



Programmer's Guide

Monitor Client Library

12.5

DOCUMENT ID: 32865-01-1250-01

LAST REVISED: May 2001

Copyright © 1989-2001 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase database management software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor. Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase, the Sybase logo, ADA Workbench, Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server, Adaptive Server Anywhere, Adaptive Server Enterprise, Adaptive Server Enterprise Monitor, Adaptive Server Enterprise Replication, Adaptive Server Everywhere, Adaptive Server IQ, Adaptive Warehouse, AnswerBase, Anywhere Studio, Application Manager, AppModeler, APT Workbench, APT-Build, APT-Edit, APT-Execute, APT-FORMS, APT-Translator, APT-Library, Backup Server, ClearConnect, Client-Library, Client Services, Data Pipeline, Data Workbench, DataArchitect, Database Analyzer, DataExpress, DataServer, DataWindow, DB-Library, dbQueue, Developers Workbench, Direct Connect Anywhere, DirectConnect, Distribution Director, E-Anywhere, E-Whatever, Embedded SQL, EMS, Enterprise Application Studio, Enterprise Client/Server, Enterprise Connect, Enterprise Data Studio, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, EWA, Financial Fusion, Financial Fusion Server, Gateway Manager, ImpactNow, InfoMaker, Information Anywhere, Information Everywhere, InformationConnect, InternetBuilder, iScript, Jaguar CTS, jConnect for JDBC, KnowledgeBase, MainframeConnect, Maintenance Express, MAP, MDI Access Server, MDI Database Gateway, media.splash, MetaWorks, MySupport, Net-Gateway, Net-Library, ObjectConnect, ObjectCycle, OmniConnect, OmniSQL Access Module, OmniSQL Toolkit, Open Client, Open ClientConnect, Open Client/Server, Open Client/Server Interfaces, Open Gateway, Open Server, Open ServerConnect, Open Solutions, Optima++, PB-Gen, PC APT Execute, PC DB-Net, PC Net Library, Power++, power.stop, PowerAMC, PowerBuilder, PowerBuilder Foundation Class Library, PowerDesigner, PowerDimensions, PowerDynamo, PowerJ, PowerScript, PowerSite, PowerSocket, Powersoft, PowerStage, PowerStudio, PowerTips, Powersoft Portfolio, Powersoft Professional, PowerWare Desktop, PowerWare Enterprise, ProcessAnalyst, Report Workbench, Report-Execute, Replication Agent, Replication Driver, Replication Server, Replication Server Manager, Replication Toolkit, Resource Manager, RW-DisplayLib, RW-Library, S-Designor, SDF, Secure SQL Server, Secure SQL Toolset, Security Guardian, SKILS, smart.partners, smart.parts, smart.script, SQL Advantage, SQL Anywhere, SQL Anywhere Studio, SQL Code Checker, SQL Debug, SQL Edit, SQL Edit/TPU, SQL Everywhere, SQL Modeler, SQL Remote, SQL Server, SQL Server Manager, SQL SMART, SQL Toolset, SQL Server/CFT, SQL Server/DBM, SQL Server SNMP SubAgent, SQL Station, SQLJ, STEP, SupportNow, Sybase Central, Sybase Client/Server Interfaces, Sybase Financial Server, Sybase Gateways, Sybase MPP, Sybase SQL Desktop, Sybase SQL Lifecycle, Sybase SQL Workgroup, Sybase User Workbench, SybaseWare, Syber Financial, SyberAssist, SyBooks, System 10, System 11, System XI (logo), SystemTools, Tabular Data Stream, Transact-SQL, Translation Toolkit, UNIBOM, Unilib, Uninull, Unisep, Unistring, URK Runtime Kit for UniCode, Viewer, Visual Components, VisualSpeller, VisualWriter, VQL, WarehouseArchitect, Warehouse Control Center, Warehouse Studio, Warehouse WORKS, Watcom, Watcom SQL, Watcom SQL Server, Web Deployment Kit, Web.PB, Web.SQL, WebSights, WebViewer, WorkGroup SQL Server, XA-Library, XA-Server and XP Server are trademarks of Sybase, Inc. 3/01

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names used herein may be trademarks or registered trademarks of their respective companies.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608.

