IBM solidDB
IBM solidDB Universal Cache
Version 6.5

Shared Memory Access and Linked Library Access User Guide

SC23-9876-01

Note
Before using this information and the product it supports, read the information in "Notices" on page 87.

First edition, first revision

This edition applies to version 6, release 5, Fix Pack 3 of IBM solidDB (product number 5724-V17) and IBM solidDB Universal Cache (product number 5724-W91) and to all subsequent releases and modifications until otherwise indicated in new editions.

© Solid Information Technology Ltd. 1993, 2010

Contents

Figures v

Tables vii

Summary of changes ix

About this manual xi
Typographic conventions xi
Syntax notation conventions xii

1 Overview of shared memory access and linked library access 1
Shared memory access (SMA) 2
System requirements for SMA 4
SMA components and packaging 4
Linked library access (LLA) 6
System requirements for LLA 6
LLA components and packaging 6
Static vs. dynamic link libraries for LLA 7
solidDB APIs and drivers for SMA and LLA 8
solidDB SA API 9
solidDB ODBC API 9
solidDB JDBC API 10
solidDB Server Control API (SSC API) 10
solidDB Server Control API (SSC API) for Java 10
Configurations with local and remote applications types 10

2 Creating and running SMA applications 13
Creating SMA applications - overview 13
Modifying shared memory kernel parameters - overview 14
Preparing applications for SMA use with driver manager 20
Preparing applications for SMA use without driver manager 21
Establishing local connections for SMA 22
Starting and shutting down SMA server 23
Starting SMA server 24
Shutting down SMA server 24
Starting SMA server as a service (Windows) . . 25
Monitoring SMA 26
Troubleshooting SMA 27

3 Creating and running SMA applications with Java 29
Overview of using SMA with Java 29
Configuring your environment for SMA use with Java 29
Starting and shutting down SMA server 31
Starting SMA server 31
Shutting down SMA server 31
Making JDBC connections for SMA 31

4 Creating and running LLA applications 33
Configuring your environment for LLA use . . . 33
Establishing a local connection for LLA 34
Starting and shutting down LLA server 35
Explicit startup with SSC API function SSCStartServer 36
Implicit startup with ODBC API function call SQLConnect 37
Implicit startup with SA API function call SaConnect 38
Shutting down LLA server 39
Sample C applications for LLA 39
Samples for using LLA with advanced replication 40

5 Creating and running LLA applications with Java 43
Overview of using LLA with Java 43
Limitations 44
Configuring your environment for LLA use with Java 44
Starting and stopping LLA server with SSC API for Java 45
Making JDBC connections for LLA 45
Compiling and running a sample LLA program . . 45

6 Using the diskless capability 47

7 Creating and running remote or dual-mode applications 49
Example: Creating a dual-mode LLA application with ODBC and SSC API function calls 49
Establishing remote connections 49

Appendix A. Shared memory access parameters 51

Appendix B. Linked library access parameters 53

Appendix C. Configuration parameters for a diskless server 55
Parameters used in diskless servers 55
IndexFile section. 55
Com section 56
Configuration parameters that do not apply to diskless engines 57

Appendix D. solidDB Server Control API (SSC API). 59

Using SSC API 59
Retrieving task information 59
Notifying functions of a special event 59
Obtaining solidDB status and server information 59
SSC API and equivalent ADMIN COMMANDs 60
Summary of SSC API functions 60
SSC API reference 62
SSCGetServerHandle 64
SSCGetStatusNum 65
SSCIsRunning 65
SSCIsThisLocalServer 66
SSCRegisterThread 66
SSCSetNotifier 67
SSCSetState 69
SSCStartDisklessServer 70
SSCStartServer 72
SSCStartDisklessSMAServer 75
SSCStartSMAServer. 77
SSCStopServer 78
SSCUnregisterThread 79

Appendix E. SolidServerControl class interface 81

Index 85

Notices 87

Figures

1. Configurations with SMA, LLA and network connection based solidDB server 2
2. Example: SMA and LLA APIs for C/C++ programs. 9

Tables

1. Typographic conventions xi
2. Syntax notation conventions xii
3. SMA drivers (libraries) 5
4. SMA server applications 5
5. LLA libraries 7
6. SSC API and SA API libraries for remote applications 11
7. Minimum requirements of shared memory kernel parameters for SMA (HP-UX) 16
8. Minimum requirements of shared memory kernel parameters for SMA (Linux) 18
9. Minimum requirements of shared memory kernel parameters for SMA (Solaris) 19
10. SMA drivers (libraries) 20
11. SMA drivers (libraries) 22
12. Starting the SMA server 24
13. SOLSMASTART default address spaces 28
14. SMA drivers (libraries) 30
15. Starting the SMA server 31
16. Linked library access (LLA) system libraries 33
17. SSCStartServer parameters 36
18. Shared memory access parameters 51
19. Shared memory access parameters (client-side) 51
20. Accelerator parameters 53
21. Configuration parameters not applicable to diskless engines 57
22. Summary of control API functions 60
23. SSC API parameter usage types 63
24. Error codes and messages for SSC API functions 64
25. SSCGetStatusNum parameters 65
26. SSCIsRunning parameters 66
27. SCCRegisterThread parameters 67
28. SSCSetNotifier parameters 67
29. SSCSetState parameters 70
30. SSCStartDisklessServer parameters 71
31. SSCStartServer parameters 73
32. SSCStartDisklessSMAServer parameters 75
33. SSCStartSMAServer parameters 77
34. SSCStopServer parameters 79
35. SCCUnregisterThread parameters 80

Summary of changes

Changes for revision 01

- Usage information for runflags in SSCStartServer, SSCStartDisklessServer, SSCStartSMAServer, SSCStartDisklessSMAServer, and SSCSetState updated:
  - SSC_STATE_PREFETCH flag has been discontinued.
  - New values introduced:

    SSC_STATE_CLOSED – all new network and LLA or SMA connections are rejected, with the exception of solidDB® Remote Control (solcon) connections.

    SSC_DISABLE_NETCOPY – in HotStandby configuration, no netcopy can be received by the server for which this is set.
- Information on SMA driver signal handler added in sections Preparing applications for SMA use with driver manager, Preparing applications for SMA use without driver manager, and Shared memory access parameters.
- Section Starting SMA server as a service (Windows) added.
- New parameter **Accelerator.ReturnListenErrors** added in section Linked library access parameters.
- SSC API functions SSCRegisterThread and SSCUnregisterThread marked as deprecated; it is no longer necessary to register and unregister threads explicitly when using solidDB with linked library access (LLA). The thread registration is now handled implicitly.
- Dependency between the solidDB parameter **SharedMemoryAccess.MaxSharedMemorySize** and the different kernel parameters clarified in section Modifying shared memory kernel parameters - overview.
- Troubleshooting instructions updated in section Troubleshooting SMA.

About this manual

The IBM® solidDB shared memory access (SMA) and linked library access (LLA) enable applications to link to solidDB server directly, without the need to communicate through network protocols such as TCP/IP. SMA allows to link multiple applications while LLA allows to link one application. By replacing the network connection with local function calls, performance is improved significantly.

This guide contains information specific to SMA and LLA. This guide supplements the information contained in the *IBM solidDB Administrator Guide*, which contains details on administration and maintenance of solidDB.

This guide assumes a working knowledge of the C programming language, general DBMS knowledge, familiarity with SQL, and knowledge of a solidDB data management product, such as solidDB in-memory database, or solidDB disk-based engine. If you are going to use SMA or LLA with Java™, then this manual also assumes a working knowledge of Java.

Typographic conventions

solidDB documentation uses the following typographic conventions:

Table 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.
LockHashSize	This font is used for parameter names, function arguments, and Windows® registry entries.
argument	Words emphasized like this indicate information that the user or the application must provide.
Administrator Guide	This style is used for references to other documents, or chapters in the same document. New terms and emphasized issues are also written like this.

Table 1. Typographic conventions (continued)

Format	Used for
File path presentation	Unless otherwise indicated, 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. There may also be different chapters for different operating systems.

Syntax notation conventions

solidDB documentation uses the following syntax notation conventions:

Table 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.
. . .	A column of three dots indicates continuation of previous lines of code.

1 Overview of shared memory access and linked library access

The solidDB *shared memory access* (SMA) and *linked library access* (LLA) allow applications to link to solidDB server directly, without the need to communicate through performance-consuming network protocols such as TCP/IP. SMA allows to link multiple applications while LLA allows to link one application.

SMA and LLA are implemented as library files that contain a complete copy of the solidDB server in library format.

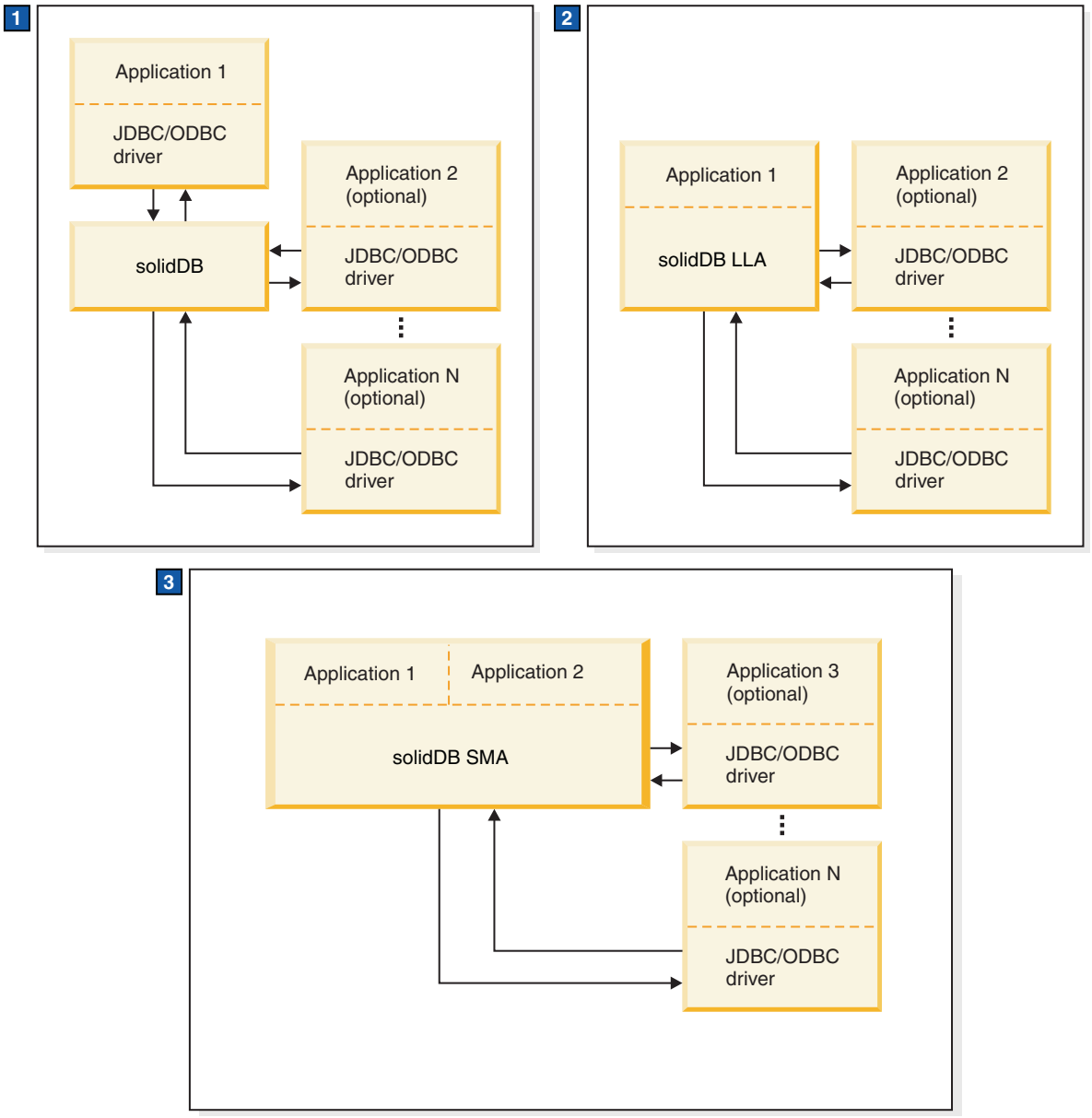
- In the case of SMA, the library that the applications link to can be seen as a driver. Before the linked application is started, the solidDB server is started in a SMA mode, which loads the SMA driver dynamically and allocates and initializes a shared memory segment that the applications use to access the database.
- In the case of LLA, the application links to the LLA library and the application and server are built into a single executable.

Your application does not have to be rewritten to use SMA or LLA. The applications communicate with the solidDB server using ODBC or JDBC calls, or the solidDB proprietary SA API.

The SMA and LLA servers can also handle requests from remote applications which connect to the server through communications protocols such as TCP/IP. The remote applications see the SMA or LLA server as similar to any other solidDB server, while the local SMA and LLA applications see a faster, more precisely controllable version of the solidDB server.

Also, similarly to network-based servers, multiple SMA and LLA servers can be run on the same node.

The solidDB server used with SMA and LLA can be disk-based or diskless. Both in-memory tables and disk-based tables are also supported.



1. In a standard solidDB database configuration, the applications and the server are separate programs.
2. In LLA configuration, LLA is a library that is linked into an application. Other applications may also communicate with the LLA server.
3. In SMA configuration, SMA is a driver library that multiple applications can link to. Other network connections based applications may also communicate with the SMA server.

