www.ibm.com/software/data/**soliddb**

Getting Started With solidDB For VxWorks^{tm}

Version 6.0 | October 2009

solidDB™

Getting Started With solidDB For VxWorks

Copyright © Solid Information Technology Ltd. 2008, 2009
Document number: GVxW-6.00
Product version: 06.00.1061
Date: 2009-09-30

Table of Contents

1 Welcome 1
1.1 About This Guide 1
1.1.1 Organization 1
1.1.2 Audience 2
1.2 Conventions 2
1.2.1 About solidDB 2
1.2.2 Typographic Conventions 2
1.2.3 Syntax Notation 3
1.3 solidDB Documentation 4
2 Introduction 7
2.1 Purpose 7
2.2 What You Should Already Know 7
2.3 Software and Hardware Requirements 7
3 Overview 9
3.1 The AcceleratorLib 9
3.2 Overview of Building and Executing a solidDB Application on VxWorks 11
4 Configuring, Starting, and Stopping solidDB 13
4.1 Before You Start: `solid.ini` File 13
4.1.1 FileSpec 13
4.1.2 CacheSize 13
4.1.3 Listen 14
4.1.4 MergeInterval and CheckPointInterval 14
4.1.5 LogEnabled 14
4.2 Starting solidDB 15
4.3 Stopping solidDB 16
5 Incorporating the AcceleratorLib into a VxWorks Project 17
5.1 Development Environment 17
5.1.1 Directory Structure 17
5.1.2 VxWorks Options 18
5.1.3 Link solidac.a 19
5.1.4 Header Files 19
5.2 Source Code Modifications 19
5.3 Working Directory 20
6 Design Considerations and Performance Tuning Tips 23
6.1 Overview 23
6.2 Starting the Server 24
6.3 Tuning a Multi-threaded solidDB Server 26
6.4 SSCAdvanceTasks 27
6.4.1 SSCAdvanceTasks, and checking Server state 27

6.5 Adjusting Stack Size 28
6.6 Assigning Memory Partitions for solidDB 28
6.7 Reclaiming System Resources and Uninitializing the Server 29
6.8 Selecting Memory Allocator 29
6.9 Redirecting Server Messages 30
7 ODBC Client 31
7.1 Connection 32
7.2 Statement Execution 33
7.3 Disconnecting 34
8 solidDB VxWorks Development Differences (from Other Platforms) 35
A A Quick Overview of Developing Applications 37
B Different Approaches to "Building Your Image" 39
B.1 Abstract 39
B.2 Background 39
B.3 Possible Combinations of Images 40
C SSC API Functions 43
C.1 SSCLogHookAdd 43
C.1.1 Synopsis 43
C.1.2 Return Value 44
C.2 SSCSetMemoryAllocatorType 44
C.2.1 Synopsis 44
C.3 SSCSetStacksize 44
C.3.1 Synopsis 44
C.3.2 Return Value 44
C.4 Conversion Functions 45
C.4.1 Synopsis 45
D SsSys Utility Functions 47
D.1 SsSysMemGlobalInit 47
D.1.1 Synopsis 47
D.1.2 Return Value 47
D.2 SsSysMemGlobalDone 47
D.2.1 Synopsis 47
D.3 SsSysResGlobalDone 47
D.3.1 Synopsis 48
Glossary 49
Index 53

List of Figures

7.1 Local Clients and ODBC Driver 31
B.1 One image: VxWorks + solidac.a + your application 40
B.2 Two separate images: VxWorks, solidac.a + your application 41

List of Tables

1.1 Typographic Conventions 2
1.2 Syntax Notation Conventions 3
C.1 SSCLogHookAdd Parameters 43

List of Examples

6.1 Using SSCAdvanceTasks() to Monitor the Server 27

Chapter 1. Welcome

IBM solidDB provides the advanced data management software that makes networks flow.

IBM solidDB (solidDB) ensures data consistency and integrity network wide with its Intelligent Transaction technology and proven bi-directional, multi-point synchronization. When used with options, such as High Availability (formerly called HotStandby), AcceleratorLib, and Diskless Edition, solidDB provides an extensible platform that achieves an even higher level of fault tolerance, performance, and embeddability.

1.1 About This Guide

This guide is intended for developers who are using solidDB for Wind River Workbench/VxWorks. It provides the proper procedure for installing and starting solidDB within the Workbench/VxWorks environment. Note that this guide supplements the information contained in *solidDB SmartFlow Data Replication Guide*, and *solidDB SQL Guide*.

1.1.1 Organization

This guide contains the following sections:

- Chapter 2, *Introduction* lists hardware and software requirements for this product.
- Chapter 3, *Overview* describes the solidDB AcceleratorLib library and gives an overview of how to use that library to build a VxWorks process that contains the solidDB server.
- Chapter 4, *Configuring, Starting, and Stopping solidDB* explains how to configure your system, for example, how to specify which disk drives and files data should be stored in.
- Chapter 5, *Incorporating the AcceleratorLib into a VxWorks Project* explains in more detail how to incorporate the solidDB AcceleratorLib into an existing VxWorks-based project.
- Chapter 6, *Design Considerations and Performance Tuning Tips* suggests ways to maximize performance.
- Chapter 7, *ODBC Client* details the requirements in order to connect an ODBC client to the solidDB server.
- Chapter 8, *solidDB VxWorks Development Differences (from Other Platforms)* explains some of the differences between developing for VxWorks and developing for other platforms. If you have worked with solidDB on other platforms and are working with solidDB on VxWorks for the first time, you may find this chapter helpful.

- Appendix A, *A Quick Overview of Developing Applications* lists key points to remember when developing applications.
- Appendix B, *Different Approaches to "Building Your Image"* explains some details of building a VxWorks executable "image" that includes solidDB.
- Appendix C, *SSC API Functions* explains the SSC API functions related to solidDB for Workbench/VxWorks.

Glossary

Glossary provides definitions of terms.

1.1.2 Audience

This guide assumes a working knowledge of the C programming language, general DBMS knowledge, and familiarity with SQL and solidDB. It also assumes knowledge of Wind River Workbench and VxWorks products.

1.2 Conventions

1.2.1 About solidDB

solidDB provides advanced database solutions for mission-critical applications.

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

1.2.2 Typographic Conventions

This manual uses the following typographic conventions:

Table 1.1. Typographic Conventions

Format	Used for
Database table	This font is used for all ordinary text.
NOT NULL	Uppercase letters on this font indicate SQL keywords and macro names.

Format	Used for
`solid.ini`	These fonts indicate file names and path expressions.
`SET SYNC MASTER YES;` `COMMIT WORK;`	This font is used for program code and program output. Example SQL statements also use this font.
run.sh	This font is used for sample command lines.
`TRIG_COUNT()`	This font is used for function names.
`java.sql.Connection`	This font is used for interface names.
`LockHashSize`	This font is used for parameter names, function arguments, and Windows registry entries.
`argument`	Words emphasised like this indicate information that the user or the application must provide.
solidDB Administration Guide	This style is used for references to other documents, or chapters in the same document. New terms and emphasised issues are also written like this.
File path presentation	File paths are presented in the Unix format. The slash (/) character represents the installation root directory.
Operating systems	If documentation contains differences between operating systems, the Unix format is mentioned first. The Microsoft Windows format is mentioned in parentheses after the Unix format. Other operating systems are separately mentioned.

1.2.3 Syntax Notation

This manual uses the following syntax notation conventions:

Table 1.2. Syntax Notation Conventions

Format	Used for
`INSERT INTO` *`table_name`*	Syntax descriptions are on this font. Replaceable sections are on *`this`* font.
`solid.ini`	This font indicates file names and path expressions.