Contents

| | |
|---|-----------|
| About This Book | xi |
| | |
| CHAPTER 1 Getting started with Monitor Client Library | 1 |
| Overview | 1 |
| What is Adaptive Server Enterprise Monitor | 1 |
| Adaptive Server Enterprise Monitor components | 2 |
| Adaptive Server Enterprise Monitor architecture | 2 |
| Writing a Basic Monitor Client Library program | 4 |
| Application logic flow | 5 |
| Step 1: define error handling | 5 |
| Step 2: connect to a server | 6 |
| Step 3: create a view | 7 |
| Step 4: create filters | 10 |
| Step 5: set alarms | 11 |
| Step 6: request performance data and process results | 12 |
| Step 7: close and deallocate connections | 13 |
| Playing back recorded data | 13 |
| A sample Monitor Client Library program | 14 |
| Example program | 14 |
| | |
| CHAPTER 2 Data Items and Statistical Types | 43 |
| Overview | 43 |
| Result and key data items | 43 |
| Data items and views | 44 |
| Rows with no data versus no rows in views | 45 |
| Server-level status | 45 |
| Combining data items | 46 |
| Result and key combinations | 46 |
| Connection summaries | 46 |
| Current statement and application name data items | 46 |
| Data item definitions | 47 |
| Deciphering the names of data items | 48 |
| SMC_NAME_ACT_STP_DB_ID | 49 |

| | |
|---|----|
| SMC_NAME_ACT_STP_DB_NAME | 50 |
| SMC_NAME_ACT_STP_ID | 50 |
| SMC_NAME_ACT_STP_NAME | 51 |
| SMC_NAME_ACT_STP_OWNER_NAME | 51 |
| SMC_NAME_APPLICATION_NAME | 52 |
| SMC_NAME_APP_EXECUTION_CLASS | 53 |
| SMC_NAME_BLOCKING_SPID | 53 |
| SMC_NAME_CONNECT_TIME | 54 |
| SMC_NAME_CPU_BUSY_PCT | 54 |
| SMC_NAME_CPU_PCT | 55 |
| SMC_NAME_CPU_TIME | 55 |
| SMC_NAME_CPU_YIELD | 56 |
| SMC_NAME_CUR_APP_NAME | 56 |
| SMC_NAME_CUR_ENGINE | 57 |
| SMC_NAME_CUR_EXECUTION_CLASS | 57 |
| SMC_NAME_CUR_PROC_STATE | 57 |
| SMC_NAME_CUR_STMT_ACT_STP_DB_ID | 58 |
| SMC_NAME_CUR_STMT_ACT_STP_DB_NAME | 59 |
| SMC_NAME_CUR_STMT_ACT_STP_ID | 59 |
| SMC_NAME_CUR_STMT_ACT_STP_NAME | 60 |
| SMC_NAME_CUR_STMT_ACT_STP_OWNER_NAME | 60 |
| SMC_NAME_CUR_STMT_ACT_STP_TEXT | 61 |
| SMC_NAME_CUR_STMT_BATCH_ID | 61 |
| SMC_NAME_CUR_STMT_BATCH_TEXT | 62 |
| SMC_NAME_CUR_STMT_BATCH_TEXT_ENABLED | 62 |
| SMC_NAME_CUR_STMT_CONTEXT_ID | 63 |
| SMC_NAME_CUR_STMT_CPU_TIME | 63 |
| SMC_NAME_CUR_STMT_ELAPSED_TIME | 64 |
| SMC_NAME_CUR_STMT_LINE_NUM | 64 |
| SMC_NAME_CUR_STMT_LOCKS_GRANTED_IMMEDIATE | 65 |
| SMC_NAME_CUR_STMT_LOCKS_GRANTED_WAITED | 65 |
| SMC_NAME_CUR_STMT_LOCKS_NOT_GRANTED | 65 |
| SMC_NAME_CUR_STMT_NUM | 66 |
| SMC_NAME_CUR_STMT_PAGE_IO | 66 |
| SMC_NAME_CUR_STMT_PAGE_LOGICAL_READ | 67 |
| SMC_NAME_CUR_STMT_PAGE_PHYSICAL_READ | 67 |
| SMC_NAME_CUR_STMT_PAGE_WRITE | 68 |
| SMC_NAME_CUR_STMT_QUERY_PLAN_TEXT | 68 |
| SMC_NAME_CUR_STMT_START_TIME | 69 |
| SMC_NAME_CUR_STMT_TEXT_BYTE_OFFSET | 69 |
| SMC_NAME_DATA_CACHE_CONTENTION | 70 |
| SMC_NAME_DATA_CACHE_EFFICIENCY | 70 |
| SMC_NAME_DATA_CACHE_HIT | 70 |
| SMC_NAME_DATA_CACHE_HIT_PCT | 71 |