Figure 1. Configurations with SMA, LLA and network connection based solidDB server

Shared memory access (SMA)

With shared memory access (SMA), multiple applications can be linked to a dynamic driver library that contains the full database server functionality. This means that the applications ODBC or JDBC requests are processed almost fully in the application process space, without a need for a context switch among

processes. To facilitate the processing of a common database, the driver has access to a shared memory segment initialized by the server.

The application that is linked to the solidDB server using the SMA driver is called an *SMA application*; the server is called *SMA server*.

SMA server

Before the first application is started with SMA, the solidDB server is initialized in a SMA mode by starting a small application (`solidsma`) that loads the SMA driver library dynamically. This SMA server application starts the server code internally and allocates and initializes a shared memory segment that the applications use to access the database.

The SMA server has the full functionality of the network server:

- The SMA server process takes care of all client-independent tasks: startup and recovery, checkpointing and logging, making backups, and so on.
- The solidDB configuration parameters, admin commands, and command line parameters can be used.
- Both in-memory and disk-based tables can be accessed equally.
- The SMA server can be used in solidDB Universal Cache, and with High Availability, Advanced Replication, and InfoSphere™ CDC Replication configurations.
- The SMA server can also be used as a regular network connection based server.

When the server is started in the SMA mode, it accepts connection requests from the SMA driver at the normal listening port. By assigning different port numbers to different SMA servers, it is possible to run several SMA servers on a single system simultaneously.

When the server is shut down, or all the users are thrown out, the applications receive a `Connection lost` error on the next request. If the application is waiting for a response during a forceful shutdown, it receives a shutdown notification.

Disk-based and diskless server

The SMA server can be disk-based or diskless. Diskless server can be useful in embedded systems that do not have hard disks, such as line cards in a network router or switch.

SMA and solidDB tools

The solidDB data management tools can be used with network-based connections to the SMA server.

SMA application

An existing ODBC or JDBC application does not need to be modified in order to use SMA, with the exception of a data source name or a connect string. For example, in an ODBC application, the connection is requested with the regular ODBC SQLConnect() call.

An existing LLA application can be changed into an SMA application or vice versa. An application can also be changed from a SMA application to a network-based application.

SMA driver

The SMA driver is a dynamic library that contains a complete copy of the solidDB server in library format. The applications can link to the SMA driver directly or using a driver manager.

The footprint of the driver's binary file corresponds to the full solidDB server, which is 3-6 MB, depending on the platform. However, because all applications link to the same driver, the in-memory footprint is not multiplicated when additional applications are added. The total memory footprint of the whole application system (applications, drivers, and the server) is comparable to the one of the client-server model.

SMA connection

Once the SMA server is running, applications can establish either SMA or network connections. For SMA connections, the applications have to be located on the same node as the server. The connection type is defined within the connect string. In the case of ODBC applications, if a driver manager is used, the SMA data source can be configured in the same way as an ODBC driver.

The connection request is sent over a network connection (handshake connection) using any locally available protocol (tcpip, named pipes, unix pipes). During the connect handshake, the shared-memory segment handle is passed to the driver so that it can access the server's shared memory segment.

System requirements for SMA

SMA is available on 64-bit platforms as well as 32-bit Linux®. For SMA with Java, Java Runtime Environment (JRE) 1.4.2 or Java Development Kit (JDK) 1.4.2 or newer is required.

Supported platforms for SMA

- AIX®
- HP-UX
- Linux 64-bit
- Linux 32-bit
- Solaris
- Windows 64-bit

For more details on the supported platforms, see the solidDB System Requirements on the solidDB Web site at http://www-01.ibm.com/software/data/soliddb/soliddb/sysreqs.html.

SMA components and packaging

The SMA driver library and SMA server application are included in the solidDB software package. For SMA with Java, the solidDB JDBC Driver is needed.

SMA driver (library)

The SMA driver libraries for most common platforms are shown in the table below:

Table 3. SMA drivers (libraries)

Platform	SMA driver library	Default installation location
Windows	`ssolidsmaxx.dll` **Note:** If you link to the SMA driver directly (without driver manager), you link to the `solidsma.lib` import library file that gives you access to the actual .dll library file.	Library: `<solidDB installation directory>\bin` Import library: `<solidDB installation directory>\lib`
Linux	`ssolidsmaxx.so`	`<solidDB installation directory>/bin`
Solaris	`ssolidsmaxx.so`	`<solidDB installation directory>/bin`
HP-UX	`ssolidsmaxx.sl`	`<solidDB installation directory>/bin`
AIX	`ssolidsmaxx.so`	`<solidDB installation directory>/bin`
`xx` is the version number of the driver library, for example, `ssolidsma65.so`.		

The SMA driver library for all platforms contains the following:

- full solidDB server functionality
- functions for three APIs
  - solidDB ODBC driver functions that allow for direct communication with the server library, without going through the network.
  - solidDB Control API (SSC API) library that contains functions to control starting and shutting down the SMA server.
  - solidDB SA API library which may be required for additional functionality using the linked library access. For example, this library allows you to insert, delete, and select records from a table.

  Since the library that your application links to contains three APIs, your application program may call functions from any combination of these APIs. For details on each of these APIs, see "solidDB APIs and drivers for SMA and LLA" on page 8.

For SMA use with Java, the solidDB JDBC Driver is needed; the solidDB JDBC Driver (`SolidDriver2.0.jar`) is installed during solidDB server installation into the `jdbc` directory in the solidDB installation directory.

SMA server application

Table 4. SMA server applications

Platform	SMA application
Windows	`solidsma.exe`
Linux	`solidsma`
Solaris	`solidsma`
HP-UX	`solidsma`
AIX	`solidsma`

Linked library access (LLA)

With the linked library access (LLA), an application links to a static library or a dynamic library 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 is linked to the solidDB server using the LLA library is called an *LLA application*; the server is called *LLA server*.

The application that links to the LLA library can also have multiple connections, using ODBC API, SA API, and JDBC API. All these APIs are reentrant, allowing simultaneous connections from separate threads.

The application that links directly to LLA library can also create remote connections to other database servers. The connection type (local or remote) is defined in the connect string that is passed to the ODBC API or SA API connect function or defined in the JDBC connection properties.

Principles of operation

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 implementations, the server runs on threads that are separate from the thread or threads 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.

Disk-based and diskless server

The solidDB server used with LLA can be disk-based or diskless. The LLA library contains two different function calls to start the server. The SSCStartServer function call starts a normal disk-based server, while SSCStartDisklessServer starts a server that does not use the disk drive.

System requirements for LLA

LLA is available on all platforms that solidDB supports. For LLA with Java, Java Runtime Environment (JRE) 1.4.2 or Java Development Kit (JDK) 1.4.2 or newer is required.

For a list of the supported platforms, see the solidDB System Requirements on the solidDB Web site at http://www-01.ibm.com/software/data/soliddb/soliddb/sysreqs.html.

LLA components and packaging

The LLA library is included in the solidDB software package. For LLA with Java, the JDBC driver and solidDB proprietary control classes are embedded in the solidDB JDBC Driver.

The LLA libraries for most common platforms are shown in the table below:

Table 5. LLA libraries

Platform	Static LLA library	Dynamic LLA library	Default location
Windows	`solidimpac.lib` This is an import library file that gives you access to the actual library file `ssolidacxx.dll`.	`ssolidacxx.dll`	Import library: `<solidDB installation directory>\lib` Library: `<solidDB installation directory>\bin`
Linux	`solidac.a`	`ssolidacxx.so`	`<solidDB installation directory>/bin`
Solaris	`solidac.a`	`ssolidacxx.so`	`<solidDB installation directory>/bin`
HP-UX	`solidac.a`	`ssolidacxx.sl`	`<solidDB installation directory>/bin`
`xx` is the version number of the dynamic library, for example, `ssolidac65.so`.			

The LLA library for all platforms contains the following:

- full solidDB server functionality
- functions for three separate APIs
  - solidDB Control API (SSC API) library that contains functions to control task scheduling.
  - solidDB ODBC Driver functions that allows for direct communication with the server library, without going through the network.
  - solidDB SA API library which may be required for additional functionality using the linked library access. For example, this library allows you to insert, delete, and select records from a table.

  Since the library that your application links to contains all three APIs (SSC, SA, and ODBC), your application program may call functions from any combination of these APIs. For details on each of these APIs, see "solidDB APIs and drivers for SMA and LLA" on page 8.

  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 "Configurations with local and remote applications types" on page 10. For information about API files for remote applications, read "solidDB APIs and drivers for SMA and LLA" on page 8.

For LLA use with Java, the solidDB JDBC driver is needed; the solidDB JDBC driver jar file (`SolidDriver2.0.jar`) contains the following packages:

- `solid.jdbc.*` solidDB JDBC driver classes
- `solid.ssc.*` solidDB Server Control classes (proprietary SSC API for Java interface)

Static vs. dynamic link libraries for LLA

solidDB provides both a *static* and a *dynamic* version of the linked library access library.

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 (for example, `lib/solidac.a`), you link your program directly to it, and 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 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 advanced replication master and replica on the computer at the same time, or you would 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 do not have multiple copies of LLA in memory at the same time.

On Microsoft Windows, the solidDB linked library access 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 linked library access functions in that `.dll` file. The `.dll` file must be in your load path when you run the program that references it.

Note: Using the dynamic link library file does not mean that you can have multiple LLA applications clients linked to a single 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. To enable multiple local applications to access solidDB, design your environment using shared memory access (SMA).

solidDB APIs and drivers for SMA and LLA

The SMA and LLA application requests are typically handled through ODBC API direct function calls or JDBC calls. The solidDB proprietary solidDB API is available also. The solidDB Server Control API (SSC API and SSC API for Java) is included in the LLA library for handling local requests to control solidDB background processes and client tasks. SMA includes limited support for SSC API; only calls for starting or stopping the SMA server are included.

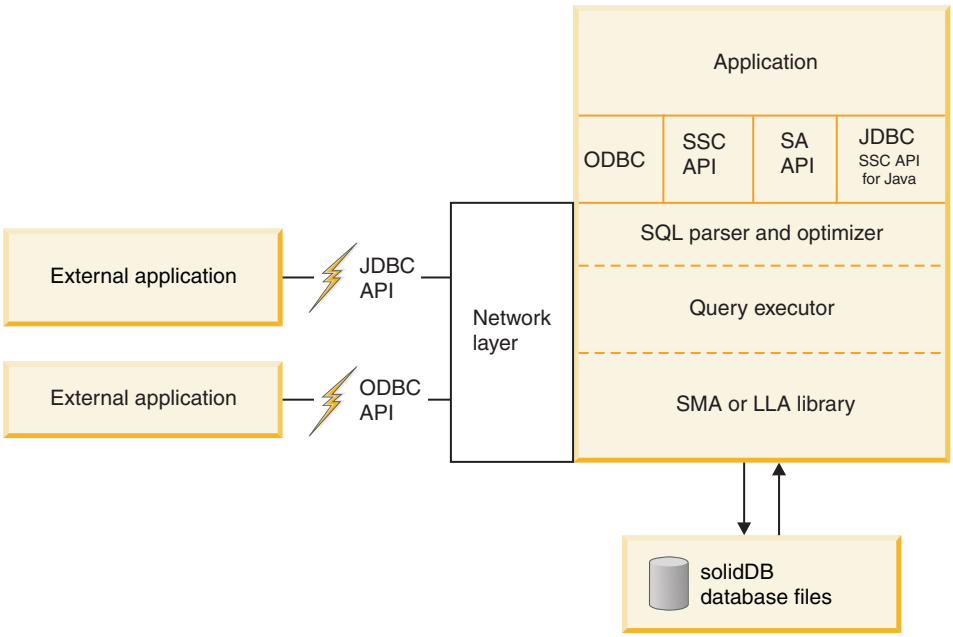


Figure 2. Example: SMA and LLA APIs for C/C++ programs

solidDB SA API

solidDB SA API is a low-level proprietary C-language API to solidDB data management services. The SA API provides support for local applications using SA API function calls.

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

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

Remote applications can also use the SA API function calls. However, you must link to a separate SA API library file (for example, `solidimpsa.lib` for Windows).

solidDB ODBC API

solidDB 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.

solidDB ODBC API is a Call Level Interface (CLI) for solidDB databases. It is compliant with ANSI X3H2 SQL CLI.

SMA and LLA support the ODBC 3.51 standard. The SMA and LLA libraries include solidDB ODBC 3.x, which provides support for local applications that require direct function calls to the server.

See *IBM solidDB Programmer Guide* for more details on the solidDB ODBC API.

solidDB JDBC API

The JDBC API defines Java classes to represent database connections, SQL statements, result sets, database metadata, and so on. It allows you to issue SQL statements and process the results. JDBC is the primary API for database access in Java.

SMA and LLA support both JDBC 1.x and 2.x.

The JDBC interface and the solidDB JDBC Driver, including descriptions of the solidDB specific enhancements are documented in *IBM solidDB Programmer Guide*.

solidDB Server Control API (SSC API)

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

The SSC API functions are included in the SMA driver and LLA library files. However, for SMA, only the functions for starting and stopping the server are supported.

LLA provides support for local applications using SSC API function calls and a separate library is available for remote-only applications.

If your LLA application runs remotely and contains SSC API function calls, then you must link to the SSC 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 from solidDB with LLA.

solidDB Server Control API (SSC API) for Java

The solidDB Server Control API (SSC API) for Java is a proprietary API, named after SolidServerControl class. The SSC API for Java calls are used to start and stop the LLA server. The actual database connections are done with normal solidDB JDBC API. Both the SSC API for Java classes and solidDB JDBC driver classes are included in the solidDB JDBC driver (`SolidDriver2.0.jar`).