Format	Used for
[ ]	Square brackets indicate optional items; if in bold text, brackets must be included in the syntax.
|	A vertical bar separates two mutually exclusive choices in a syntax line.
{ }	Curly brackets delimit a set of mutually exclusive choices in a syntax line; if in bold text, braces must be included in the syntax.
...	An ellipsis indicates that arguments can be repeated several times.
. . .	A column of three dots indicates continuation of previous lines of code.

1.3 solidDB Documentation

Below is a complete list of documents available for solidDB. solidDB documentation is distributed in PDF format.

Electronic Documentation

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

- *solidDB Programmer Guide*. This guide explains in detail how to use features such as solidDB Stored Procedure Language, triggers, events, and sequences. It also describes the interfaces (APIs and drivers) available for accessing solidDB and how to use them with a solidDB database.
- *solidDB In-Memory Database User Guide*. This manual describes how to use the in-memory database of solidDB In-memory Engine.
- *solidDB SmartFlow Data Replication Guide*. This guide describes how to use the solidDB SmartFlow technology to synchronize data across multiple database servers.
- *solidDB AcceleratorLib User Guide*. Linking the client application directly to the server improves performance by eliminating network communication overhead. This guide describes how to use the AcceleratorLib library, a database engine library that can be linked directly to the client application.

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

  This manual also explains how to set up a solidDB to run without a disk drive.
- *solidDB High Availability User Guide*. solidDB CarrierGrade Option (formerly called the HotStandby Option) allows your system to maintain an identical copy of the database in a backup server or "secondary server". This secondary database server can continue working if the primary database server fails.
- *Getting Started With solidDB For VxWorks*. This guide describes how to take into use solidDB on the VxWorks environment. It also provides guidelines for application development and performance tuning. This manual is included only in packages for VxWorks.

Chapter 2. Introduction

2.1 Purpose

The purpose of this document is to describe the installation and use of solidDB under VxWorks, which is Wind River's Real-Time Operating System (RTOS) for embedded systems. This document will also highlight some of the differences between usage on RTOSs and non-RTOSs, and between usage on embedded systems and non-embedded systems.

2.2 What You Should Already Know

This solidDB documentation for VxWorks assumes that you have a basic familiarity with:

1. Wind River Workbench and VxWorks products.
2. The Workbench user interface. In particular, we assume that you know how to create a VxWorks project and modify its build properties as well as its kernel options. We also assume that you know how to create a downloadable object on the host (development machine) and then download it to the target (execution machine).
3. Concepts such as processes, tasks, and threads, especially as these apply to VxWorks. The output file executed on the VxWorks target is made of one process incorporating several tasks (that is, threads).
4. The C programming language.

Most users will also have to be familiar with:

1. SQL
2. ODBC

Familiarity with solidDB AcceleratorLib option is very helpful. However, this document provides some information about the AcceleratorLib option in case you are not already familiar with it. Complete documentation for the AcceleratorLib is in *solidDB AcceleratorLib User Guide*.

2.3 Software and Hardware Requirements

To develop an embedded application, you typically need two computers, the "target" and the "host". The "target" machine is the machine where solidDB will run on (for example, the VxWorks machine). The "host" machine is the machine on which you will do development work. For VxWorks development, the host system should have

- Wind River Workbench development environment

The target system should have

- Wind River VxWorks 6.x

Note that VxWorks is available for more than one type of microprocessor. When you order the solidDB Development Kit (SDK) for VxWorks, make sure that you specify the correct processor family, for example, x86, Power PC, StrongARM, etc.

The following must be available in the target machine:

- A Real-Time Clock
- A file system (unless you are using the diskless version of the AcceleratorLib)
- POSIX-compliant timers
- ftruncate

Chapter 3. Overview

This chapter provides the following:

1. A description of the solidDB AcceleratorLib, which is a library that contains the solidDB subroutines.

2. An overview of how to build and execute a solidDB application on VxWorks.

3.1 The AcceleratorLib

The solidDB AcceleratorLib library is a complete version of the solidDB database server in the form of a callable subroutine library. This library can be linked to your client application code so that your client and the solidDB database server are in the same executable. When your client is linked to the library, it may call database server functions directly rather than by going through a communications protocol, such as TCP/IP. This improves performance by eliminating network overhead. The server subroutines can also be called remotely through an ODBC interface.

Note

A remote client refers to a client calling through the TCP/IP interface. Such a client is usually, but not necessarily, on a separate node (VxWorks target board) from the server.

On most platforms, solidDB is provided in two forms: as an executable, or as the AcceleratorLib (which is an option). Although on non-embedded platforms, this library file is called the "Accelerator option", on embedded platforms, this library file is the ONLY form in which the server is provided, and therefore it is not truly an "option". Thus, in this VxWorks manual we will usually refer to it as the " Accelerator library" rather than the "Accelerator option".

The AcceleratorLib library file is named `solidac.a`. Like any software designed to run on VxWorks. it must be linked to VxWorks to run.

The AcceleratorLib library contains not only a complete set of solidDB subroutines, but also the `SSCStartServer()` and `SSCStartDisklessServer()` functions, which you can use to start the server running as a separate "thread" or task within the overall process.

solidDB AcceleratorLib contains three separate APIs. Two of the APIs are proprietary to solidDB: the SSC API and the SA API. The third API is the ODBC API, which is a standard API for exchanging data with database servers.

The SSC API (solidDB Control API) allows, for example, setting the priority for various tasks that run within the server. The solidDB SA API is solidDB's proprietary low-level interface to AcceleratorLib subroutines. Detailed documentation of the SSC and SA APIs is in *solidDB AcceleratorLib User Guide*.

You do not need to use the SSC and SA APIs. The database server functionality in the AcceleratorLib may also be accessed by using ODBC calls. The ODBC API is available both to the client application linked directly to the AcceleratorLib (that is, the "local" client) and to clients that are not linked to the AcceleratorLib and which communicate with the library via TCP/IP protocol (that is, "remote" clients). Remote clients also have the option to access the server by using JDBC calls.

Note that this document focuses almost exclusively on using the solidDB AcceleratorLib option with VxWorks. This document does not cover other options such as the "Diskless" and "High Availability" ("Hot Standby") options, although these are available on VxWorks.

If you are familiar with developing with solidDB on other platforms, you will find significant differences when you use RTOS platforms and embedded systems. For details, see Chapter 8, *solidDB VxWorks Development Differences (from Other Platforms)*.

Under VxWorks, there is only one process (which includes the OS, and applications such as the solidDB server and your client applications), so you must follow certain rules to be able to compile, link, and load your programs. If you do not follow these rules, you may find that some required code is not loaded, or symbols are not resolved. The simplest way to make sure that everything is linked and loaded properly is to create a single "image" that contains:

1. The VxWorks operating system code.
2. The `solidac.a` (server library) code.
3. Your application code.

There are other possible approaches. Those are discussed in Appendix B, *Different Approaches to "Building Your Image"*.

Typically your code will be linked to the operating system, and the operating system must call the equivalent of a main() function in each application that you want to run. Therefore, the process of developing a solidDB based program (or any other program) requires that you follow specific rules for the embedded system. Two of the key rules are:

1. Your application will not have a function named `main()`. The OS will start your application by calling an "entry routine" in your application, but this entry routine is not called main().

2. Your application and the OS are linked into a single executable, typically called an "image". The compilation and linking are done on the host, and then the image is downloaded to the target machine and executed on the target.

3.2 Overview of Building and Executing a solidDB Application on VxWorks

The basic process is described below. This process is consistent with the way that you build and execute any application on VxWorks.