| | |
|---|----|
| SMC_NAME_DATA_CACHE_ID | 71 |
| SMC_NAME_DATA_CACHE_LARGE_IO_DENIED | 72 |
| SMC_NAME_DATA_CACHE_LARGE_IO_PERFORMED | 73 |
| SMC_NAME_DATA_CACHE_LARGE_IO_REQUESTED | 73 |
| SMC_NAME_DATA_CACHE_MISS | 74 |
| SMC_NAME_DATA_CACHE_NAME | 74 |
| SMC_NAME_DATA_CACHE_PREFETCH_EFFICIENCY | 75 |
| SMC_NAME_DATA_CACHE_REUSE | 75 |
| SMC_NAME_DATA_CACHE_REUSE_DIRTY | 76 |
| SMC_NAME_DATA_CACHE_REF_AND_REUSE | 76 |
| SMC_NAME_DATA_CACHE_SIZE | 77 |
| SMC_NAME_DB_ID | 77 |
| SMC_NAME_DB_NAME | 78 |
| SMC_NAME_DEADLOCK_CNT | 78 |
| SMC_NAME_DEMAND_LOCK | 79 |
| SMC_NAME_DEV_HIT | 79 |
| SMC_NAME_DEV_HIT_PCT | 79 |
| SMC_NAME_DEV_IO | 80 |
| SMC_NAME_DEV_MISS | 80 |
| SMC_NAME_DEV_NAME | 81 |
| SMC_NAME_DEV_READ | 81 |
| SMC_NAME_DEV_WRITE | 82 |
| SMC_NAME_ELAPSED_TIME | 82 |
| SMC_NAME_ENGINE_NUM | 83 |
| SMC_NAME_HOST_NAME | 83 |
| SMC_NAME_KPID | 84 |
| SMC_NAME_LOCK_CNT | 84 |
| SMC_NAME_LOCK_HIT_PCT | 85 |
| SMC_NAME_LOCK_RESULT | 85 |
| SMC_NAME_LOCK_RESULT_SUMMARY | 86 |
| SMC_NAME_LOCK_STATUS | 86 |
| SMC_NAME_LOCK_STATUS_CNT | 87 |
| SMC_NAME_LOCK_TYPE | 88 |
| SMC_NAME_LOCKS_BEING_BLOCKED_CNT | 88 |
| SMC_NAME_LOCKS_GRANTED_IMMED | 89 |
| SMC_NAME_LOCKS_GRANTED_WAITED | 90 |
| SMC_NAME_LOCKS_NOT_GRANTED | 90 |
| SMC_NAME_LOG_CONTENTION_PCT | 91 |
| SMC_NAME_LOGIN_NAME | 91 |
| SMC_NAME_MEM_CODE_SIZE | 92 |
| SMC_NAME_MEM_KERNEL_STRUCT_SIZE | 92 |
| SMC_NAME_MEM_PAGE_CACHE_SIZE | 93 |
| SMC_NAME_MEM_PROC_BUFFER | 93 |
| SMC_NAME_MEM_PROC_HEADER | 93 |