The SolidServerControl classes for accessing solidDB server have been embedded inside solidDB JDBC driver file, inside the `solid.ssc` package. The solidDB JDBC driver jar file (`SolidDriver2.0.jar`) contains the following packages:

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

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

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

For more details, see Appendix E, “SolidServerControl class interface,” on page 81.

Configurations with local and remote applications types

With SMA and LLA, the applications always connect to a local solidDB server (SMA server or LLA server); the application and solidDB server are located on the same node. In addition to handling requests from this local SMA or LLA application, the SMA or LLA server can also handle requests from remote

applications, which connect to the server through communications protocols such as TCP/IP. A dual mode application can also be written; it switches modes between local and remote, depending upon how it is compiled and linked.

The SMA or LLA application is a *local application*; the server and the application are located on the same node. For example, calls to ODBC functions go directly to the server, rather than going through an ODBC driver and the communications protocol (such as TCP/IP).

A *remote application* is not linked to the SMA driver or LLA library. It is a separate executable that must communicate with the server using a network connection (such as TCP/IP) or other connection. Remote applications are usually run on a different node from the one that is running the server, but the application is also considered remote if it uses a network communication protocol to communicate with the server. A single node can run a local SMA or LLA application, while running one or more remote applications as separate processes.

The remote applications see the SMA and LLA server as similar to any other solidDB server, while the local application sees a faster, more precisely controllable version of the solidDB server.

Most applications are either local (linked to the SMA driver or LLA library) or remote (never linked). However, it is also possible to write a *dual-mode application* that uses both local and network-based connections. For example, the application can use 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.

Dual-mode applications may be useful for example in the following cases:

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

The remote applications 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 LLA with Java, the remote client programs may use C, Java, or both.

SSC API and SA API libraries for remote applications

Remote applications that contain SSC API or SA API function calls must link to separate libraries.

Table 6. SSC API and SA API libraries for remote applications

Platform	SSC API stub library	SA API library	Default location
Windows	`solidctrlstub.lib`	`solidimpsa.lib`	`<solidDB installation directory>\lib`
Other platforms	`solidctrlstub.a`	`solidsa.a`	`<solidDB installation directory>/bin`

The SSC API stub library is required for remote applications because the SSC API functions included in the SMA and LLA libraries cannot be used with remote applications. For example, assume you have a local application (containing SSC

API functions) that links to a standard ODBC library. You want to run the same application remotely. By linking to the SSC API stub library, you avoid having to remove the SSC API function calls from your code. In this way, you can easily turn your LLA or SMA application into a normal remote client application.

Note: The SSC API stub library contains "do-nothing" functions; if you call them in a remote application, they have no effect on the server. This means that the SSC API stub 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 from solidDB with LLA or SMA.

2 Creating and running SMA applications

To create SMA applications, you configure solidDB as necessary, link your applications to the SMA driver, start an SMA server, and establish a local connection between the applications and the server. After you have created your application, you can monitor the SMA performance using the monitoring features provided with solidDB.

Important: The instructions for creating and running SMA applications provide SMA specific additions, supplements, and usage differences in comparison to the solidDB without SMA.

For information about solidDB SQL, solidDB data management tools, general solidDB administration and maintenance, and database error codes, refer to the *IBM solidDB Administrator Guide*.

For detailed information about the APIs and solidDB JDBC and ODBC Drivers, see *IBM solidDB Programmer Guide*.

Creating SMA applications - overview

To create an application that uses SMA, you must prepare your system for SMA use, configure solidDB, set your application to use the SMA driver, start the SMA server and connect your application to it.

About this task

This procedure provides an overview on how to create SMA applications. The SMA applications for C/ODBC environments are created in a similar way to applications that are not using SMA.

Note: When developing your application, it is recommended to use a network-based connection. Once your application is ready, switch to using the SMA connection.

For an example of a SMA application written in C, see the SMA sample in the `samples/sma` directory in your solidDB installation directory.

Procedure

1. **Check the system settings for shared memory use in your environment.**

   The default values for shared memory use in your environment may not be sufficient for using SMA. For details on viewing and setting the shared memory system parameters on your system, see "Modifying shared memory kernel parameters - overview" on page 14.
2. **Set up your database environment by creating a working directory, your solidDB database, and user accounts.**

   For instructions, see *Creating a new database* in the *IBM solidDB Administrator Guide*.

   Note: The application and the SMA server processes must have identical file access permissions (database files, log files and so on). The file access

permissions are not checked at startup; subsequently, insufficient file access permissions may cause the SMA server to crash at a later point.

3. **Configure solidDB to meet your environment, performance, and operation needs.**

   Use the `solid.ini` configuration file to define basic configuration settings such as database file names and locations, database block size and so on.

   - In normal setups, it is not necessary to modify the SMA-specific parameters in the `[SharedMemoryAccess]` section of the `solid.ini` file; the factory values are applicable to most use cases.
   - The parameter **Srv.ProcessMemoryLimit** is not effective with SMA servers.

   If there is no configuration file, the factory values are used.

4. **Prepare your application for SMA use.**

   You can set up your application to use SMA with or without a driver manager.

   - "Preparing applications for SMA use with driver manager" on page 20
   - "Preparing applications for SMA use without driver manager" on page 21

5. **Start the SMA server.**

   For instructions, see "Starting SMA server" on page 24.

6. **Start the application.**

Modifying shared memory kernel parameters - overview

Shared memory is allocated in segments. The shared memory system parameters control the maximum size and number of segments allowed on your system.

Typically solidDB uses 8 MB sizes.

The shared memory parameters and their management mechanisms depend on the system. In Linux and UNIX environments, you may need to address the type of kernel parameters described below.

Important: This section and the sections below discuss only the requirements set by solidDB. Other processes running on the same system may require higher limit values.

- Maximum size of a shared memory segment

  Typically you do not need to modify the default system setting. This is because the solidDB segment size of 8 MB is considerably small.

- Maximum number of shared memory segments in a system/process
  - Because solidDB allocates the majority of the segments in 8 MB, you may need more segments than allowed by your system by default, especially if you have a large database.

    The maximum number of shared memory segments should be at least the solidDB process size in MB divided by 8.

    For example, for a process size of 1 GB (1024 MB), at least 128 segments are needed.
  - You should always set the maximum number of segments to a clearly higher value than required by your database size; a higher value has no side effects.
  - solidDB uses only one process; if your environment requires you to set the maximum number of segments for a process and for the system separately, you can use the same value for both.
- Maximum total size of all shared memory segments

The total combined size of all shared memory segments depends on the size of your database and availability of disk space.

Note: In addition to this kernel parameter, the maximum total size of shared memory used by solidDB is controlled with the solidDB parameter **MaxSharedMemorySize** (in the `[SharedMemoryAccess]` section of the `solid.ini` file) as follows:

– The value set with the **MaxSharedMemorySize** parameter takes precedence over the value set with the kernel parameter. Because of this, the value set with the **MaxSharedMemorySize** parameter should never be higher than the value set with the kernel parameter.

– By default, solidDB is set to use the maximum size of the physical memory of the computer (**MaxSharedMemorySize=0**). Because of this, the default value set with the kernel parameter may be too low.

Example 1

If the system has 2 GB of memory and **MaxSharedMemorySize** is set to '0', solidDB will use maximum of 2 GB of memory. If the kernel parameter for maximum total size of all shared memory segments is then set to 1 GB, solidDB will run out of memory when the 1 GB is reached.

Example 2

If the system has 2 GB of memory and **MaxSharedMemorySize** is set to '500M', solidDB will never use more than 500 MB of memory. As long as the kernel parameter for maximum total size of all shared memory segments is set to 500 MB or higher, solidDB will never run out of memory. It is recommended to set the value to a higher value than the memory required by solidDB.

Shared memory kernel parameters for SMA on AIX

On AIX systems, the shared memory kernel parameters do not need to be modified. The upper limits are defined for the AIX IPC mechanisms, which are not configurable. The shared memory limits are allocated and deallocated dynamically as needed, so that the memory requirements always depend on the current system usage.

Important: Do not set the page space allocation policy to *early page space allocation*. Instead, use the *deferred* (default) or *late* space allocation policy.

For more details, see the following sections in the IBM Systems Information Center (http://publib16.boulder.ibm.com/pseries/index.htm):

- Inter-Process Communication (IPC) Limits – Shared memory default limits and the IPC mechanisms
- Page space allocation – Page space allocation policies

Modifying shared memory kernel parameters for SMA on HP-UX

The default values for shared memory kernel parameters on HP-UX may not be sufficient for running a SMA application. The kernel parameter values can be changed dynamically using the `kctune` command.

Before you begin

You must have root authority to modify shared memory kernel parameters.

About this task

The following steps show how to set shared memory kernel parameters on HP-UX. The minimum values that are shown according to the requirements set by solidDB. Other processes running on the same system may require higher limit values.

In HP-UX environments, you may need to modify the following shared memory kernel parameters:

- **shmmni** — Maximum number of shared memory segments on the system
- **shmseg** — Maximum number of shared memory segments attached to a process
- **shmmax** — Maximum size of a single shared memory segment (bytes)

Procedure

1. **View the shared memory kernel parameters to determine if there are any necessary changes required for your system.**

   View the **shmmni** parameter:

```
kctune -v shmmni
Tunable             shmmni
Description         Maximum number of shared memory segments on the system
Module              vm_asi
Current Value       400 [Default]
Value at Next Boot  400 [Default]
Value at Last Boot  400
Default Value       400
Constraints         shmmni >= 3
                    shmmni <= 32768
                    shmmni >= shmseg
Can Change          Immediately or at Next Boot
```

   View the **shmseg** parameter:

```
kctune -v shmseg
Tunable             shmseg
Description         Maximum number of shared memory segments attached to a process
Module              vm_asi
Current Value       300 [Default]
Value at Next Boot  300 [Default]
Value at Last Boot  300
Default Value       300
Constraints         shmseg >= 1
                    shmseg <= shmmni
Can Change          Immediately or at Next Boot
```

   View the **shmmax** parameter:

```
kctune -v shmmax
Tunable             shmmax
Description         Maximum size of a shared memory segment (bytes)
Module              vm_asi
Current Value       1073741824 [Default]
Value at Next Boot  1073741824 [Default]
Value at Last Boot  1073741824
Default Value       1073741824
Constraints         shmmax >= 2048
                    shmmax <= 4398046511104
Can Change          Immediately or at Next Boot
```

Table 7. Minimum requirements of shared memory kernel parameters for SMA (HP-UX)

Parameter	Description	When to modify	Notes
shmmni	Maximum number of shared memory segments on the system	If the value is smaller than the solidDB process size (MB) divided by 8 For example, for a process size of 1 GB (1024 MB), at least 128 segments are needed.	You should always set this parameter to a clearly higher value than required by your database size; a higher value has no side effects.

Table 7. Minimum requirements of shared memory kernel parameters for SMA (HP-UX) (continued)

Parameter	Description	When to modify	Notes
shmseg	Maximum number of shared memory segments attached to a process	If the value is smaller than the solidDB process size (MB) divided by 8 For example, for a process size of 1 GB (1024 MB), at least 128 segments are needed.	Because solidDB uses only one process, the value of **shmmni** and **shmseg** can be the same.
shmmax	Maximum size of a single shared memory segment (bytes)	If the value is smaller than 32768 KB (32 MB)	Setting this parameter to a higher value has no side effects.

2. **To modify the parameters, use the `kctune` command.**

   For example, to set the maximum number of shared memory segments to 1024, use the following command:

   ```
   kctune shmmni=1024
   ```

   The parameter value change becomes effective immediately and stays effective after reboot.

What to do next

If you modified the shared memory parameters after getting an `out of memory` error, you may need to clear hanging shared memory segments with the `ipcrm` command. For more details, see "Troubleshooting SMA" on page 27.

Modifying shared memory kernel parameters for SMA on Linux

The default values for shared memory kernel parameters on Linux may not be sufficient for running a SMA application. To modify the shared memory kernel parameters on Linux, edit the `/etc/sysctl.conf` file.

Before you begin

You must have root authority to modify kernel parameters.

About this task

The following steps show how to update kernel parameters on Red Hat and SUSE Linux using the shared memory requirements set by solidDB. Other processes running on the same system may require higher limit values.

In Linux environments, you may need to modify the following shared memory parameters:

- **SHMMNI** — Maximum number of shared memory segments on a system
- **SHMMAX** — Maximum size of a single shared memory segment on a system
- **SHMALL** — Maximum allocation of shared memory pages on a system

Procedure

1. **Run the `ipcs -l` command.**

   For example:

Note: Comments have been added following the // to show what the parameter names are.

```
# ipcs -l

------ Shared Memory Limits --------
max number of segments = 4096               // SHMMNI
max seg size (kbytes) = 32768               // SHMMAX
max total shared memory (kbytes) = 8388608  // SHMALL
```

2. **Analyze the output to determine if there are any necessary changes required for your system.**

Table 8. Minimum requirements of shared memory kernel parameters for SMA (Linux)

Kernel parameter	Description	When to modify	Notes
SHMMNI	Maximum number of shared memory segments on a system	If the value is smaller than the solidDB process size (MB) divided by 8 For example, for a process size of 1 GB (1024 MB), at least 128 segments are needed.	You should always set this parameter to a clearly higher value than required by your database size; a higher value has no side effects.
SHMMAX	Maximum size of a single shared memory segment on a system	If the value is smaller than 32768 KB (32 MB)	Setting this parameter to a higher value has no side effects. **Note:** The `ipcs` output has converted SHMMAX kilobytes. The kernel requires the SHMMAX value in bytes.
SHMALL	Maximum allocation of shared memory pages on a system	If **MaxSharedMemorySize=0** and the value of this parameter is smaller than the maximum size of the physical memory size of your computer in KB divided by 4. or If the value of this parameter is smaller than the value (in KB divided by 4) you have set with the parameter **MaxSharedMemorySize**.	• The value set with the **MaxSharedMemorySize** parameter takes precedence over the value set with the kernel parameter. Because of this, the value set with the **MaxSharedMemorySize** parameter should never be higher than the value set with the kernel parameter. • By default, solidDB is set to use the maximum size of the physical memory of the computer (**MaxSharedMemorySize=0**). Because of this, the default value set with the kernel parameter may be too low. **Note:** The `ipcs` output has converted SHMALL into kilobytes. The kernel requires the SHMALL value as a number of pages.