1. Make sure that the following requirements are met:

    a. The required storage device(s), such as disk drives, must be initialized. (Storage devices are not required if you are using the "diskless" version of AcceleratorLib.)

    b. The files that solidDB requires at startup time must be present. These files are: the `solid.ini` configuration file and the `solid.lic` license file. These files must be in the directory that the solidDB server starts in. You specify the solidDB server's start directory by passing a parameter to the `SSCStartServer()` function (described later in this manual and also in the AcceleratorLib manual).

2. Create a VxWorks project.

3. Write the application program. This must be written so that it can be linked to the AcceleratorLib library (`solidac.a`). For more details about writing an application that works with the AcceleratorLib library, see *solidDB AcceleratorLib User Guide*. Make sure that you write the application program so that it has an entry point instead of a `main()` function. Either the operating system will call that entry point, or else the development shell (typically the windsh) will directly call (or spawn a task with) that entry point.

4. Your application program should have code to do the following:

    a. Call the `SSCStartServer()` or `SQLConnect()` function to start solidDB before any client application tries to connect to the server.

    b. Run user applications as spawned VxWorks tasks (these are usually spawned by your application program, but it is possible to start them independently of your application).

    c. Call the `SSCStopServer()` function to shut down solidDB when all work by the server and application has been completed.

5. Build your application program. Depending on your needs, you can choose to:

   a. Create an application/AcceleratorLib image that is separate from the VxWorks image; or

   b. Link VxWorks, application, and AcceleratorLib as one image. Typically, the compilation and linking will be done with a Workbench project.

   See Appendix B, *Different Approaches to "Building Your Image"* for more information about different ways to link your application and the AcceleratorLib library.

6. Begin executing your application/AcceleratorLib.

   a. If you choose to link everything as one image (OS, application, and AcceleratorLib), then `usrAppInit.c` in your Workbench project should do a taskSpawn with your application code entry as the task entry point. In this case, a reboot on the target will start everything (VxWorks, server, and your application).

   b. If you create separate VxWorks and application/AcceleratorLib images, then rebooting the target will start VxWorks. After rebooting, download the application image. Then start your application from the windsh by calling your code's entry point. This will start the AcceleratorLib server and your application.

After you complete these steps, the target machine should be running the operating system and the Accelerated application (solidDB along with the application code that you wrote).

Note that the accelerated application can accept both "local" and "remote" connections. A "local" connection is a "direct" connection from the application code that is compiled and linked to the AcceleratorLib subroutine library. A remote connection is a connection made through the TCP/IP protocol.

Chapter 4. Configuring, Starting, and Stopping solidDB

After you have installed solidDB, you must configure it before you can start running it. To configure it, you must set values in the `solid.ini` configuration file, and set VxWorks-specific options within your Workbench development environment.

4.1 Before You Start: `solid.ini` File

This section discusses some of the initialisation parameters that require special consideration in an embedded environment such as VxWorks. None of these parameters are specific to VxWorks or even to all embedded environments; however, it is important to set them to appropriate values.

4.1.1 FileSpec

The *`FileSpec`* parameter(s) specify the location and size of the files that hold the database.

```
[IndexFile]
FileSpec_1 = //path_to_working_directory/solid1.db 8388608
FileSpec_2 = //path_to_working_directory/solid2.db 8388608
```

The size of the database is of prime importance in an embedded system. In the above example, the database is split into two files of equal size (8M) located in the same directory under the names solid1.db and solid2.db. These files can be located in different directories and can have different sizes. When all the files are filled, solidDB stops inserting new entries in the database.

Important

It is highly recommended that the FileSpec path is an absolute path. On VxWorks, it is not possible to refer to different devices by using relative paths. It is extremely important not to use chdir() together with relative paths after starting up the server. This can have unexpected consequences.

4.1.2 CacheSize

The *`CacheSize`* parameter defines the amount of memory available to "cache" parts of the database in RAM.

```
CacheSize = 4194304
```

In the above example, 25% of the database size (4 Megabytes of RAM for 16MB of database files) will be stored in main memory. Caching the most-frequently used parts of the database increases the overall efficiency of the solidDB server. We recommend that the size of the cache be at least 10% of the total size of all the database files. (You can use a smaller amount, but performance is likely to be reduced.) If enough memory is available, then you can allocate up to 30% or more of the estimated size of the database. We do not recommend running solidDB with a cache that is larger than 40% of its database size because then the cache management operations may offset the speed gains of faster memory access.

4.1.3 Listen

Under VxWorks, solidDB supports remote connections through the TCP communication protocol only. The "listen" string must contain the IP address and the port number. Note that the "local" client function calls to the server bypass the communications protocol, so the TCP information is only needed for remote connections.

```
[Com]
Listen = tcp 1.2.3.4 1320
```

4.1.4 MergeInterval and CheckPointInterval

`MergeInterval` defines the number of inserts before solidDB triggers a merge operation. *`CheckPointInterval`* defines the number of inserts before solidDB executes a checkpoint operation.

```
[General]
MergeInterval = 5000
CheckPointInterval = 10000
```

Smaller, more frequent, checkpoints may be preferable in real-time operating systems. However, because the server is multi-threaded and because both the operating system and solidDB's internal "tasking" system allow you to specify priorities, merge and checkpoint operations generally do not prevent the system from responding quickly to high-priority tasks.

4.1.5 LogEnabled

The "Logging" section of the configuration file specifies whether the server uses logging to enable it to recover more data after a failure. Turning on logging reduces performance and uses up more disk space, but increases safety.

```
[Logging]
LogEnabled = Yes
```

We recommend that you turn on logging to ensure no data is lost even during a crash.

Details about each of these parameters are available in the *solidDB Administration Guide*.

4.2 Starting solidDB

To start solidDB, you must call the `SSCStartServer()` function (or the `SQLConnect()` function). *solidDB AcceleratorLib User Guide* details the `SSCStartServer` function and its parameters. When solidDB starts, it operates in the working directory specified in the parameters passed to `SSCStartServer` (argv). If a solidDB database already exists in the specified working directory, then `SSCStartServer` opens it. If the specified working directory does not contain a solidDB database, then `SSCStartServer` creates an "empty" database and opens it.

The code sample below shows how to start solidDB on VxWorks. Note that this code contains code that applies to all platforms, not just VxWorks.

```
/* Solid server handle. Most server-control functions */
/* require a handle to the server. */
SscServerT h;

/* Prepare arguments to start the server. */
argc = 5;
char *argv[5];
argv[0] = "";           /* empty string */
argv[1] = "-xpathprefix:/workingdirectory/";
argv[2] = "-Udba";
argv[3] = "-Pdba";
argv[4] = "-Cdba";

SscRetT rc = 0;         /* Return Code from API calls */

/* Standard Solid. Start the server and serve */
/* the clients until shutdown. */
/* Note: SSC_STATE_OPEN means "accept remote connections". */
rc = SSCStartServer(argc, argv, 'h, SSC_STATE_OPEN);
if (rc != SSC_ERROR)
{
```

```
/* If rc is not SSC_ERROR, then Solid server is */
/* up and running. See solidDB AcceleratorLib */
/* User Guide for a complete definition of */
/* the parameters and return code. */
}
src = SSCStopServer(h, TRUE);
assert(src == SSC_SUCCESS);
```

Important

The argument for -xpathprefix must be absolute. If it is not, using chdir() after the server startup can have unexpected consequences.

Note

-xpathprefix:/xyz is the recommended way to specify the working directory under VxWorks. `/xxx` is the working directory for solidDB. The application needs to mount the working directory and ensure it is accessible (read-write) before calling SSCStartServer.

Note

The **-Cdba** parameter specifies that when the database is created for the first time, the server creates a catalogue named "dba". The parameters **-Udba** and **-Pdba** specify that the username and password are "dba" and "dba", respectively. This user will have system administration rights (all database rights including the right to grant equal or lesser rights to other users).

