPEN     = (1 << 0);
        public final static int SSC_STATE_PREFETCH = (1 << 1);


/**
 * Initiates a SolidServerControl class. Output is not directed to any
 * PrintStream.
 *
 * @return        SolidServerControl instance
 */
public static  SolidServerControl instance()
     throws SolidServerInitializationError;


/**
 * Initiates a SolidServerControl class. Output is being directed
 * to a PrintStream object given in parameter 'os'.
```

```
 *
 * @param os      the PrintStream for output
 * @return        SolidServerControl instance
 *
 */
public static SolidServerControl instance( PrintStream os )
     throws SolidServerInitializationError;

/**
 * setOutStream method sets the output to the given PrintStream
 *
 * @param os      the PrintStream for output
 */
public void setOutStream( PrintStream os );


/**
 * getOutStream returns the stream used for output in class
 * SolidServerControl
 *
 * @return        returns the outputstream of this object
 */
public PrintStream getOutStream();


/**
 * startServer starts the Solid Accelerator server
 *
 * @param argv       parameter vector for the accelerator server
 *                   ( be sure to give the working directory containing
 *                   Solid license file (f.ex. -c\tmp) first, in front
 *                   of other parameters. ) See Solid AcceleratorLib
 *                   User Guide for details of parameters that can
 *                   be passed to the Accelerator server.
 *
 * @param runflags   Options for this parameter are SSC_STATE_OPEN
 *                   (remote connections are allowed) and
 *                   SSC_STATE_PREFETCH (server will do a "prefetch"
 *                   if needed). Prefetch refers to the memory
 *                   and/or disk cache that provides read-ahead
 *                   capability for table content. Following is
 *                   a runflags parameter entry:
```

```
 *                    runflags |= SSC_STATE_OPEN & SSC_STATE_PREFETCH
 *
 * @return   the return value from the server :
 *              SSC_SUCCESS
 *              SSC_ERROR
 *              SSC_INVALID_LICENSE - No license or license file found.
 *              SSC_NODATABASEFILE  - No database file found.
 */
public long startServer( String[] argv, long runflags );


/**
 * stopServer stops the Solid Accelerator server
 *
 * @param runflags   Runflags for stopping the Solid Accelerator server.
 *                   See Solid AcceleratorLib User Guide for more
 *                   details.
 *
 * @return   the return value from the server
 *              SSC_SUCCESS if server is stopped.
 *              SSC_CONNECTIONS_EXIT if there are open connections.
 *              SSC_UNFINISHED_TASKS if there are still tasks that are
 *                                   executing.
 *              SSC_SERVER_NOTRUNNING if the server is not running.
 */
public long stopServer( int runflags );


/**
 * returns the state of the server, i.e. is the server running or not
 *
 * @return SSC_STATE_OPEN if server is up and running
 */
public int getState();




/**
* registerThread registers this user thread to Solid Accelerator server
```

```
*
*
* @return           the return value from the server
*                    SSC_SUCCESS            Registration succeeded.
*                    SSC_ERROR              Registration failed.
*                    SSC_INVALID_HANDLE     Invalid local server handle
given.
*                    SSC_SERVER_NOTRUNNING  Server is not running.
*/
public long registerThread();



/**
* unregisterThread unregisters this user thread from the
* Solid Accelerator server
*
*
* @return           the return value from the server
*                    SSC_SUCCESS            Registration succeeded.
*                    SSC_ERROR              Registration failed.
*                    SSC_INVALID_HANDLE     Invalid local server handle given.
*                    SSC_SERVER_NOTRUNNING  Server is not running.
*/
public long unregisterThread();
```

Appendix A. AcceleratorLib Parameters

This appendix provides a list of all parameters for the AcceleratorLib. Accelerator parameters appear in the *[Accelerator]* section of the solidDB configuration file (`solid.ini`).

For a description of all other solidDB parameters, refer to the appropriate Appendix in *solidDB Administration Guide*.

Note that you can change solidDB parameters in the following ways:

- Using the SolidConsole Configuration page.
- Entering the **ADMIN COMMAND 'parameter'** command in Solid solsql or SolidConsole.
- Manually editing the `solid.ini` configuration file.

Note that any changes to the `solid.ini` file using the methods above do not take effect until the next time that the server starts.

A.1 Accelerator Section

Table A.1. Accelerator Parameters

[Accelerator]	Description	Factory Value
ImplicitStart	If set to yes, this parameter starts solidDB automatically as soon as the ODBC API function `SQLConnect` is called in a user application. If set to no, solidDB must be explicitly started with a call to the Control API function `SSC-StartServer`.	yes

Glossary

A

Accelerated Application
: solidDB AcceleratorLib is implemented as a subroutine library. This library can be linked to your client application to create a single executable program, which we call an Accelerated Application. This single executable is both a server and a client of that server.

ASCIIZ
: The normal format for strings in the C programming language is ASCIIZ. A string is an ASCIIZ string if the last byte stores the value 0 ('\0', not the ASCII character for the digit '0') to mark the end of the string. Such a string is also sometimes referred to as a "null-terminated string", although strictly speaking this is incorrect because "NULL" is a pointer value and is multiple bytes on most platforms, while the '\0' is a single-byte character.

L

Local sort
: A "local sort" is a sort that is done on the client side by the SA library rather than on the server side by the server.

Lost update
: A "lost update" occurs when one user's update writes over another user's update without seeing the earlier update. For example, user1 starts a transaction, then user2 updates a column, then user1 updates that same column and commits her transaction. User2's update was "lost". Note that this is different from the "normal" case where one user makes a change and then commits it, and then another user starts a transaction, sees the result of the earlier change by the other user. The difference is that when there is a "lost" update, the first changes was not visible to the user who made the second change. Most modern database software prevents lost updates by using record locking or optimistic concurrency control.

T

Task
: On a Real-Time Operating System (RTOS), such as VxWorks, OSE, etc., a task is a "thread" of control running within the context of a process. A process could have one or more independent yet cooperating "programs" running within it. Each of these programs is called a task.

In a typical RTOS environment, tasks have immediate, shared access to system resources, while also keeping enough separate context to maintain individual threads of control. However, all codes of tasks within a process execute in a single common address space. Memory protection is not pre-assumed and is the responsibility of the programmers.

Index

M

N

O

P

R

S

T

U