3. **To modify these kernel parameters, edit the `/etc/sysctl.conf` file.**

   If the file does not exist, create it.

   The following lines are examples of what should be placed into the file:

```
#Example shmmni for a 1 GB database
kernel.shmmni=400
#Example shmmax for a 64-bit system
kernel.shmmax=1073741824
#Example shmall for 16 GB memory
kernel.shmall=3774873
```

4. **Run `sysctl` with `-p` parameter to load in `sysctl` settings from the default file `/etc/sysctl.conf`.**

```
sysctl -p
```

5. **Make the changes effective after every reboot.**
   - In SUSE Linux: make `boot.sysctl` active.
   - In Red Hat Linux: the `rc.sysinit` initialization script will read the `/etc/sysctl.conf` file automatically.

What to do next

If you modified the shared memory parameters after getting an `out of memory` error, you may need to clear hanging shared memory segments with the `ipcrm` command. For more details, see "Troubleshooting SMA" on page 27.

Modifying shared memory kernel parameters for SMA on Solaris

The default values for shared memory kernel parameters on Solaris 10 may not be sufficient for running a SMA application. In Solaris 10, the shared memory kernel parameter values can be changed dynamically with the Solaris resource control facilities.

Before you begin

You must have root authority to modify shared memory parameters.

About this task

The following steps show how to set shared memory kernel parameters on Solaris 10. The minimum values that are shown according to the requirements set by solidDB. Other processes running on the same system may require higher limit values.

In Solaris environments, you may need to modify the following shared memory parameters:

- **max-shm-ids** — Maximum number of shared memory segments on a system
- **max-shm-memory** — Maximum size of all shared memory segments on a system (MB)

In the examples below, the operating system default project is used.

Procedure

1. **View the shared memory parameters to determine if there are any necessary changes required for your system.**

   View the **project.max-shm-ids** parameter:

```
prctl -n project.max-shm-ids -i project default
project: 3: default
NAME    PRIVILEGE       VALUE    FLAG   ACTION          RECIPIENT
project.max-shm-ids
        privileged        128       -   deny                    -
        system          16.8M     max   deny                    -
```

   View the **project.max-shm-memory** parameter:

```
prctl -n project.max-shm-memory -i project default
project: 3: default
NAME    PRIVILEGE       VALUE    FLAG   ACTION          RECIPIENT
project.max-shm-memory
        privileged      62.7GB       -   deny                    -
        system          16.0EB     max   deny
```

Table 9. Minimum requirements of shared memory kernel parameters for SMA (Solaris)

Parameter	Description	When to modify	Notes
max-shm-ids	Maximum number of shared memory segments on a system	If the value is smaller than the solidDB process size (MB) divided by 8 For example, for a process size of 1 GB (1024 MB), at least 128 segments are needed.	You should always set this parameter to a clearly higher value than required by your database size; a higher value has no side effects.

Table 9. Minimum requirements of shared memory kernel parameters for SMA (Solaris) (continued)

Parameter	Description	When to modify	Notes
max-shm-memory	Maximum size of all shared memory segments on a system	If **MaxSharedMemorySize=0** and the memory size set with this parameter is smaller than the maximum size of the physical memory size of your computer. or If the memory size set with this parameter is smaller than the memory size you have set with the parameter **MaxSharedMemorySize**.	Setting this parameter to a higher value has no side effects. • The value set with the **MaxSharedMemorySize** parameter takes precedence over the value set with the kernel parameter. Because of this, the value set with the **MaxSharedMemorySize** parameter should never be higher than the value set with the kernel parameter. • By default, solidDB is set to use the maximum size of the physical memory of the computer (**MaxSharedMemorySize=0**). Because of this, the default value set with the kernel parameter may be too low.

2. **To modify the parameters, use the `prctl` command.**

   For example, to set the maximum number of shared memory segments to 1024, use the following command:

   ```
   prctl -n project.max-shm-ids -r -v 1024 -i project default
   ```

3. **Make the changes effective after every reboot.**

   ```
   /usr/sbin/projmod -sK "project.max-shm-ids=(privileged,1024,deny)" default
   ```

What to do next

If you modified the shared memory parameters after getting an `out of memory` error, you may need to clear hanging shared memory segments with the `ipcrm` command. For more details, see "Troubleshooting SMA" on page 27.

Preparing applications for SMA use with driver manager

When using SMA with a driver manager, you connect to a SMA data source in a similar way as when connecting to a solidDB ODBC data source.

About this task

The SMA driver library file is installed during the solidDB installation. The table below lists the file names and their default installation locations for the most common platforms.

Table 10. SMA drivers (libraries)

Platform	SMA driver library	Default installation location
Windows	`ssolidsmaxx.dll` **Note:** If you link to the SMA driver directly (without driver manager), you link to the `solidsma.lib` import library file that gives you access to the actual .dll library file.	Library: `<solidDB installation directory>\bin` Import library: `<solidDB installation directory>\lib`
Linux	`ssolidsmaxx.so`	`<solidDB installation directory>/bin`
Solaris	`ssolidsmaxx.so`	`<solidDB installation directory>/bin`
HP-UX	`ssolidsmaxx.sl`	`<solidDB installation directory>/bin`
AIX	`ssolidsmaxx.so`	`<solidDB installation directory>/bin`

Table 10. SMA drivers (libraries) (continued)

Platform	SMA driver library	Default installation location
xx is the version number of the driver library, for example, ssolidsma65.so.		

Procedure

1. **Connect to the SMA data source.**

   Use the SMA-specific connect string when defining the data source connection information.

   The connect string syntax for a SMA connection is:

   ```
   sma <protocol name> <port number or pipe name>
   ```

   For example:

   ```
   sma tcp 2315
   ```

2. **Check the use of signal handlers.**

   Signal handlers are used to report the occurrence of an exceptional event to the application. The SMA driver installs by default its own signal handler that can help the SMA system to survive the most common application failures, such as killing or interrupting the applications from outside. Upon the capture of certain signals, the signal handler closes the SMA connections safely and exits the SMA application. This means that in most cases, the SMA server continues to run despite abnormal application exits.

   By default, the SMA driver handles the following signals that can cause the SMA connection to break:

   - Linux and UNIX: SIGINT, SIGTERM
   - Windows: SIGINT

   You can modify the set of signals that the SMA driver handles with the client-side parameter **SharedMemoryAccess.Signals**. You can also disable the SMA driver signal handler by setting the client-side parameter **SharedMemoryAccess.SignalHandler** to 'no'.

   If the **SharedMemoryAccess.SignalHandler** parameter is set to 'yes' (default), do not set signal handlers in your application for those signals that are handled by the SMA driver; the application setting will override the SMA driver settings.

Related reference

Appendix A, “Shared memory access parameters,” on page 51

Preparing applications for SMA use without driver manager

When using SMA without a driver manager, you link your application to the SMA driver library directly, in a similar way as when linking directly to an solidDB ODBC driver library.

Procedure

1. **Link your application to the SMA driver library.**

   The SMA driver library files are installed during the solidDB installation. The table below lists the file names and their default installation locations for the most common platforms.

Table 11. SMA drivers (libraries)

Platform	SMA driver library	Default installation location
Windows	`ssolidsmaxx.dll` **Note:** If you link to the SMA driver directly (without driver manager), you link to the `solidsma.lib` import library file that gives you access to the actual .dll library file.	Library: `<solidDB installation directory>\bin` Import library: `<solidDB installation directory>\lib`
Linux	`ssolidsmaxx.so`	`<solidDB installation directory>/bin`
Solaris	`ssolidsmaxx.so`	`<solidDB installation directory>/bin`
HP-UX	`ssolidsmaxx.sl`	`<solidDB installation directory>/bin`
AIX	`ssolidsmaxx.so`	`<solidDB installation directory>/bin`
`xx` is the version number of the driver library, for example, `ssolidsma65.so`.		

2. **Change the connect string to the local SMA server name.**

   The connect string syntax for a SMA connection is:

   ```
   sma <protocol name> <port number or pipe name>
   ```

   For example:

   ```
   sma tcp 2315
   ```

   For examples of the connect string when using ODBC API or SA API, see "Establishing local connections for SMA."

3. **Check the use of signal handlers.**

   Signal handlers are used to report the occurrence of an exceptional event to the application. The SMA driver installs by default its own signal handler that can help the SMA system to survive the most common application failures, such as killing or interrupting the applications from outside. Upon the capture of certain signals, the signal handler closes the SMA connections safely and exits the SMA application. This means that in most cases, the SMA server continues to run despite abnormal application exits.

   By default, the SMA driver handles the following signals that can cause the SMA connection to break:

   - Linux and UNIX: SIGINT, SIGTERM
   - Windows: SIGINT

   You can modify the set of signals that the SMA driver handles with the client-side parameter **SharedMemoryAccess.Signals**. You can also disable the SMA driver signal handler by setting the client-side parameter **SharedMemoryAccess.SignalHandler** to 'no'.

   If the **SharedMemoryAccess.SignalHandler** parameter is set to 'yes' (default), do not set signal handlers in your application for those signals that are handled by the SMA driver; the application setting will override the SMA driver settings.

Related reference

Appendix A, "Shared memory access parameters," on page 51

Establishing local connections for SMA

For SMA use, the application needs to establish a local SMA connection with the SMA server. The connection type is defined with a SMA-specific connect string.

For SMA, the connection request is sent over the network connection (handshake connection) using any locally available protocol (tcpip, named pipes, unix pipes). During the connect handshake, the shared-memory segment handle is passed to the driver so that it can access the server's shared memory.

The connect string syntax for a SMA connection is:

```
sma <protocol name> <port number or pipe name>
```

For example:

```
sma tcp 2315
```

If an SMA connect request is made to the server that is not an SMA Server, a connect error is returned.

Important: A single application can connect to only one SMA server. However, the SMA application may make regular network-based connections to any local or remote server.

ODBC API

When using the ODBC API, define the SMA-specific connect string in the SQLConnect function call.

Examples

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

```
rc = SQLConnect(hdbc, "sma tcp 1315", (SWORD)SQL_NTS, "dba", 3, "dba", 3);
```

or

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

SA API

When using the SA API, define the SMA-specific connect string in the SaConnect function call.

The following code examples connect directly to the local SMA solidDB server with username dba and password dba:

```
SaConnectT* sc = SaConnect("sma tcp 1315", "dba", "dba");
```

or

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

Driver manager

When using a Driver manager, define the SMA-specific connect string in the SMA Data Source.

Starting and shutting down SMA server

The SMA server is started, restarted and shutdown in a similar way as the normal network protocol based solidDB server.

Starting SMA server

The SMA server is started using the command prompt in the same way as with a normal network protocol based solidDB server. When SMA server is started, it checks if a database already exists. The server first looks for a `solid.ini` configuration file and reads the value of `FileSpec` parameter. If a database file is found with the names and paths specified in the `FileSpec` parameter, that database is opened automatically. If no database is found, the server prompts you to create a new database.

Procedure

To start the SMA server:

Table 12. Starting the SMA server

Operating system	To start the SMA server:
Linux and UNIX	Enter the command `solidsma` at the command prompt. When you start the server for the first time, enter the command `solidsma -f` at the command prompt to force the server to run in the foreground.
Windows	Enter the command `solidsma` at the command prompt. To start the SMA server as a service:

Results

When the server is started in the SMA mode, it loads the SMA driver library dynamically, accepting connection requests from the SMA driver at the normal listening port. By assigning different port numbers to different SMA servers, it is possible to run several SMA servers on a single system at the same time.

Tip: It is also possible to start the solidDB server in SMA mode by having the application call a SSC API function SSCStartSMAServer. However, in such a setup, only one application can start (and stop) the SMA server. For details on the SSC API calls, see "SSCStartSMAServer" on page 77.

Shutting down SMA server

SMA server is shutdown using the solidDB ADMIN COMMANDs.