4.3 Stopping solidDB

To stop solidDB, you must call the `SSCStopServer()` function, passing the solidDB server's handle:

```
SSCStopServer(h, TRUE);
```

The stopping of solidDB is not specific to VxWorks. Ensure that all clients are disconnected before shutting down solidDB. *solidDB AcceleratorLib User Guide* describes the `SSCStopServer` function, including its input parameters and the return code.

Chapter 5. Incorporating the AcceleratorLib into a VxWorks Project

This chapter explains in more detail how to incorporate the solidDB AcceleratorLib library into an existing VxWorks-based project.

The AcceleratorLib option is delivered in the form of a library (`solidac.a`) that has to be linked to the existing project. Additionally some header files (.h) have to be included in the project.

5.1 Development Environment

This chapter describes the parameters that must be included in a VxWorks-based project in order to use the solidDB AcceleratorLib option. These parameters can be modified through the Workbench user interface.

5.1.1 Directory Structure

Below is an outline of the files provided with the SDK for VxWorks:

- `/lib`

  This directory contains several libraries. These libraries will already be compiled for your target platform.

  - `solidac.a`: a linkable library which has all the functionality of the solidDB server.
  - `socvxw.a` or `socvpx.a`: the odbc library (not in this release).

- `/doc`

  This directory contains both VxWorks-specific documentation and platform-independent manuals.

- `/include`

  This directory contains header files that are required if you want to compile a program that uses the AcceleratorLib library.

Assuming an existing Workbench/VxWorks project has the following directory structure:

- `//path/`
  - `my_project.wpj` (project file)

  - `Makefile` (make file)
  - `PrjObj.lst` (project object list file)
- `//path/source/` (source code directory)
- `//path/xyzgnu/` (output files directory)

One must include two new directories: One for the library `solidac.a` to be linked with the project and one for the header files (.h) to be included. For example:

- `//path/solid/lib/solidac.a`
- `//path/solidDb5.0/include/sscapi.h, sqltypes.h, sa.h, cli0cli.h. /include` is the directory containing the header files related to:
  - solidDB itself: `sscapi.h`
  - The client APIs:
    - ODBC clients - using SQL: `solidodbc.h`, `sqltypes.h`
    - solidDB SA API: `sa.h`
    - solidDB Light Client API: `cli0cli.h`

The header files `sscapi.h` and `c.h` are mandatory for any client application that is linked to the AcceleratorLib library. Depending on the application's client types (ODBC, SA or light client) you may need to incorporate other headers. For simplicity and consistency, we recommend that you install all the header files mentioned above.

5.1.2 VxWorks Options

The VxWorks kernel must be configured so that it supports:

- Full access (that is, read-write), to a storage medium where at least one directory can be created in order to hold solidDB files: database file, initialisation file, license file, log files (if any), and trace files (if any).
- Floating point unit if one exists on the target processor, or floating point emulation if the target processor does not have a floating point unit.

To build a solidDB application, be sure to include the following as the compiler flags in the Workbench project:

```
-DSS_VXW -DSS_UNIX -DCPU=XXX -I$(SOLIDDIR)/include
```

where

```
XXX = PPC604, PPC403, etc.
SOLIDDIR = base directory where solidDB is installed
```

If software floating point support is required, be sure to include *`-msoft-float`*, and also link with `lib$(XXX)gnuvx.a`. (for example, `libPPC604gnuvx.a`, `libPPC403gnuvx.a`, and so on.)

5.1.3 Link solidac.a

Through the Workbench interface, click on the build tag and display the properties of the project build specifications. Then, click on the Macros tab. In the Macros tab, select the PRJ_LIBS parameters. In the Value field, enter the path to the `solidac.a` library file.

5.1.4 Header Files

Through the Workbench interface, click on the build tag and display the properties of the project build specifications. Then, click on the C/C++ compiler tab. In the option field, enter the path to the header files (.h) directory. Example:

```
-I//path/solidDb5.0/include - .....
```

5.2 Source Code Modifications

In addition to the above modifications, you must also incorporate both project-related #include files and solidDB related #include files in the project's source code.

```
/* Project-related includes. Note: These are examples */
/* and are not related to the Solid AcceleratorLib option. */
#include "VxWorks.h"
#include "prjParams.h"
#include "stdio.h"
#include "stdlib.h"
#include "taskLib.h"
#include "timers.h"
#include "assert.h"
```

```
#include "nfsLib.h"

/* Solid-related includes*/
#include "sscapi.h"
#include "solidodbc3.h"
```

Note

The sa.h and cli0cli.h header files must also be included if you intend to use solidDB SA and/or solidDB light client in the application.

```
/* VxWorks specific */
#define SS_VXW

#if defined(SS_NT)
#ifndef SS_CDECL
#define SS_CDECL __cdecl
#endif
#elif defined(SS_VXW)
#define SS_CDECL
#define main embed_main
#else
#define SS_CDECL
#endif
```

Note

Ensure the SS_VXW compilation flag is defined before the above switch.

```
/* Solid start */
#define USE_XPATHPREFIX
```

5.3 Working Directory

The target system must include a storage medium that allows solidDB to store persistent data. Such a medium can be a local disk, a flash RAM card, or even a remote disk accessed through a network (NFS for example). This requirement must be fulfilled when using the solidDB AcceleratorLib option. Only the diskless option enables designers to run solidDB on a system with no persistent storage capabilities. Throughout this document, the storage medium is referred to as a 'disk' regardless of its actual technology.

Prior to the use of the solidDB AcceleratorLib option, create a working directory on the target machine's disk. The directory must include:

- A valid solidDB license that includes the AcceleratorLib option (on VxWorks), in the form of a `solid.lic` file.
- A solidDB initialisation file, in the form of a `solid.ini` file. This file can be edited and it should contain all solidDB parameters. The following chapter details the tuning of some of these parameters when using the AcceleratorLib option in a VxWorks environment.
- Enough space so that the database files, log files, trace files, and output message files can be created. To create all these files, the minimum size required is approximately 1MB, but of course the size of the database grows as the amount of data grows.

After solidDB starts, the working directory will also include:

- The database file: `solid.db`
- The log files: `sol00001.log`
- The output message file: `solmsg.out`

All these file are created and managed by solidDB and do not require any user intervention.

Note

The `solid.db` file can be stored in another directory that is specified in the `solid.ini` file. The log files are recommended but not mandatory; they can be turned off - that is, not created when running solidDB - when specifying appropriate parameter values in the `solid.ini` file. *solidDB Administration Guide* details the setting of the `solid.ini` file in order to store database file and log files - if enabled - in different directories and under specific names.

Chapter 6. Design Considerations and Performance Tuning Tips

After solidDB has started running, you can control it through the SSC (solidDB Server Control) API. Below are points to consider when tuning solidDB performance. Please note that such tuning is not mandatory and that you can simply start and stop the solidDB server, as explained in the preceding chapters, if you need only minimal control over its behavior.

6.1 Overview

On VxWorks, solidDB is composed of a number of separate but tightly coupled tasks. These tasks are one of the following types:

```
iomgr_thread
SsTimerThread
com_selectthread
thread_rpcserve
thread_sqltask_thread
dnet_clientthread
```

You will see these tasks if you execute the "**i**" command in the VxWorks shell (windsh).

Each client runs as a separate task.

There is always exactly one "local" client. The local client is the client that has the ability to directly control certain aspects of the server by using the SSC (solidDB Server Control) API. This client communicates with the server by means of direct function calls that do not go through the network communication protocols.