Procedure

1. To prevent new connections to solidDB, close the database 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'
   ```

Results

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.

Tip: It is also possible to stop the solidDB server in SMA mode by having the application call a SSC API function SSCStopServer(). Only one application can start and stop the SMA server. The same application that started the SMA server must also perform the shutdown. For details, see "SSCStopServer" on page 78.

Starting SMA server as a service (Windows)

solidDB with SMA can be run as a service in Windows. The first time you want to run the SMA server as a service, you must install the service, that is, allow Windows to run the SMA server as a service. After that, you can start and stop the services with the Windows Service dialog or command prompt, or remove the services using solidDB command line options.

Before you begin

To be able to install and start services in some Windows environments (for example, Windows 2008 Server), you need to run the Windows command prompt with administrator rights.

1. In the **Start** menu, right-click **Command Prompt**.
2. Select **Run as administrator**.
3. Log in with an administrator account.

About this task

The first time you want to run the SMA server as a service, you must first install the service, and then start the service with the Windows Service dialog or command prompt.

Procedure

1. **Allow (install) Windows to run the SMA server as a service.**

   In the solidDB working directory, issue the following command:

   ```
   solidsma -s"install,<name>,<fullexepath> -c<working directory>[,autostart]"
   ```

   where

   `<name>` is the service name

   `<fullexepath>` is the full path for `solidsma.exe`

   `<working directory>` is the full path for solidDB working directory (where your `solid.ini` and license file are located)

   `[autostart]` is an optional parameter that sets the Startup Type of the service to *Automatic*, that is, the SMA server will run automatically as a service when Windows is started.

   Note:

   Regardless of the `[autostart]` parameter, the service is not started automatically at the time of install. For the first time, the service has to be started manually in the Windows Services dialog or command prompt. (See step 2 below.)

   Example 1

   The following command installs a service named SOLIDSMA (with Startup Type *Manual*) when the SMA server is installed into the directory `C:\soliddb` and the working directory is `C:\soliddb`.

```
solidsma -s"install,SOLIDSMA,C:\soliddb\bin\solidsma.exe -cC:\soliddb"
```

Example 2

The following command installs a service named SOLIDSMA (with Startup Type *Automatic*) when the SMA server is installed into the directory `C:\soliddb` and the working directory is `C:\soliddb`. The next time Windows is started, the SMA server will automatically run as a service.

```
solidsma -s"install,SOLIDSMA,C:\soliddb\bin\solidsma.exe -cC:\soliddb,autostart"
```

Tip:

Alternatively, you can create the service using the Windows command line utility `sc.exe`. In that case, to start the SMA server in a services mode, you must include the solidDB `-sstart` command line option in the command. For example:

```
sc create SOLIDSMA binPath= "c:\soliddb\bin\solidsma.exe -cC:\soliddb -sstart"
```

The `-sstart` command line option is required to remove the GUI-based interactions between the SMA server and the user. Programs running as a Windows service cannot use those.

2. **Start the service manually in the Windows Services dialog or command prompt.**
   - You can access the Windows Services dialog through Control Panel: **Control Panel → Administrative Tools → Services**.
   - In the command prompt, issue the following command:

     ```
     sc start <name>
     ```

     For example:

     ```
     sc start SOLIDSMA
     ```

Results

When running as an Windows service, solidDB with SMA will log warning and error messages to the Windows event log. These messages can be viewed from Windows by using the Event Viewer, available through Control Panel: **Control Panel → Administrative Tools → Event Viewer**. Messages are also logged to the `solmsg.out` file.

Monitoring SMA

solidDB includes means for monitoring and collecting data on the type and number of SMA connections as well as the SMA memory segment sizes.

- Use `ADMIN COMMAND 'userlist'` to print a list of the type of user connections (network client or SMA client).
- Use `ADMIN COMMAND 'report'` to print a list of the connections by the type.
- Check the `solmsg.out` file login entries for the type of the connections made.
- Use the performance counter `SMA connection count` to collect data on the number of SMA connections.
- Use the performance counter `SMA shared memory used` to collect data on the SMA memory segment size.

For details on using the ADMIN COMMANDs, the `solmsg.out`, and the performance counters, see *IBM solidDB Administrator Guide*.

Troubleshooting SMA

This section provides instructions and guidelines on how to prevent or troubleshoot common problems while configuring or using SMA.

Error: Server could not allocate shared memory segment by id -1

Symptoms

When trying to start a SMA server, the following type of error is displayed, and the SMA server cannot be started.

```
IBM solidDB process has encountered an internal error and is unable to
continue normally. Please report the following information to technical support.
SOLID Fatal error: Out of central memory when allocating buffer memory (size = 33554432)
Date: 2009-08-24 15:39:44
Product: IBM solidDB
Version: 99.99.0.0 Build 0096

[rd@bench12]~ ./solidsma -f -c .
Server could not allocate shared memory segment by id -1
```

Causes

The SMA server startup fails because there is no memory available. This situation can occur if:

- When a SMA application or solidDB crashes, they may leave shared memory allocated. Even if you shut down all SMA processes, the shared memory is still left reserved.
- You have allocated too little memory for SMA use.

This leads to a situation where all memory is used and you cannot start a SMA server any more.

Resolving the problem

In Linux and UNIX environments, clear the hanging shared memory segments with the `ipcrm` command.

For example in Linux environments, use the following script to identify and remove the unused shared memory segments.

```
#!/bin/sh

if [ $# -ne 1 ]
then
   echo "$0 user"
   exit 1
fi

for shm_id in $(ipcs -m|grep $1|awk -v owner=$1 ' { if ( owner == $3 ) {print $2} }')
do
  ipcrm -m $shm_id
done
```

For more details on the `ipcrm` command, see your operating system documentation.

Cannot map shared memory area

Symptoms

When trying to connect to a SMA server, the following type of error is displayed, and the connection fails.

```
cannot map shared memory area 1288077395 to 0x2b0029800000
Cannot connect to target database.
```

Causes

When started, the SMA starts attaching shared memory segments to an address space that is used by another process.

Resolving the problem

Before starting the SMA server, force the start address space for the SMA server to the solidDB default with an environment variable SOLSMASTART.

- Linux and UNIX operating systems:
  ```
  export SOLSMASTART=<default_start_address_space>
  ```
- Windows operating systems:
  ```
  set SOLSMASTART=<default_start_address_space>
  ```

The `<default_start_address_space>` depends on the operating system.

Table 13. SOLSMASTART default address spaces

Operating system	Default start address space*
AIX	0x700000010000000ul
Linux 64-bit	0x2c0000000000
Linux 32-bit	0x50000000
Solaris Intel	0xffffffff60000000
Solaris Sparc	0x2b0000000000
Windows	0x0000000080000000
*The start address space is the value of the parameter **shmaddr** in the shmat() system call.	

For example:

```
export SOLSMASTART=0x2c0000000000
```

Important: In addition to the start address spaces listed in the above tables, values close to the shown value may work.

For example, in Linux 64-bit operating systems, you could set the value to 0x2b0000000000 (instead of 0x2c0000000000).

3 Creating and running SMA applications with Java

Java applications are linked to the SMA driver library. The actual database connections are done with the normal JDBC API.

Overview of using SMA with Java

A Java application that uses SMA is created in the same way as an application that uses a regular solidDB server, with the exception that you start an SMA server instead of a regular solidDB server. The Java application connects to the SMA server and uses the services solidDB server provides through a standard JDBC API. Linking to the dynamic library allows the application to avoid the overhead of RPC (Remote Procedure Calls) through the network.

Java/JDBC programs that want to use SMA link to the SMA driver library (`ssolidsmaxx`). 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 libraries used with Java/JDBC are the same as the ones used with C/C++ applications; there are not separate versions for Java.

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

- SMA driver library,
- your Java-language client program, and
- the JVM.

The layers in the executable process are, from top to bottom:

- Local Java/JDBC client application
- JVM (Java Virtual Machine)
- SMA driver library

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 SMA driver library. The function call is direct, it does not go through the network (through RPC). The calls are made using Java Native Interface (JNI). 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.

Every application that uses SMA follows the same basic four-step pattern:

1. Configure the solidDB server and connection settings.
2. Start the SMA server.
3. Access the database by using normal JDBC API.
4. When database processing is done, stop the SMA server.

Configuring your environment for SMA use with Java

When using SMA with Java, your LD_LIBRARY_PATH or LIBPATH (Linux and UNIX) or PATH (Windows) environment variable must include the location of the SMA driver library.

Before you begin

It is assumed that you have a working installation of the solidDB JDBC Driver.

About this task

The SMA driver library file is installed during the solidDB installation. The table below lists the file names and their default installation locations for the most common platforms.

Table 14. SMA drivers (libraries)

Platform	SMA driver library	Default installation location
Windows	`ssolidsmaxx.dll` **Note:** If you link to the SMA driver directly (without driver manager), you link to the `solidsma.lib` import library file that gives you access to the actual .dll library file.	Library: `<solidDB installation directory>\bin` Import library: `<solidDB installation directory>\lib`
Linux	`ssolidsmaxx.so`	`<solidDB installation directory>/bin`
Solaris	`ssolidsmaxx.so`	`<solidDB installation directory>/bin`
HP-UX	`ssolidsmaxx.sl`	`<solidDB installation directory>/bin`
AIX	`ssolidsmaxx.so`	`<solidDB installation directory>/bin`
`xx` is the version number of the driver library, for example, `ssolidsma65.so`.		

Procedure

1. **Add the location of the SMA driver library to the LD_LIBRARY_PATH or LIBPATH (Linux and UNIX) or PATH (Windows) environment variable.**
   - In Linux and UNIX environments, use the following syntax:

     ```
     export LD_LIBRARY_PATH=<path to SMA library>:$LD_LIBRARY_PATH
     ```

     or

     in AIX environments:

     ```
     export LIBPATH=<path to SMA library>:$LIBPATH
     ```
   - In Windows environments, use the following syntax:

     ```
     set PATH=<path to SMA library>;%PATH%
     ```
2. **Set up your database environment by creating a working directory, your solidDB database, and user accounts.**

   For instructions, see *Creating a new database* in the *IBM solidDB Administrator Guide*.

   Note: The application and the SMA server processes must have identical file access permissions (database files, log files and so on). The file access permissions are not checked at startup; subsequently, insufficient file access permissions may cause the SMA server to crash at a later point.
3. **Configure solidDB to meet your environment, performance, and operation needs.**

   Use the `solid.ini` configuration file to define basic configuration settings such as database file names and locations, database block size and so on.
   - In normal setups, it is not necessary to modify the SMA-specific parameters in the `[SharedMemoryAccess]` section of the `solid.ini` file; the factory values are applicable to most use cases.

- The parameter **Srv.ProcessMemoryLimit** is not effective with SMA servers.

If there is no configuration file, the factory values are used.

Starting and shutting down SMA server

The SMA server is started, restarted and shutdown in a similar way as the normal network protocol based solidDB server.

Starting SMA server

The SMA server is started using the command prompt in the same way as with a normal network protocol based solidDB server. When SMA server is started, it checks if a database already exists. The server first looks for a `solid.ini` configuration file and reads the value of `FileSpec` parameter. If a database file is found with the names and paths specified in the `FileSpec` parameter, that database is opened automatically. If no database is found, the server prompts you to create a new database.

Procedure

To start the SMA server:

Table 15. Starting the SMA server

Operating system	**To start the SMA server:**
Linux and UNIX	Enter the command `solidsma` at the command prompt. When you start the server for the first time, enter the command `solidsma -f` at the command prompt to force the server to run in the foreground.
Windows	Enter the command `solidsma` at the command prompt. To start the SMA server as a service:

Shutting down SMA server

SMA server is shutdown using the solidDB ADMIN COMMANDs.

Procedure

1. To prevent new connections to solidDB, close the database 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'
   ```

Making JDBC connections for SMA

In order to make a local (non RPC-based) JDBC connection to the SMA server, you need to define that you are connecting to a SMA server, using the solidDB specific connection property `solid_shared_memory`, and that your are using a local server at a given port.

Connecting with Driver Manager

Include the non-standard connection property `solid_shared_memory` in your code.

For example:

```
Properties props = new Properties();
// enable the direct access property
props.put("solid_shared_memory", "yes");
// get connection
Connection c = DriverManager.getConnection
("jdbc:solid://localhost:1315", props);
```

Defining connection property in the connect string

Include the connection property in the connect string.

For example:

```
Connection c = DriverManager.getConnection
("jdbc:solid://localhost:1315?solid_shared_memory=yes");
```

Note: In addition to the `DriverManager` class, a similar syntax is available for classes `SolidDataSource` and `SolidConnectionPoolDataSource`.

4 Creating and running LLA applications

Creating LLA applications includes linking your application to the library, starting the server, and establishing a local connection between the application and the server. You can start and stop the server with the SSC API, ODBC API, and SA API.

The instructions LLA specific additions, supplements, and usage differences in comparison to the solidDB without LLA.

For information about solidDB SQL, solidDB data management tools, general solidDB administration and maintenance, and database error codes, refer to the *IBM solidDB Administrator Guide*.

For detailed information about the APIs and solidDB JDBC and ODBC Drivers, see *IBM solidDB Programmer Guide*.

Configuring your environment for LLA use

When using LLA, you must link your application to the LLA library file.

Procedure

1. **Link your application to the LLA library file specific to your operating system.**

Table 16. Linked library access (LLA) system libraries

Platform	LLA Library
Windows	`solidimpac.lib` This is an import library file that gives you access to the actual library file `ssolidacxx.dll`.
Solaris	`solidac.a`
HP-UX	`solidac.a`
Linux	`solidac.a`

Example: Makefile for providing the LLA library name in Windows

In the makefile example below, the `solidimpac.lib` library is used.

```
# 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
```

2. **If you do not plan to use the implicit start method for starting the solidDB server with SSC API, set the ImplicitStart parameter to 'no'.**

   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 in the same same. When the SQLConnect or SaConnect is called for the first time, the server is implicitly started.

3. **Disable 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 linked library access. For example, if the user application sets a signal handler for floating point exceptions, that setting overrides the handler set by the linked library access. Thus the server is unable to catch, for example, division by zero.