In addition to the local client, there may also be zero or more "remote" clients. A remote client is a client that communicates with the server through the network protocol, rather than by direct function calls. Note that "remote" in this sense does not necessarily mean running on a separate target board from the server. In fact, there may be "remote" clients on the same target as the server, and there can also be additional "remote" clients on other target boards.

solidDB and Clients Running as Real-Time Processes

As a rule, solidDB server runs in the kernel mode. Client applications may run in also as Real-Time Processes (RTP) but in this case the client and the server run in different memory space. Because of this, only remote connections can be used when a client runs as RTP.

6.2 Starting the Server

There are two ways to start the server from inside your C program. The first way is to simply call the `SSC-StartServer()`function. `SSCStartServer()` will start all five server tasks, and each of those tasks will inherit the priority of the caller. If you need to control the priority more precisely (that is, you do not want the server to have the same priority as the program that started it), then you can explicitly spawn a task and specify the priority for that task; that task, in turn, will start the server, which inherits the priority of the caller. Both methods are illustrated in the code below.

If you start the program by directly executing the `testMain()` function, then `testMain()` will call `SSCStartServer()`, which will inherit the priority of its caller. this will start the server. The server will inherit the priority of the task that calls `SSCStartServer()`.

```
#include "solidodbc.h"
#include "stdio.h"      /* header file for standard input and output */
#include "assert.h"     /* header file for assert */

#include "sscapi.h"     /* acceleratorLib API header */

/* VxWorks header file for spawning the task */
#include "vxworks.h"
#include "taskLib.h"

#define SOLID_TASK_PRIORITY 40

/* ------------------------------------------------------------------*\
This function starts the server by calling SSCStartServer().
\* ------------------------------------------------------------------*/
int testMain(){
int g_argc;
char *g_argv[5];
SscServerT h;           /* Returns a handle to start the server. This handle
                        * is needed when referencing the server with other
                        * control API functions */
SscRetT src = SSC_SUCCESS;  /* SscRetT is of type enum used to
                        * catch API return code */

g_argc = 5;
g_argv[0] = "";
g_argv[1] = "-xpathprefix:/ram";
/* Changes the dir to "/ram/" to find solid.ini */
g_argv[2] = "-Udba";    /* User name = "dba" */
g_argv[3] = "-Pdba";    /* Password = "dba" */
g_argv[4] = "-Cdba";    /* Catalog name = "dba" */

/* API call to start the server */
src=SSCStartServer(g_argc, g_argv, 'h, SSC_STATE_OPEN);
printf("Return code is %d", src);
assert(src==SSC_SUCCESS);   /* check the status of the server */
...
} /* End of testMain */



/* ------------------------------------------------------------------*\
This function starts the server by first spawning a separate task at a
specified priority; that task then calls SSCStartServer(). This allows
you to indirectly specify the priority at which the server will run.
\* ------------------------------------------------------------------*/
void StartSrv()
{
taskSpawn ("tSolid", SOLID_TASK_PRIORITY, VX_FP_TASK, 20000,
                    (FUNCPTR)testMain,0,0,0,0,0,0,0,0,0,0);
}
```

Once you have this program compiled into an object module, there are two ways to execute that object module. From the windsh, you may execute the command:

testMain

or the command:

sp StartSrv

If you call `testMain` directly, the five server tasks will get the default priority assigned by VxWorks. If you call `startSrv`, the server will get the priority specified in the `taskSpawn()` call in the `StartSrv()` function.

Although the server is composed of five separate tasks, you only need to spawn one task; the server itself will take care of the rest. The spawner itself is a separate task. The spawner may exit as soon as it has spawned the server; the spawner does not need to remain in memory. The spawner may, however, continue running and may become one of the clients, if desired.

In both cases, all five tasks that comprise the server have the same priority. There is no way to give each of the five tasks a separate priority.

In the VxWorks system, the object modules of the server and the clients may be linked directly with the OS as a single object module, or they may be downloaded separately from the OS and then run. For more details, see Appendix B, *Different Approaches to "Building Your Image"*, and especially Section B.3, “Possible Combinations of Images”.

6.3 Tuning a Multi-threaded solidDB Server

You may add additional general-purpose worker threads to solidDB. The *`thread`* parameter in the *`[srv]`* section of the `solid.ini` file defines the number of these 'general purpose' internal worker threads. For example:

```
[Srv]
Threads = 2
```

This parameter setting will provide the designer with two general purpose threads. General-purpose threads handle the following tasks:

- checkpoint operations
- merge operations
- backup operations
- SQL operations (that is, responding to client SQL queries)

If you list all the tasks executing on VxWorks, you will see one task named *`"thread_sqlt"`* for each general-purpose thread that you specified in the `solid.ini` file.

Note

These threads are server threads. They are created automatically by the solidDB server task. They are different from, and not related to, the client tasks.

6.4 SSCAdvanceTasks

The primary purpose of the `SSCAdvanceTasks` function is to allow the application to yield some of its CPU time slice to server tasks, such as checkpointing. `SSCAdvanceTasks` also returns some useful status information.

In some cases, you can improve performance by yielding the CPU when you are not using it.

Note that calls to `SSCAdvanceTasks` are synchronous; the server does not generally return to you immediately, since you are yielding the CPU.

6.4.1 SSCAdvanceTasks, and checking Server state

Because `SSCAdvanceTasks` returns a small amount of status information, the function can also be used to monitor the server, as well as yield the CPU. The caller can choose what to do depending on the solidDB server status it returns. Note that all calls to this function are synchronous -- in other words, the client thread that called this function does not proceed until the function returns.

Important

All calls to the `SSCAdvanceTasks` function yield the CPU. Do not call this function to get status information unless you are also willing to yield the CPU.

Example 6.1. Using SSCAdvanceTasks() to Monitor the Server

The example below illustrates the use of `SSCAdvanceTasks()` to monitor the server (as well as yield CPU time):

```
/* In this example, runState is an application-defined flag */
/* to control how long the application runs. */
while (runState == ALL_GO)
{
        serverState = SSCAdvanceTasks(h, 100);
        if (serverState == SSC_SERVER_NOTRUNNING)
        {
            runState = ALL_STOP;
            printf("Server dead \n");
        }
        /* My custom function to load a table etc. */
        loadMySolidTable(scon, scur, mySaDate);
}
```

solidDB AcceleratorLib User Guide describes the `SSCAdvanceTasks` function, including its input parameters and return codes, in more detail.

6.5 Adjusting Stack Size

The stack size for tasks started by solidDB can be preset before starting the server. The stack size is configured using the SSC API call `SSCSetStackSize(int stackSize)`. If the function is called after starting the server, the call will only change the stack size of those tasks that were started after configuring the stack size.

Default value of the stack size is 128 KB.

For example, to set the stack size to 2 KB:

```
SSCSetStackSize(20*1024)
```

For details on `SSCSetStackSize(int stackSize)`, see Section C.3, “SSCSetStacksize”.

6.6 Assigning Memory Partitions for solidDB

To avoid fragmentation of the whole memory of the VxWorks system, solidDB can be configured to use memory partitions. In such a case solidDB only uses memory available in a given memory partition.

Note

Some operating system resources may cause memory allocations which claim memory from global pool.

The memory pool is created using the `SsSysMemGlobalInit(unsigned long sz)` function. This reserves memory pool of the size defined with `sz`.

Subsequent calls to `SsSysMemGlobalInit(unsigned long sz)` will cause memory pool size to be increased by the size defined with `sz`.

Memory pool is deleted using command `SsSysMemGlobalDone()`. This is not necessary after shutting down the server - the server can use the previously allocated memory partition.

When memory pools are used, `SsSysMemGlobalInit(unsigned long sz)` must be called before calling any other solidDB function.

By default solidDB allocates memory from global memory pool.

6.7 Reclaiming System Resources and Uninitializing the Server

solidDB maintains a catalog of all system resources it has allocated (sockets, semaphore objects, memory). After shutting down the server, these resources can be given back to server.

After returning the resources back to the operating system, several solidDB internal variables must be set to their initial state. This can be done using `SsSysResGlobalDone()` function. In normal use, it is not necessary to call the `SsSysResGlobalDone()` function between the shutdown and restart of the server; if the `SsSysResGlobalDone()` function is not called, solidDB continues to use the previously allocated resources.

6.8 Selecting Memory Allocator

By default solidDB server uses its internal memory allocator which enables task level and solidDB level memory pooling. However, if you want to use only solidDB level memory pooling or disable solidDB wide memory pooling, you can do this by using the `SSCSetMemoryAllocatorType` SSC API function.

The default is to use task level and solidDB level memory pooling.

From performance perspective, the fastest option is to use the default memory allocator. However, with the default memory allocator, the memory consumption for normal operation is the highest as each server task has its own local freelists for memory. Additionally there are global freelists.

The global solidDB wide memory pooling is slightly slower than the local task level memory model. It also consumes a slightly less memory because each task does not have its own freelist.

The slowest model is to disable all solidDB internal memory pooling; in this case, all allocations result always in an allocation from the operating system and no internal freelists are maintained. This model also consumes the least amount of memory.

For more information on the `SSCSetMemoryAllocatorType` function, see Section C.2, “SSCSetMemory-AllocatorType”.

6.9 Redirecting Server Messages

By default solidDB writes output messages about its operation to solmsg.out, solerror.out, soltrace.out, and ssdebug.out.

The solidDB server can be configured to redirect these messages to a user-defined callback function instead of the default files. This can be done using the `SSCLogHookAdd(SSCLogHookT hook)` function where `hook` is a callback function.

The default operation is restored by setting hook to NULL.

For example:

```
...
#include sscapi.h

...
void hook(ss_msglog_type_t t, const char* fmt, ...)
{
 char buf[512];
         va_list ap;
         va_start(ap, fmt);
         vsnprintf(buf, 512, fmt, ap);
         va_end(ap);
         syslog(LOG_INFO, "%s", buf); /* unix like syslog facility used in exampl
}
...
 SSCLogHookAdd(hook); /* set log to hook */
...
 SSCLogHookAdd(NULL); /* restore default behavior */
...
```

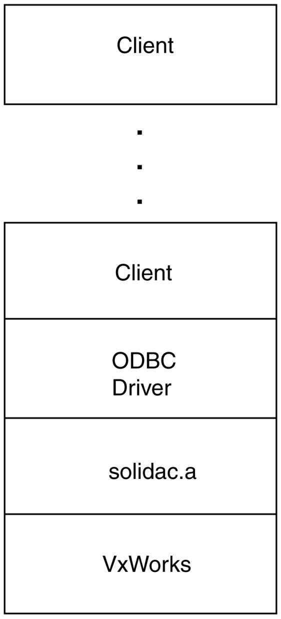
For more information on the `SSCLogHookAdd` function, see Section C.1, “SSCLogHookAdd”.

Chapter 7. ODBC Client

This chapter details the requirements in order to connect an ODBC client to the solidDB server.

ODBC clients may be local or remote. A client is "local" if it is compiled and linked to the AcceleratorLib library. Such a client runs on the same VxWorks target as the server. When a local client makes an ODBC call, the call goes either directly or through the TCP/IP communications protocol to the server. "Remote" clients run on a different VxWorks target board than the server runs on, and use the ODBC driver, which uses TCP/IP rather than direct calls to communicate with the server. The figure below illustrates the use of ODBC with one or more local clients.

Figure 7.1. Local Clients and ODBC Driver



Note that only one copy of the ODBC driver is required.

A single application can contain several independent clients, each of which has one or more connections to the server. Each of the clients may handle different data and executes different kinds of operations using different statements. Each ODBC client is essentially a task spawned by the application. The paragraphs below describe the essential steps common to all clients.

7.1 Connection

To start, each task using solidDB memory management must register itself. Otherwise the solidDB memory management is not aware of the client. A client can register itself by using the call below:

```
/* Register a user thread */
SscRetT SSC_CALL SSCRegisterThread(
            SscServerT h);
```

Next, each client must instantiate three ODBC handles:

```
SQLHENV     henv;   /* ODBC ENVironment handle */
SQLHDBC     hdbc;   /* ODBC DataBase Connection handle */
SQLHSTMT    hstmt;  /* ODBC STateMenT handle */
```

The data types must be the ones shown above, but the variable names do not have to match.

The SQLHENV is an ODBC environment handle. It contains all information for the other two handles. It can be a global handle, shared among several tasks, or it can be defined locally for each ODBC client. The latter is the recommended solution in order to maintain the simplicity and portability of each client.

The SQLHDBC is an ODBC connection handle that points to an area that contains all information related to the ODBC connection: Connect string, username, password, and so on.

The SQLHSTMT is an ODBC statement handle that points to an area that contains all information related to the active ODBC statement.

An ODBC client uses these handles to establish a connection to solidDB and allocate resources required to create ODBC statements. The following example illustrates this:

```
void client1 (void)
{
SQLHENV henv = (SQLHENV) NULL;
SQLHDBC hdbc = (SQLHDBC) NULL;
SQLHSTMT hstmt = (SQLHSTMT) NULL;

/* Allocate environment */
SQLAllocEnv('henv);
SQLAllocConnect(henv, 'hdbc);
/* Note that the client is local, i.e. bypasses the RPC */
/* mechanism.  The connect string is "localserver". The */
/* username and password are "dba") */
SQLConnect(hdbc, "localserver", SQL_NTS, "dba", SQL_NTS, "dba", SQL_NTS);
SQLAllocStmt(hdbc, 'hstmt);
printf("\n Client1 connected, statement allocated\n\n");
}
```

7.2 Statement Execution

In this section, statements are defined and executed using the *`hstmt`* variable and appropriate ODBC functions. This is usually implemented in the form of an endless loop -- that is, as long as the client is alive, it processes statements. It can be the same identical statement but the statement handle can be closed, and its parameters can be reset without being freed. Thus it can be reused again. The example below shows part of the client1 function to illustrate the statement execution phase. Nothing in this example is specific to VxWorks; ODBC clients on other platforms go through these same steps.

```
RETCODE rc;
SQLINTEGER id;

while (runState == ALL_GO)
{
/* Prepare and execute statement */
rc = SQLExecDirect(hstmt, (SQLCHAR*)"SELECT COUNT (*) FROM SYNCDEMO", SQL_NTS);
CHECK(rc);
/* Bind the variable named "id" to the first column of the result set. */
rc = SQLBindCol(hstmt, 1, SQL_C_SLONG, 'id, 0, NULL);
CHECK(rc);
rc = SQLFetch(hstmt);
CHECK(rc);
printf("Number of rows in SYNCDEMO table is (%d) \n", id);

/* Reset statement*/
rc = SQLFreeStmt(hstmt, SQL_RESET_PARAMS);
CHECK(rc);
/* Unbind variables*/
rc = SQLFreeStmt(hstmt, SQL_UNBIND);
CHECK(rc);
/* Free statement*/
rc = SQLFreeStmt(hstmt, SQL_CLOSE);
CHECK(rc);

/* Now feel free to add whatever logic suits your application. You may */
/* re-run the statement with the same variable or define a new statement. */

}
```

7.3 Disconnecting

When the application decides to close a client, it must unregister itself from solidDB memory management. A client can unregister itself by using the call below:

```
/* Unregister a user thread */
SscRetT SSC_CALL SSCUnregisterThread(
            SscServerT h);
```

When the application decides to close a client, it must disconnect it and release the handles as shown below:

```
SQLDisconnect(hdbc);
SQLFreeConnect(hdbc);
SQLFreeEnv(henv);
```