Establishing a local connection for LLA

Once an application is linked to the linked library access 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 linked library access.

In the ODBC API, to establish a connection to a local server (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 linked library access to use a "remote" connection (to go through the network rather than to use the direct function calls to the linked library access 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 linked library access to use a "remote" connection (to go through the network rather than to use the direct function calls to the linked library access 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");
```

Starting and shutting down LLA server

You can start up, restart, and shut down the LLA server using the SSC API, ODBC API, or SA API function calls. The ODBC API and SA API function calls can be used to start the server only if a database exists already. The SSC API can be used to create a new database at the startup.

At server startup, 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.

Explicit startup and shutdown with SSC API

The SSC API is used to start and shut down the LLA server *explicitly*. The application calls the SSC API function SSCStartServer to start solidDB and SSCStopServer to shut it down.

When you start a new LLA server that does not already have a database, you must specify explicitly that a new database is created by giving the following parameters with the SSCStartServer() function:

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

Note: If you want to start a diskless server, you must start the server with SSC API function SSCStartDisklessServer.
For details, see “Explicit startup with SSC API function SSCStartServer” on page 36.

Implicit startup and shutdown with ODBC API and SA API

The ODBC API and SA API can only be used to start and shut down the LLA server *implicitly*. When the application connects locally to LLA server for the first time, it calls the 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 the LLA server is started implicitly from the application, it checks if a database already exists in the solidDB working 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 with the appropriate parameters, or create a database as for a normal solidDB server.

For details, see “Implicit startup with ODBC API function call SQLConnect” on page 37 and “Implicit startup with SA API function call SaConnect” on page 38.

For instructions on how to create a database in a normal solidDB setup, see section *Creating a new database* in the *IBM solidDB Administrator Guide*.

Explicit startup with SSC API function SSCStartServer

To start solidDB explicitly, have the user application call the solidDB Server Control API function SSCStartServer().

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

where the parameters are:

Table 17. 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: you create an SSCServerT *, which is a pointer to a handle (essentially 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 value for this parameter is a combination of two flags: the open flag and the netcopy disabling flag. The following flag symbols can be used:
	• SSC_STATE_OPEN – the open flag is set to 1: new connections are allowed.
	• SSC_STATE_CLOSED – the open flag is set to 0: all new network and LLA connections are rejected, with the exception of connections from solidDB Remote Control (solcon) program.
	• SSC_DISABLE_NETCOPY – the netcopy disabling flag is set to 1: in HotStandby configuration, no netcopy can be received by the server for which this is set.
	The flag does not prevent the server to act as a source of netcopy. If only SSC_DISABLE_NETCOPY flag is set, the server is in the closed state. To enable netcopy, use the SSC API function SSCSetState() with the runflag value SSC_STATE_OPEN or SSC_STATE_CLOSED.
	`runflags = SSC_STATE_OPEN | SSC_STATE_CLOSED | SSC_DISABLE_NETCOPY`
	
	Tip: The flags can be used in combinations, for example:
	`...`
	`rc = SSCStartServer(g_argc, g_argv, &hh, SSC_STATE_OPEN|SSC_DISABLE_NETCOPY);`
	`...`
	
	If the server is started as closed, it can be opened with ADMIN COMMAND 'open', or with the solcon command `open`. The same effect can be achieved with the SSC API function SSCSetState().

Starting LLA server without an existing database

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 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.

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. 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.

Starting LLA server with an existing database

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

Implicit startup 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 “Explicit startup with SSC API function SSCStartServer” on page 36.

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
```

Implicit startup 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 "Explicit startup with SSC API function SSCStartServer" on page 36.

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.

Shutting down LLA server

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

Programmatically, you can perform the shut down from an application such as solidDB SQL Editor (solsql), or solidDB 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 the explicit method SSCStopServer to shut down a server that was started with implicit methods (SQLConnect). The converse is not true; for example, you cannot use SQLDisconnect to stop a server that was started with SSCStartServer.

Shutting down LLA server 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);
```

Sample C applications for LLA

The solidDB package includes samples of LLA applications written in C that use ODBC API functions to connect to solidDB servers.

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

The samples are located in the following directories in the solidDB installation directory:

- `samples/aclib`: sample of a LLA application using a single solidDB
- `samples/aclib_control_api`: sample of a LLA application that uses the SSC API functions.
- `samples/aclib_diskless`: sample of a LLA application using a diskless solidDB server
- `samples/aclib_replication`: sample of a LLA application that combines LLA and advanced replication

  See "Samples for using LLA with advanced replication" for information on how to use the replication samples.

Each directory contains a `readme.txt` file that provides instructions on how to set up your system and run the samples.

Samples for using LLA with advanced replication

If you are new to solidDB data synchronization, *IBM solidDB Advanced Replication User Guide* contains information about how to use the sample scripts provided with solidDB.

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

- Use solidDB without SMA or LLA to run the SQL scripts contained in *IBM solidDB Advanced Replication User Guide*. These scripts are found in `samples/replication`.
- Run the SQL scripts locally, using the solidDB SMA or LLA. As a prerequisite, you are required to set up an application to start the server according to the instructions in this document.

Note: You cannot use the SA API to run synchronization commands.

- Running the implementation sample file `aclibstandalone.c`, which with the linked library access 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 *IBM solidDB Advanced Replication User Guide*'s chapter titled *Getting Started with Data Synchronization* using solidDB SQL Editor (solsql).

Setting up your ODBC application with the advanced replication sample 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_replication`.

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/replication/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 solidDB SQL Editor (solsql).

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

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

5 Creating and running LLA applications with Java

Java applications are linked to the LLA library and they use solidDB SSC API calls to start and stop the solidDB. The actual database connections are done with the normal JDBC API. Both the SolidServerControl API calls and JDBC driver are included in the solidDB JDBC Driver .jar file (`SolidDriver2.0.jar`).

Overview of using LLA with Java

LLA enables Java applications to start a local solidDB server, which will be loaded into the Java Virtual Machine context from a dynamic library. The Java application will then be able to connect to the solidDB server and use the services solidDB server provides through a standard JDBC API. Linking to the dynamic library allows the application to avoid the overhead of RPC (Remote Procedure Calls) through the network.

Java/JDBC programs that want to use LLA link to the LLA 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 libraries used with Java/JDBC are the same as the ones used with C/C++ applications; there are not separate versions for Java.

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

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

The layers in the executable process are, from top to bottom:

- Local Java/JDBC client application
- JVM (Java Virtual Machine)
- LLA library

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, it does not go through the network (through RPC). The calls are made using Java Native Interface (JNI). 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 with LLA is identical to accessing a solidDB database through RPC — with one exception: in order to access the database services, the application using LLA must first start the LLA server. This is done with a proprietary API called solidDB Server Control (SSC) API for Java (named after SolidServerControl class. The actual database connections are done with normal solidDB JDBC API. Both the SSC API for Java and solidDB JDBC driver can be found in the .jar file named `SolidDriver2.0.jar`.

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

Every application that uses SMA or LLA follows the same basic four-step pattern:

1. Configure the solidDB and connection settings.
2. Start the LLA server with SolidServerControl class.
3. Access the database by using normal JDBC API.
4. When database processing is done, stop the LLA server again with SolidServerControl class.

Limitations

- All solidDB '`admin commands`' cannot be used when using LLA with Java.
- Java does not 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 abnormally or hangs up completely. In the latter case, you may have to kill the dangling Java process manually.
- To minimize memory consumption, you should drop all allocated statements explicitly; all allocated JDBC statement objects must be explicitly freed by calling the close() method. This is important especially if your setup is using Transparent Connectivity (TC).

Configuring your environment for LLA use with Java

When using LLA with Java, your LD_LIBRARY_PATH or LIBPATH (Linux and UNIX) or PATH (Windows) environment variable must include the location of the LLA library.

Before you begin

It is assumed that you have installed and registered the solidDB JDBC Driver.

Procedure

1. **Add the location of the LLA library to the LD_LIBRARY_PATH or LIBPATH (Linux and UNIX) or PATH (Windows) environment variable.**
   - In Linux and UNIX environments, use the following syntax:

     ```
     export LD_LIBRARY_PATH=<path to LLA library>:$LD_LIBRARY_PATH
     ```

     or

     in AIX environments:

     ```
     export LIBPATH=<path to LLA library>:$LIBPATH
     ```
   - In Windows environments, use the following syntax:

     ```
     set PATH=<path to LLA library>=;%PATH%
     ```
2. **Set up your database environment by creating a working directory, your solidDB database, and user accounts.**

   For instructions, see *Creating a new database* in the *IBM solidDB Administrator Guide*.

   Note: The application and the SMA server processes must have identical file access permissions (database files, log files and so on). The file access

permissions are not checked at startup; subsequently, insufficient file access permissions may cause the SMA server to crash at a later point.

Starting and stopping LLA server with SSC API for Java

To start a solidDB server from a Java application, you must instantiate the class SolidServerControl in the beginning of your application and call the `ssc.startServer` method with correct parameters. After starting the server, you are ready to make a JDBC connection to the server. Similarly, the server is stopped with the call `ssc.stopServer`.

Procedure

1. **Starting the server**
   - **LLA server:** `ssc.startServer`

   When starting the server, you must pass the solidDB server at least the following parameters:

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

   Note that upper and lower case "C" are both used, and they mean different things.
2. **Stopping the server**
   - **LLA server:** `ssc.stopServer`

Making JDBC connections for LLA

Linked library access (LLA) with Java 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.

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

If you are making the database connection by using, for example, JDBC class DriverManager, connect by using the following statement:

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

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 try to connect with RPC.

Compiling and running a sample LLA program

About this task

The examples in this procedure are given for Windows command prompt.

Procedure

1. Set the paths.

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

   Make sure you have the directory containing solidDB 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. Compile the sample `SJASample.java` file (located in the `samples/aclib_java` directory) with the following command:

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

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

   ```
   java -Djava.library.path=<path to ssolidacxx DLL> \ -classpath
   <IBM solidDB JDBC driver directory>/SolidDriver2.0.jar;. \ SJAsample
   ```

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

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

   On 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 solidDB license file>
   -U<username>
   -P<password>
   -C<catalog>
   ```

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

5. If you have all the necessary files (`ssolidacxx` library, communication libraries, JDBC driver and license file 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;. SJAsample
   ```

Results

Your LLA server is up and running.

6 Using the diskless capability

SMA and LLA servers can be used 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 ways to run a diskless server: as a single server (alone) or as a replica in an advanced replication system. In both cases, you need to start the server by using the SSC API or SSC API for Java function calls.

SSC API

- Use SSCStartDisklessSMAServer to start a diskless SMA server.
- Use SSCStartDisklessServer to start a diskless LLA server.

SSC API for Java

- Use startDisklessServer to start a diskless LLA server.

Diskless server Alone

If you run a diskless server alone, 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.

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 solidDB HotStandby component to create a 'hot standby' machine that contains a copy of the data. For more information about the hot standby capabilities, see *IBM solidDB High Availability User Guide*.

Diskless server as part of advanced replication systems

A diskless server may be a replica in a advanced replication 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 advanced replication system.

7 Creating and running remote or dual-mode applications

The SMA and LLA server can be accessed by remote applications. If your remote application includes SSC API or SA API function calls, you must link to separate SSC API and SA API libraries since the functions included in the SMA and LLA libraries cannot be accessed by remote applications. If your remote application only uses ODBC or JDBC, you can build you applications normally using the ODBC and JDBC interfaces. The remote connection type is defined in the connect string.

Example: Creating a dual-mode LLA application with ODBC and SSC API function calls

If your application is a dual-mode application that uses SSC API and ODBC function calls, you will need two different executables, one to be run locally and one to be run remotely.

Procedure

1. Create the application version that will run in local mode.
   a. Link the application to the LLA library (for example, `solidimpac.lib` for Windows).

      The LLA library provides support for both the ODBC functions and the SSC API functions.
   b. Modify the connection string to use local connection.
2. Create the application version that will run in remote mode.
   a. Link the application to both the solidDB ODBC driver and to the SSC API stub library (for example, `solidctrlstub.lib` for Windows).

      The 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".
   b. Modify the connection string to use remote connection.

Establishing remote connections

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

ODBC API

With the ODBC API, to establish a remote connection, the application calls the SQLConnect function with the name of the remote server.

Example

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);
```

SA API

With the SA API, to establish a remote connection, the application calls the SaConnect function with the name of the remote server.

Example

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");
```

JDBC

With JDBC, to establish a remote connection, the name of the remote server is defined in the connect string.