Chapter 8. solidDB VxWorks Development Differences (from Other Platforms)

If you are already familiar with developing solidDB based applications on non-embedded systems, you may benefit from knowing some of the differences between developing on VxWorks compared to non-embedded systems. Some of the key differences are listed below:

- Some platforms, including VxWorks, run the operating system and all applications (including solidDB) as a single "process". Within that process, there are separate tasks. solidDB runs as a separate task(s) from your application.
- On VxWorks, solidDB is provided not as a standalone executable program, but as a library (named `solidac.a`), which can be linked into the operating system, and run as part of a single process.
- Because there is only one process, there is only one address space. All symbols must be unique within that address space. Thus every task cannot have its own function named `main()`. Each task's "entry routine" must have a unique name.
- Because all "programs" running under VxWorks share a global name space, every function name in the global namespace must be unique. This means that:
  - You cannot run more than one copy of the AcceleratorLib server at a time on a single VxWorks computer. Each of the function names in the `solidac.a` file can only exist once in a VxWorks image.
  - You should write re-entrant code for your client applications.
- Because the solidDB server subroutines (the AcceleratorLib library) and your application are all part of the same process as the OS, a failure in your application or in the solidDB server will crash the whole system.
- Avoid using the C-language assert() macro. It is difficult to completely avoid use of this macro (even the solidDB solidac library uses this occasionally) but its use should be avoided as much as possible.
- solidDB does not gracefully handle "out-of-memory" errors in some situations. It is important not to overload solidDB.

Appendix A. A Quick Overview of Developing Applications

This chapter provides a quick overview of key things to remember when developing applications.

- The solidDB server is supported in the form of AcceleratorLib. A pure "executable" version of the server is not provided, so you must link the AcceleratorLib library.
- solidDB is a multi-tasking server; the AcceleratorLib server runs on multiple VxWorks tasks instead of a single big task.
- solidDB has multitasking support for user applications. User applications can be implemented and run on multiple VxWorks tasks, all within the same context as the server.
- The solidDB server AcceleratorLib library file, `solidac.a`, is delivered as an archived library.
- A typical configuration for boot image would be linking VxWorks, user application, and AcceleratorLib library together to boot VxWorks, bring up AcceleratorLib, and (or) run user applications automatically during boot time.

  We recommend that if you choose to create separate loadable modules for the VxWorks boot image and the application image, then you link `solidac.a` with the application instead of with VxWorks. This ensures that the symbols in the user applications are resolved and that the AcceleratorLib library will be pulled in.
- You must start the AcceleratorLib server by calling `SSCStartServer()` or `SQLConnect()`, and stop the AcceleratorLib server by calling `SSCStopServer()` or by using ADMIN COMMAND 'shut-down'.

  Alternatively, if there is already a valid database and license file in the startup directory, an `SQLConnect()` call from the client will also start the server without a call to `SSCStartServer()`. An `SQLDisconnect()` call will stop the AcceleratorLib server if no other connection to the database exists.

  User applications that start and stop the server can be spawned from the `usrAppInit.c`, or separately loaded and started. Correct settings, such as start-up directory, username, password, and catalog name, are still needed before making the `SSCStartServer()` call.
- If the client applications run locally on the same target board as the AcceleratorLib server runs on, then the only solidDB library that the client needs to link with is the AcceleratorLib library, `solidac.a`.

However, if the client applications were to run remotely on another VxWorks-based board and interface with a server running somewhere else, then linking the application with the ODBC driver library would be necessary.

You may also wish to look at the samples, such as the one in the samples/multitasking directory. This sample demonstrates an application that starts the server by calling the `SSCStartServer` function.

Appendix B. Different Approaches to "Building Your Image"

or

How to Resolve "UNRESOLVED SYMBOL" Errors

B.1 Abstract

This section explains how to recognize and prevent problems during the "load and resolve symbols" step of running a program on VxWorks. This problem can occur if the solidDB server library (solidac.a) and your application are in separate images.

This problem is most likely to occur if you are migrating from an old solidDB version and if you are are used to downloading solfe.o separately from your application program.

B.2 Background

In general, to execute a program on VxWorks, you build an "image" that contains the code that you want to execute, then you download that code to the target computer and execute the code on the target.

Building the image requires two major steps:

1. compiling, and

2. linking.

"Downloading" also requires two major steps:

1. Copying the code to the target computer, and

2. using the "loader" to resolve symbols as the code is copied into the target's memory to be executed.

If function `A()` calls function `B()` and if function `B()` is not present in the target's memory, then the loader will be unable to resolve the symbol (translate it to an address) and therefore the loader will give an error rather than load the image that contains function `A()`. Depending upon how many images you use and how you combine code in those images, you may get error messages about unresolved symbols during the load-and-resolve-symbols phase.

B.3 Possible Combinations of Images

In this section, "building" refers to compiling and linking, and "downloading" refers to copying the image and doing loading/symbol-resolution.

There are many possible variations on this basic scenario. For example, the image can be stored in ROM so that the target machine does not need to be connected to the host.

For the moment, we will ignore the issue of how you deploy the code and whether you put it in ROM. Instead, we will assume that you are in development mode and that each time you want to execute your program you will build it on the host machine, then download it to the target machine and execute it.

Even within this approach, there are several possible variations, depending upon whether you want to create a single image that contains "everything" (your application, the solidDB AcceleratorLib library (`solidac.a`), and VxWorks itself) or whether you want to create two or three separate images. Because the linker is a "smart" linker, when it links the code it will include only the subroutines that have been called somewhere. If you compile and link your application and the solidDB server library separately, you might find that some code has been excluded because during the compile and link step the linker did not find any references to that code.

We recommend that you use one of the following two approaches to creating your images:

1. One image: VxWorks + `solidac.a` + your application.

2. Two separate images, where your application and the AcceleratorLib library are combined in one image and VxWorks is the other image.

Figure B.1. One image: VxWorks + solidac.a + your application

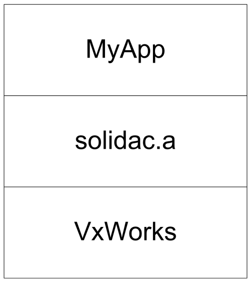
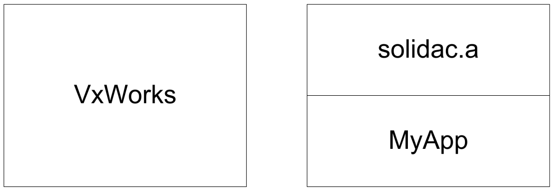


Figure B.2. Two separate images: VxWorks, solidac.a + your application



In the case of two separate images, your application and the AcceleratorLib library are combined in one image and VxWorks is the other image.

Creating a single image that contains all three "pieces" is the simplest approach. The compiler and linker will create an image that contains all the subroutines that are referenced by any of the three "pieces". During the load-and-resolve-symbols phase, there will not be any unresolved symbols. Similarly, if you put your application and the solidac code in the same image, this prevents errors due to unresolved symbols.

There are several other theoretically possible combinations, but the two described above are usually the most appropriate. If you create three separate images, then at the time that the complier/linker creates the image with `solidac.a`, the compiler/linker does not know which routines your application is going to call. The linker is likely to discard `solidac.a` subroutines that your application will need. You will be able to compile and link correctly, but at the time that you download (load and resolve symbols), the loading process will fail. A similar problem can occur if you create either of the following "pairs" of images:

a. VxWorks + `solidac.a` (one image). Your application (the other image).

b. VxWorks + your application (one image). `solidac.a` (the other image).

If you want to put your application and the `solidac.a` code in separate images, you may do so, but you will need to create "dummy" calls to all the routines that your application needs in `solidac.a`, and you will need to compile and link those dummy calls into the same image as you link `solidac.a` into.