```
jdbc:solid://<hostname>:<port>
```

Appendix A. Shared memory access parameters

Server-side parameters

Table 18. Shared memory access parameters

[SharedMemoryAccess]	Description	Factory value	Startup
MaxSharedMemorySize	This parameter sets the maximum total size of the shared memory area used by solidDB. If the SMA server tries to allocate more, an "out of memory" error occurs. With value "0", the maximum value is set automatically to be the size of the physical memory of the computer (platform specific). **Note:** The value set with the **SharedMemoryAccess.MaxSharedMemorySize** parameter takes precedence over the value set with any corresponding kernel parameter (for example, SHMALL in Linux environments). Because of this, the value set with the **SharedMemoryAccess.MaxSharedMemorySize** parameter should never be higher than the value set with the corresponding kernel parameter.	0 (automatic) Unit: 1 byte, G=GB, M=MB, K=KB	RW/Startup
SharedMemoryAccessRights	This parameter sets a validation context for the user access to the shared memory area. It is modeled after a traditional file validation mask having components "user" (only the same user as the one that started the SMA server), "group" (any user belonging to the same group) and "all" (any user).	group	RW/Startup

Client-side parameters

Table 19. Shared memory access parameters (client-side)

[SharedMemoryAccess]	Description	Factory value	Startup
SignalHandler	The **SignalHandler** parameter controls the SMA signal handler functionality. When set to 'yes', the SMA driver signal handler handles the signals defined with the **Signals** parameter. The SMA driver signal handler enables the SMA system to survive the most common application failures, such as killing or interrupting the applications from outside, or when one of the application threads runs within the server code, and another thread running application code causes application to crash. Upon the capture of certain signals, the signal handler closes the SMA connections safely and exits the SMA application. This means that in most cases, the SMA server continues to run despite abnormal application exits. The SMA driver signal handler installs itself when the first SMA connection is established and uninstalls itself when the last SMA connection is closed. Previously installed signal handlers are retained.	yes	NA

Table 19. Shared memory access parameters (client-side) (continued)

[SharedMemoryAccess]	Description	Factory value	Startup
Signals	The **Signals** parameter defines the signals that can break the SMA connection and should be handled by the SMA driver. The signals are defined as integers or with the following mnemonics: SIGSTOP, SIGKILL, SIGINT, SIGTERM, SIGQUIT, SIGABORT.	Linux and UNIX: SIGINT, SIGTERM Windows: SIGINT	NA

Appendix B. Linked library access parameters

Linked library access (LLA) parameters appear in the `[Accelerator]` section of the `solid.ini` configuration file.

Table 20. Accelerator parameters

[Accelerator]	Description	Factory Value
ImplicitStart	If set to yes, solidDB starts 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 SSC API function SSCStartServer.	yes
ReturnListenErrors	If this parameter is set to yes and network listening fails, the SSCStartServer function returns an error. If this parameter is set to no and network listening fails, the SSCStartServer starts the LLA server but network connections are not possible.	no

Appendix C. Configuration parameters for a diskless server

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

Parameters used in diskless servers

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

IndexFile section

In the `IndexFile` section, the **FileSpec** and **CacheSize** parameters have specific settings for a diskless server.

Filespec_[1...n]

The **FileSpec** parameter describes the name and the maximum size of the database file. To define the maximum size in bytes for the main memory engine, 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 `solid.db`, 2147483647 bytes (2GB -1). For example:

```
FileSpec_1=solid.db 2147483647
```

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

The maximum size is limited by the physical memory available; this is because a diskless machine has no disk to use as swap space for virtual memory.

Note: 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 advanced replication 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; 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

If you are using the diskless server as a advanced replication replica server, the **Listen** parameter in the `Com` section affects the communication between the master and the diskless replica server.

Listen

The **Listen** parameter defines the protocol and the name that the diskless server uses when it starts listening to the network.

For example:

```
[Com]
Listen=tcpip 2315
```

The default value of the **Listen** parameter is `tcp 1964`.

For more information about the network names and protocols, see section *Managing network connections* in *IBM solidDB Administrator Guide*.

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 21. 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. **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 rollforward recovery.

Appendix D. solidDB Server Control API (SSC API)

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

Note: Some information on the functions apply also to SSC API for Java. For more information about SSC API for Java, see section Appendix E, "SolidServerControl class interface," on page 81.

Using SSC API

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.

Notifying functions of a special event

The linked library access 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).

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

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 linked library access server library (for example, `ssolidacxx.dll` for Windows platforms) or a "dummy" server library (for example, `solidctrlstub.lib` for Windows platforms) that are used to test remote applications that are using Control API.

SSC API and equivalent ADMIN COMMANDs

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

Refer to Appendix B, "Linked library access parameters," on page 53 for details on Control API equivalent ADMIN Commands.

Summary of SSC API functions

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

Table 22. Summary of control API functions

Function	Description	Supported in	For more details, see
SSCStartSmaServer	Starts a SMA server.	SMA	See "SSCStartSMAServer" on page 77.
SSCStartDisklessSmaServer	Starts a diskless SMA server.	SMA	See "SSCStartDisklessSMAServer" on page 75.
SSCStartServer	Starts a LLA server.	LLA and SSC API stub library	See "SSCStartServer" on page 72.
SSCStartDisklessServer	Starts a diskless LLA server.	LLA and SSC API stub library	See "SSCStartDisklessServer" on page 70.

Table 22. Summary of control API functions (continued)

Function	Description	Supported in	For more details, see
SSCSetState	Sets the state of a solidDB server (for example, SSC_STATE_OPEN indicates that subsequent connections are allowed). Setting the state to ~SSC_STATE_OPEN will block LLA connections and remote network connections. The following flag symbols can be used: • SSC_STATE_OPEN – the open flag is set to 1: new connections are allowed. • SSC_STATE_CLOSED – the open flag is set to 0: all new network and LLA connections are rejected, with the exception of connections from solidDB Remote Control (solcon) program. • SSC_DISABLE_NETCOPY – the netcopy disabling flag is set to 1: in HotStandby configuration, no netcopy can be received by the server for which this is set. The flag does not prevent the server to act as a source of netcopy. If only SSC_DISABLE_NETCOPY flag is set, the server is in the closed state. To enable netcopy, use the SSC API function SSCSetState() with the runflag value SSC_STATE_OPEN or SSC_STATE_CLOSED.	LLA and SSC API stub library	See "SSCSetState" on page 69.
SSCRegisterThread - deprecated as of 6.5 FP1	Registers an linked library access application thread for the server. Registration is required in every thread in the user application before any LLA API function can be called.	LLA and SSC API stub library	See "SSCRegisterThread" on page 66.
SSCUnregisterThread - deprecated as of 6.5 FP1	Unregisters a LLA application thread for the server. Registration removal is required in every thread that is registered before terminating.	LLA and SSC API stub library	See "SSCUnregisterThread" on page 79.
SSCStopServer	Stops SMA or LLA server.	SMA, LLA and SSC API stub library	See "SSCStopServer" on page 78.
SSCSetNotifier	Specifies a user-defined function which solidDB calls at a specified event, such as merge, backup, shutdown, etc.	LLA and SSC API stub library	See "SSCSetNotifier" on page 67.

Table 22. Summary of control API functions (continued)

Function	Description	Supported in	For more details, see
SSCIsRunning	Returns non-zero if the server is running.	LLA and SSC API stub library	See "SSCIsRunning" on page 65.
SSCIsThisLocalServer	Indicates whether the application is linked to the solidDB server with the LLA or the "dummy" (`solidctrlstub`) library to test solidDB remote applications using the SSC API.	LLA and SSC API stub library	See "SSCIsThisLocalServer" on page 66.
SSCGetServerHandle	Returns the solidDB server handle if the server is running.	LLA and SSC API stub library	See "SSCGetServerHandle" on page 64.
SSCGetStatusNum	Gets solidDB status information.	LLA and SSC API stub library	See "SSCGetStatusNum" on page 65.

SSC API reference

The SSC API reference describes each SSC API function in alphabetic order. Each description includes the purpose, synopsis, parameters, return value, and comments.

- "Function synopsis"
- "Parameters"
- "Return values" on page 63
- "SSC API error codes and messages" on page 64

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.

Parameters

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 SSC 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 23. SSC 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.
	
	Attention:
	
	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:
	`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);`

Return values

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".

SSC API error codes and messages

SSC API functions may return the error codes and messages listed in the table below.

These constants are defined in the `sscapi.h` file.

Table 24. Error codes and messages for SSC 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).

SSCGetServerHandle

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

Synopsis

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

Comments

This function has no corresponding solidDB SQL extension ADMIN COMMAND.

Return value

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

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 25. 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 solidDB 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 (HotStandby only)
- SSC_SERVER_NOTRUNNING - The server is not running.

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 26. 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 solidDB SQL extension ADMIN COMMAND.

SSCIsThisLocalServer

SSCIsThisLocalServer indicates whether the application is linked to a solidDB server or the "dummy" (`solidctrlstub`) library. The `solidctrlstub` library allows developers to test solidDB remote applications using Control API without linking the linked library access 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 solidDB SQL extension ADMIN COMMAND.

SSCRegisterThread

Note: SSCRegisterThread is deprecated as of 6.5 FP1; it is no longer necessary to register and unregister threads explicitly when using solidDB with linked library access (LLA). As of 6.5 FP1, thread registration is handled implicitly.

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 linked library access 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 27. 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 solidDB SQL extension ADMIN COMMAND.

See also

"SSCUnregisterThread" on page 79

SSCSetNotifier

SSCSetNotifier sets the callback functions that an linked library access 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 28. SSCSetNotifier parameters

Parameters	Usage Type	Description
h	in	Handle to the server.

Table 28. SSCSetNotifier parameters (continued)

Parameters	Usage Type	Description
what	in	Specifies the 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_CONT. • 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 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 HotStandby component 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 *IBM solidDB High Availability User Guide*. • SSC_NOTIFY_NETCOPY_FINISHED This callback function applies to the HotStandby component 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.
userdata	in, hold	User data to be passed to the notify function. Read the warning on releasing a parameter of usage type *hold* under "Parameters" on page 62.

Return value

- SSC_SUCCESS - Request from the server accepted.

  HotStandby 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 *IBM 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.

  HotStandby 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 (HotStandby only).

  SSC_SERVER_NOTRUNNING - The server is not running.

Comments

This function has no corresponding solidDB SQL extension ADMIN COMMAND.

Releasing a parameter of usage type *hold* should be done with caution. Read the warning for hold "Parameters" on page 62.

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

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 SSC_CALL user_own_shutdownrequest(SscServerT h, SscNotFunT what, void
    *userdata);
{
  if (shutdown not needed) {
    return SSC_ABORT;
  }
  return SSC_CONT; /*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_CONT, the shut down can proceed.

SSCSetState

SSCSetState sets the state of a LLA or SMA server. This allows you to control whether the server accepts subsequent connections.

If the server is set to "open", 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.

Synopsis

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

The SSCSetState function accepts the following parameters:

Table 29. SSCSetState parameters

Parameter	Usage Type	Description
h	in, use	Handle to the server.

runflags	in	The value for this parameter is a combination of two flags: the open flag and the netcopy disabling flag. The following flag symbols can be used:
		• SSC_STATE_OPEN – the open flag is set to 1: new connections are allowed.
		• SSC_STATE_CLOSED – the open flag is set to 0: all new network, LLA, and SMA connections are rejected, with the exception of connections from solidDB Remote Control (solcon) program.
		• SSC_DISABLE_NETCOPY – the netcopy disabling flag is set to 1: in HotStandby configuration, no netcopy can be received by the server for which this is set.
		The flag does not prevent the server to act as a source of netcopy. If only SSC_DISABLE_NETCOPY flag is set, the server is in the closed state. To enable netcopy, use the SSC API function SSCSetState() with the runflag value SSC_STATE_OPEN or SSC_STATE_CLOSED.
		`runflags = SSC_STATE_OPEN | SSC_STATE_CLOSED | SSC_DISABLE_NETCOPY`
		
		Tip: The flags can be used in combinations, for example:
		`...`
		`rc = SSCStartServer(g_argc, g_argv, &hh, SSC_STATE_OPEN|SSC_DISABLE_NETCOPY);`
		`...`
		
		If the server is started as closed, it can be opened with ADMIN COMMAND 'open', or with the solcon command `open`. The same effect can be achieved with the SSC API function SSCSetState().

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 solidDB SQL extension ADMIN COMMAND. The command is:

ADMIN COMMAND 'close';

SSCStartDisklessServer

SSCStartDisklessServer starts a diskless server using the linked library access.

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 30. 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 available arguments, see section *solidDB command line options* in the *IBM solidDB Administrator Guide*.
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 value for this parameter is a combination of two flags: the open flag and the netcopy disabling flag. The following flag symbols can be used:
		• SSC_STATE_OPEN – the open flag is set to 1: new connections are allowed.
		• SSC_STATE_CLOSED – the open flag is set to 0: all new network and LLA connections are rejected, with the exception of connections from solidDB Remote Control (solcon) program.
		• SSC_DISABLE_NETCOPY – the netcopy disabling flag is set to 1: in HotStandby configuration, no netcopy can be received by the server for which this is set.
		The flag does not prevent the server to act as a source of netcopy. If only SSC_DISABLE_NETCOPY flag is set, the server is in the closed state. To enable netcopy, use the SSC API function SSCSetState() with the runflag value SSC_STATE_OPEN or SSC_STATE_CLOSED.
		`runflags = SSC_STATE_OPEN  | SSC_STATE_CLOSED | SSC_DISABLE_NETCOPY`
		
		Tip: The flags can be used in combinations, for example:
		`...`
		`rc = SSCStartServer(g_argc, g_argv, &hh, SSC_STATE_OPEN|SSC_DISABLE_NETCOPY);`
		`...`
		
		If the server is started as closed, it can be opened with ADMIN COMMAND 'open', or with the solcon command `open`. The same effect can be achieved with the SSC API function SSCSetState().

lic_string	in	Specifies the string containing the solidDB license file.
ini_string	in	Specifies the string containing the solidDB configuration file.

Return value

- SSC_SUCCESS - The server is started.
- SSC_ERROR - The server failed to start.
- SSC_SERVER_INNETCOPYMODE - The server is netcopy mode (HotStandby 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 solidDB SQL extension ADMIN COMMAND.

Example

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 6, “Using the diskless capability,” on page 47.

SSCStartServer

SSCStartServer starts the linked library access. In multithread 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 subroutines 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 “SSCStartDisklessServer” on page 70.

Synopsis

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

The SSCStartServer function accepts the following parameters:

Table 31. 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 available arguments, see section *solidDB command line options* in the *IBM solidDB Administrator Guide*.
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 value for this parameter is a combination of two flags: the open flag and the netcopy disabling flag. The following flag symbols can be used:
		• SSC_STATE_OPEN – the open flag is set to 1: new connections are allowed.
		• SSC_STATE_CLOSED – the open flag is set to 0: all new network and LLA connections are rejected, with the exception of connections from solidDB Remote Control (solcon) program.
		• SSC_DISABLE_NETCOPY – the netcopy disabling flag is set to 1: in HotStandby configuration, no netcopy can be received by the server for which this is set.
		The flag does not prevent the server to act as a source of netcopy. If only SSC_DISABLE_NETCOPY flag is set, the server is in the closed state. To enable netcopy, use the SSC API function SSCSetState() with the runflag value SSC_STATE_OPEN or SSC_STATE_CLOSED.
		`runflags = SSC_STATE_OPEN | SSC_STATE_CLOSED | SSC_DISABLE_NETCOPY`
		
		Tip: The flags can be used in combinations, for example:
		`...`
		`rc = SSCStartServer(g_argc, g_argv, &hh, SSC_STATE_OPEN|SSC_DISABLE_NETCOPY);`
		`...`
		
		If the server is started as closed, it can be opened with ADMIN COMMAND 'open', or with the solcon command `open`. The same effect can be achieved with the SSC API function SSCSetState().

Example: Starting up SSCStartServer

Start up SSCStartServer with the server name, 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.

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 (HotStandby 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 solidDB 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 "Explicit startup with SSC API function SSCStartServer" on page 36.

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

SSCStartDisklessSMAServer

SSCStartDisklessSMAServer starts a diskless server using SMA.

Synopsis

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

The SSCStartDisklessSMAServer function accepts the following parameters:

Table 32. SSCStartDisklessSMAServer parameters

Parameter	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 available arguments, see section *solidDB command line options* in the *IBM solidDB Administrator Guide*.
h	out	Returns a handle to the started server. This handle is needed when referencing the server with other Control API functions.

Table 32. SSCStartDisklessSMAServer parameters (continued)

Parameter	Usage Type	Description
runflags	in	The value for this parameter is a combination of two flags: the open flag and the netcopy disabling flag. The following flag symbols can be used: • SSC_STATE_OPEN – the open flag is set to 1: new connections are allowed. • SSC_STATE_CLOSED – the open flag is set to 0: all new network and SMA connections are rejected, with the exception of connections from solidDB Remote Control (solcon) program. • SSC_DISABLE_NETCOPY – the netcopy disabling flag is set to 1: in HotStandby configuration, no netcopy can be received by the server for which this is set. The flag does not prevent the server to act as a source of netcopy. If only SSC_DISABLE_NETCOPY flag is set, the server is in the closed state. To enable netcopy, use the SSC API function SSCSetState() with the runflag value SSC_STATE_OPEN or SSC_STATE_CLOSED. `runflags = SSC_STATE_OPEN	SSC_STATE_CLOSED	SSC_DISABLE_NETCOPY` **Tip:** The flags can be used in combinations, for example: `...` `rc = SSCStartSMAServer(g_argc, g_argv, &hh, SSC_STATE_OPEN|SSC_DISABLE_NETCOPY);` `...` If the server is started as closed, it can be opened with ADMIN COMMAND 'open', or with the solcon command `open`. The same effect can be achieved with the SSC API function SSCSetState().
lic_string	in	Specifies the string containing the solidDB license file.
ini_string	in	Specifies the string containing the solidDB configuration file.

Return values

- SSC_SUCCESS - The server is started.
- SSC_ERROR - The server failed to start.
- SSC_SERVER_INNETCOPYMODE - The server is netcopy mode (HotStandby 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 solidDB SQL extension ADMIN COMMAND.

See also

"SSCStopServer" on page 78

SSCStartSMAServer

SSCStartSMAServer starts a server using SMA.

Synopsis

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

The SSCStartSMAServer function accepts the following parameters:

Table 33. SSCStartSMAServer 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 available arguments, see section *solidDB command line options* in the *IBM solidDB Administrator Guide*.
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 value for this parameter is a combination of two flags: the open flag and the netcopy disabling flag. The following flag symbols can be used:
		• SSC_STATE_OPEN – the open flag is set to 1: new connections are allowed.
		• SSC_STATE_CLOSED – the open flag is set to 0: all new network and SMA connections are rejected, with the exception of connections from solidDB Remote Control (solcon) program.
		• SSC_DISABLE_NETCOPY – the netcopy disabling flag is set to 1: in HotStandby configuration, no netcopy can be received by the server for which this is set.
		The flag does not prevent the server to act as a source of netcopy. If only SSC_DISABLE_NETCOPY flag is set, the server is in the closed state. To enable netcopy, use the SSC API function SSCSetState() with the runflag value SSC_STATE_OPEN or SSC_STATE_CLOSED.
		`runflags = SSC_STATE_OPEN  | SSC_STATE_CLOSED | SSC_DISABLE_NETCOPY`
		
		Tip: The flags can be used in combinations, for example:
		`...`
		`rc = SSCStartSMAServer(g_argc, g_argv, &hh, SSC_STATE_OPEN|SSC_DISABLE_NETCOPY);`
		`...`
		
		If the server is started as closed, it can be opened with ADMIN COMMAND 'open', or with the solcon command `open`. The same effect can be achieved with the SSC API function SSCSetState().

Return values

- 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 (HotStandby 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 solidDB SQL extension ADMIN COMMAND.

When you start a new solidDB server, you must explicitly specify that solidDB create a new database with the function SSCStartSMAServer() with the -U *username* -P *password* -C *catalogname* (the default database catalog name) parameters.

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

The SSCStartSMAServer 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, and so on.

See also

"SSCStopServer"

SSCStopServer

SSCStopServer stops an linked library access 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 restart the server by using SSCStartServer.

Synopsis

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

The SSCStopServer function accepts the following parameters:

Table 34. 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 B, “Linked library access parameters,” on page 53 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 solidDB SQL extension ADMIN COMMAND 'shutdown'.

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

See also

SSCStartServer

SSCSetState

SSCUnregisterThread

Note: SSCUnregisterThread is deprecated as of 6.5 FP1; it is no longer necessary to register and unregister threads explicitly when using solidDB with linked library access (LLA). As of 6.5 FP1, thread registration is handled implicitly.

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 35. 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 solidDB ADMIN COMMAND.

See also

"SSCRegisterThread" on page 66

Appendix E. SolidServerControl class interface

The complete public interface for the SolidServerControl class is described below.

Details on parameters and corresponding ADMIN COMMANDs are included in the SSC function descriptions in section "SSC API reference" on page 62.

For an example of a LLA program that uses some of the methods in this class, see the LLA for Java sample in the `samples/aclib_java` directory.

Return value 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;
/**
 * 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();
```

```
/**
* startDisklessServer starts the solidDB Linked Library Access server
* in a diskless mode
*
* @param argv       parameter vector for the accelerator server.
*
* @param runflags   Options for this parameter are SSC_STATE_OPEN,
*                   SSC_STATE_CLOSED, and SSC_DISABLE_NETCOPY.
*
* @param lic_file   The contents of the solidDB license file as a string,
*                   since the diskless version cannot read the
*                   information from the disk
*
* @param ini_file   The contents of the solid.ini configuration file as a string,
*                   since the diskless version cannot read the information
*                   from the disk
*
* @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 static long startDisklessServer( String[] argv, long runflags,
String lic_string, String ini_string )


/**
 * startServer starts the solidDB Linked Library Access server
 *
 * @param argv       parameter vector for the LLA server
 *                   Note! You must give the working directory containing
 *                   solidDB license file (f.ex. -c\tmp) first, in front
 *                   of other parameters.
 *
 * @param runflags   Options for this parameter are SSC_STATE_OPEN,
 *                   SSC_STATE_CLOSED, and SSC_DISABLE_NETCOPY.
 * *
 * @return           the return value from the server:
 *     SSC_SUCCESS
 *     SSC_ERROR
 *     SSC_INVALID_LICENSE - No license or invalid license file found.
 *     SSC_NODATABASEFILE  - No database file found.
 */
public long startServer( String[] argv, long runflags );


/**
* stopServer stops the LLA server
 *
 * @param runflags   Runflags for stopping LLA server.
 *                   See section SSCStopServer 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 solidDB Linked Library Access server
*  - deprecated as of V6.5 Fix Pack 1
*
*
* @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
* solidDB Linked Library Access server - deprecated as of V6.5 Fix Pack 1
*
*
* @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();
```

Index

A

administering
  diskless server configuration file options 55
advanced replication
  using with linked library access (LLA) 40

B

backup
  listening mode 68

C

C applications
  samples 39
CacheSize (parameter) 55
  configuring for diskless 55
Com section
  configuring for diskless 56
configuration file
  diskless 55
Control API
  SSCGetActiveTaskClass (function) 59
  SSCGetServerHandle (function) 59
  SSCGetStatusNum (function) 59
  SSCGetTaskClassState (function) 59
  SSCIsRunning (function) 59
  SSCIsThisLocalServer (function) 59
  SSCSetNotifier (function) 59

D

database
  Index file section 55
  size 36
diskless server mode 47
  parameters 55
dual-mode applications 11, 49

E

events
  notifying function of 59

F

FileSpec (parameter) 55
  configuring for diskless 55

I

ImplicitStart (parameter) 53
IndexFile section (parameters)
  diskless 55

J

JDBC API
  definition 10

L

library
  solidimpac 7
linked library access (LLA) 1
  components 6
  definition 6
  shutting down 39
  starting 35
  supported platforms 4, 6
Linux
  memory limitations 55
Listen (parameter)
  configuring for diskless 56
local application
  definition 11

M

MaxSharedMemorySize (parameter) 51
memory
  CacheSize (for diskless server) 55
  total used by diskless server 56

N

netcopy
  listening mode 68

O

ODBC
  overview 9

P

parameters
  FileSpec 55

R

remote applications 49
  definition 11
ReturnListenErrors (parameter) 53

S

SA API
  definition 9
SaConnect
  implicit startup with 38
server information
  retrieving 59

shared memory access (SMA) 1
components 4
configuring 13
definition 3
monitoring 26
troubleshooting 27
SharedMemoryAccessRights (parameter) 51
shutting down
linked library access 39
SignalHandler (parameter) 51
Signals (parameter) 51
SMA server
starting 24, 31
SMA system parameters
AIX 15
overview 14
solidctrlstub 9, 10, 62
solidDB client APIs and drivers 9
solidDB configuration file
FileSpec (parameter) 55
Listen (parameter) 56
parameter settings 55
solidDB drivers and client APIs 9
solidDB SA 9
solidDB Server Control (SSC) API for Java 10, 81
solidDB Server Control API (SSC API)
definition 10
solidctrlstub 10
solidimpac 7
SolidServerControl class 81
SQLConnect (function)
implicit startup with 37
SSC API (Control API) 60
ADMIN COMMAND equivalents 60
definition 10
summary of scheduling functions 60
SSC API for Java 10, 81
SSC_ABORT 64
SSC_CALL 62
SSC_CONNECTIONS_EXIST 64
SSC_CONT 64
SSC_ERROR 64
SSC_FINISHED 64
SSC_INFO_SERVER_RUNNING 64
SSC_INVALID_HANDLE 64
SSC_INVALID_LICENSE 64
SSC_NODATABASEFILE 64
SSC_SERVER_INNETCOPYMODE 64
SSC_SERVER_NOTRUNNING 64
SSC_STATE_OPEN 71, 76, 77
SSC_SUCCESS 64
SSC_TASK_ALL 62
SSC_TASK_BACKUP 62
SSC_TASK_CHECKPOINT 62
SSC_TASK_HOTSTANDBY 62
SSC_TASK_HOTSTANDBY_CATCHUP 62
SSC_TASK_LOCALUSERS 62
SSC_TASK_MERGE 62
SSC_TASK_NONE 62
SSC_TASK_REMOTEUSERS 62
SSC_TASK_SYNC_HISTCLEAN 62
SSC_TASK_SYNC_MESSAGE 62
SSC_UNFINISHED_TASKS 64
sscapi.h 62
SSCGetServerHandle
function description 64
SSCGetStatusNum
function description 65
SSCIsRunning
function description 65
SSCIsThisLocalServer
function description 66
SSCRegisterThread
function description 66
SSCServerT 36
SSCSetNotifier
function description 67
SSCSetState
function description 69
SSCStartDisklessServer
function description 70
SSCStartDisklessSMAServer 75, 77
SSCStartServer
explicit startup with 36
function description 72
SSCStopServer
function description 78
shut down with 39
SscTaskSetT 62
SSCUnregisterThread
function description 79
starting solidDB
with linked library access 35
status
retrieving 59

T

task information
retrieving 59

Notices

Copyright © Solid® Information Technology Ltd. 1993, 2010.

All rights reserved.

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

This product is protected by U.S. patents 6144941, 7136912, 6970876, 7139775, 6978396, 7266702, 7406489, 7502796, and 7587429.

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

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the

names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_. All rights reserved.

Trademarks

IBM, the IBM logo, ibm.com®, Solid, solidDB, InfoSphere, DB2®, Informix®, and WebSphere® are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at Copyright and trademark information (www.ibm.com/legal/copytrade.shtml).

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Printed in USA

SC23-9876-01