Appendix C. SSC API Functions

The SSC API (solidDB Control API) allows you to, for example, set the priority for various tasks that run within the server.

C.1 SSCLogHookAdd

The `SSCLogHookAdd` function sets a logging hook.

C.1.1 Synopsis

```
SscRetT SSC_CALL SSCLogHookAdd(
    SSCLogHookT hook);
```

Where `SSCLogHookT` is of the following type:

```
void (*SSCLogHookT)(SscLogType log_type, const char* fmt, ...);
```

The `SSCLogHookAdd` function accepts the following parameters:

Table C.1. SSCLogHookAdd Parameters

Parameters	**Usage Type**	**Description**
`SscLogType`	in	The `SSCLogHookAdd` function sets the *`SscLogType`* parameter value for the callback function. The application can use the *`SscLog-Type`* parameter value to decide upon the further processing of the log message. Possible values are: • SSC_LOG_UNDEFINED • SSC_LOG_SOLMSG • SSC_LOG_SOLERROR • SSC_LOG_SOLTRACE • SSC_LOG_SSDEBUG

C.1.2 Return Value

- SSC_SUCCESS

C.2 SSCSetMemoryAllocatorType

The `SSCSetMemoryAllocatorType` function sets the memory allocator type. This must be done before any other call to solidDB functions. This is a global setting and it cannot be changed after the first memory allocation has been done within the solidDB functions.

C.2.1 Synopsis

```
SscRetT SSC_CALL SSCSetMemoryAllocatorType(
    SscMemoryAllocatorType type);
```

The memory allocator type is any of the following:

- SSC_MEMALLOC_THREADLOCAL. Use thread local memory allocator. The user must register and unregister threads to avoid excessive memory leaks. This memory allocation method gives best performance on multi-threaded systems.
- SSC_MEMALLOC_GLOBAL. Use global memory allocator. All threads use the same memory pools for memory management.
- SSC_MEMALLOC_SYSMEM. Use malloc, realloc, calloc, and free functions directly. This option results in worst performance.

C.3 SSCSetStacksize

The `SSCSetStacksize` function sets the stack size for the tasks spawned by solidDB server.

C.3.1 Synopsis

```
int SscSetStackSize(int sz)
```

C.3.2 Return Value

New value of the stack size.

C.4 Conversion Functions

There are four conversion functions. The conversion function syntaxes are described below:

C.4.1 Synopsis

```
int SSC_CALL SSCTaskClass2Str(
            SscTaskSetT tasktype,
            char** p_strtaskclass);

int SSC_CALL SSCStr2TaskClass(
            char* strtaskclass,
            SscTaskSetT* p_tasktype);

int SSC_CALL SSCPrio2Str(
            SscTaskPrioT prio,
            char** p_strprio);

int SSC_CALL SSCStr2Prio(
            char* strprio,
            SscTaskPrioT* p_prio);
```

Appendix D. SsSys Utility Functions

The SsSys utility functions allows you to control solidDB memory and resource usage.

D.1 SsSysMemGlobalInit

The `SsSysMemGlobalInit` creates or expands the memory pool used by solidDB.

D.1.1 Synopsis

```
void* SsSysMemGlobalInit(unsigned long sz)
```

Where `sz` is the size of memory pool or the size of the increment of the memory pool.

D.1.2 Return Value

- Pointer to memory pool.
- NULL if operation failed.

D.2 SsSysMemGlobalDone

The `SsSysMemGlobalDone` function deletes the memory partition created by `SsSysMemGlobalInit()`. It also frees the memory in a partition.

D.2.1 Synopsis

```
void SsSysMemGlobalDone(void)
```

D.3 SsSysResGlobalDone

The `SsSysResGlobalDone` function frees the resources allocated from operating system and resets the internal variables of solidDB.

D.3.1 Synopsis

```
void SsSysResGlobalDone(void)
```

Glossary

This glossary gives you a description of the terminology used in this guide.

A

Accelerated application

An "Accelerated application" is an application program that has been linked with the solidDB AcceleratorLib option library. The application must explicitly start the server by calling `SSCStartServer()`. The application and server will execute as separate tasks; however, the application has some control over the server, such as the ability to stop and restart the server.

Application Programming Interface (API)

An API is an Application Programming Interface. In other words, it is a set of functions that you may call. The solidDB AcceleratorLib option is an API; solidDB provides a set of C language .h files that define the functions that you may call, and solidDB also provides a pre-compiled library of functions, including of course the functions specified in the .h files.

C

Checkpoint

Checkpoints are used to store a consistent state of the database quickly onto the disk. After a system crash, the database can re-start using the data that was saved at the most recent successful checkpoint.

D

Database administrator (DBA)

The database administrator is a person responsible for tasks such as:

- managing tables, and indices
- backing up data
- allocating disk space for the database files

Database management system (DBMS)

A DBMS is a system that stores information in and retrieves information from a database. A DBMS typically consists of a database server, administration utilities, an application interface, and development tools.

H

Host
: When developing for embedded systems, such as VxWorks, the "host" machine is the machine on which you will do development work. When doing development for Wind River VxWorks, for example, the host machine contains Wind River's Workbench development environment for VxWorks developers. The code that you build is actually executed on the "target" machine. (See also the definition of "target".)

I

Image
: In the context of VxWorks, an image is a set of executable code that exists as a single entity. It is the result of compiling and linking all the code into a single executable. During development, this executable is usually downloaded from the host to the target and then executed. During deployment, this image may be burned into ROM so that it is always available as soon as the computer boots. See the Wind River documentation for more details.

O

Open Database Connectivity (ODBC)
: ODBC is a programming interface standard for SQL database programs.

R

Relational database management system (RDBMS)
: solidDB is an RDBMS, which stores and retrieves information that is organized into two-dimensional tables. This name derives from the relational theory that formalizes the data manipulation requests as set operations and allows mathematical analysis of these sets.

Relational database management system (RTOS)
: Real-Time Operating System

S

solidDB Developers' Kit (SDK)
: solidDB Developers' Kit. This contains the solidDB server (either as an executable program or as a subroutine library that can be linked), along with sample programs.

T

Target

When developing for VxWorks, the "target" machine is the machine that solidDB will run on (that is, the VxWorks machine). This may be different from the "host" machine, on which the program was developed. (See also the definition of "host".)

Task

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

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

Index

A

Accelerated application, 49
Accelerator library , 9
API
  defined, 49

B

boot image , 37

C

Checkpoint, 49
conversion functions , 45

D

Database administrator, 49
Database management system, 49
DBA
  defined, 49
DBMS
  defined, 49

H

Host
  defined, 50

I

Image
  defined, 50

O

ODBC
  defined, 50

R

RDBMS
  defined, 50
RTOS
  defined, 50

S

SDK
  defined, 50
solidac.a , 9, 37
SQLConnect , 37
SQLDisconnect , 37
SSC API Functions , 43
SSCLogHookAdd
  defined , 43
  parameters , 43
  return value , 44
  synopsis , 43
SSCPrio2Str
  synopsis , 45
SSCSetMemoryAllocatorType
  defined , 44
  synopsis , 44
SSCSetStacksize
  defined , 44
  synopsis , 44
SSCStartServer , 37
SSCStr2Prio
  synopsis , 45
SSCStr2TaskClass
  synopsis , 45
SSCTaskClass2Str
  synopsis , 45
SsSys Utility Functions , 47
SsSysMemGlobalDone
  defined , 47
  synopsis , 47
SsSysMemGlobalInit
  defined , 47
  return value , 47
  synopsis , 47
SsSysResGlobalDone
  defined , 47
  synopsis , 48

T

Target
    defined, 51
Task
    defined, 51