| | |
|---|-----|
| SMC_NAME_MEM_SERVER_STRUCT_SIZE | 94 |
| SMC_NAME_MOST_ACT_DEV_IO | 94 |
| SMC_NAME_MOST_ACT_DEV_NAME..... | 95 |
| SMC_NAME_NET_BYTE_IO..... | 95 |
| SMC_NAME_NET_BYTES_RCVD..... | 96 |
| SMC_NAME_NET_BYTES_SENT | 96 |
| SMC_NAME_NET_DEFAULT_PKT_SIZE | 96 |
| SMC_NAME_NET_MAX_PKT_SIZE | 97 |
| SMC_NAME_NET_PKT_SIZE_RCVD..... | 97 |
| SMC_NAME_NET_PKT_SIZE_SENT | 97 |
| SMC_NAME_NET_PKTS_RCVD | 98 |
| SMC_NAME_NET_PKTS_SENT..... | 98 |
| SMC_NAME_NUM_ENGINES..... | 99 |
| SMC_NAME_NUM_PROCESSES | 99 |
| SMC_NAME_OBJ_ID | 100 |
| SMC_NAME_OBJ_NAME..... | 101 |
| SMC_NAME_OBJ_TYPE..... | 101 |
| SMC_NAME_OWNER_NAME | 102 |
| SMC_NAME_PAGE_HIT_PCT | 102 |
| SMC_NAME_PAGE_INDEX_LOGICAL_READ | 102 |
| SMC_NAME_PAGE_INDEX_PHYSICAL_READ | 103 |
| SMC_NAME_PAGE_IO | 104 |
| SMC_NAME_PAGE_LOGICAL_READ | 104 |
| SMC_NAME_PAGE_NUM..... | 105 |
| SMC_NAME_PAGE_PHYSICAL_READ | 105 |
| SMC_NAME_PAGE_WRITE | 106 |
| SMC_NAME_PROC_STATE | 106 |
| SMC_NAME_PROC_STATE_CNT..... | 108 |
| SMC_NAME_SPID..... | 108 |
| SMC_NAME_SQL_SERVER_NAME..... | 110 |
| SMC_NAME_SQL_SERVER_VERSION..... | 110 |
| SMC_NAME_STP_CPU_TIME..... | 110 |
| SMC_NAME_STP_ELAPSED_TIME..... | 111 |
| SMC_NAME_STP_EXECUTION_CLASS | 111 |
| SMC_NAME_STP_HIT_PCT | 112 |
| SMC_NAME_STP_LINE_NUM..... | 112 |
| SMC_NAME_STP_LINE_TEXT | 113 |
| SMC_NAME_STP_LOGICAL_READ | 113 |
| SMC_NAME_STP_NUM_TIMES_EXECUTED | 113 |
| SMC_NAME_STP_PHYSICAL_READ | 114 |
| SMC_NAME_STP_STMT_NUM | 114 |
| SMC_NAME_THREAD_EXCEEDED_MAX..... | 115 |
| SMC_NAME_THREAD_EXCEEDED_MAX_PCT | 115 |
| SMC_NAME_THREAD_MAX_USED | 116 |

| | |
|---|-----|
| SMC_NAME_TIME_WAITED_ON_LOCK | 116 |
| SMC_NAME_TIMESTAMP | 116 |
| SMC_NAME_TIMESTAMP_DATIM | 117 |
| SMC_NAME_XACT | 117 |
| SMC_NAME_XACT_DELETE | 118 |
| SMC_NAME_XACT_DELETE_DEFERRED | 118 |
| SMC_NAME_XACT_DELETE_DIRECT | 119 |
| SMC_NAME_XACT_INSERT | 119 |
| SMC_NAME_XACT_INSERT_CLUSTERED | 119 |
| SMC_NAME_XACT_INSERT_HEAP | 120 |
| SMC_NAME_XACT_SELECT | 120 |
| SMC_NAME_XACT_UPDATE | 121 |
| SMC_NAME_XACT_UPDATE_DEFERRED | 121 |
| SMC_NAME_XACT_UPDATE_DIRECT | 121 |
| SMC_NAME_XACT_UPDATE_EXPENSIVE | 122 |
| SMC_NAME_XACT_UPDATE_IN_PLACE | 122 |
| SMC_NAME_XACT_UPDATE_NOT_IN_PLACE | 123 |

CHAPTER 3

Monitor Client Library Functions..... 125

| | |
|------------------------------------|-----|
| Threads | 126 |
| Error handling | 127 |
| Error handler | 127 |
| Callback function | 128 |
| smc_close | 130 |
| smc_connect_alloc | 131 |
| smc_connect_drop | 132 |
| smc_connect_ex | 134 |
| smc_connect_props | 135 |
| smc_create_alarm_ex | 140 |
| smc_create_filter | 144 |
| smc_create_playback_session | 148 |
| smc_create_recording_session | 154 |
| smc_create_view | 156 |
| smc_drop_alarm | 159 |
| smc_drop_filter | 160 |
| smc_drop_view | 161 |
| smc_get_command_info | 162 |
| smc_get_dataitem_type | 165 |
| smc_get_dataitem_value | 167 |
| smc_get_row_count | 168 |
| smc_get_version_string | 170 |
| smc_initiate_playback | 170 |
| smc_initiate_recording | 172 |
| smc_refresh_ex | 173 |

| | | |
|-------------------|--|------------|
| | smc_terminate_playback | 174 |
| | smc_terminate_recording..... | 175 |
| CHAPTER 4 | Building a Monitor Client Library Application | 179 |
| | Building on UNIX platforms | 180 |
| | Compiling the application | 180 |
| | Linking the application..... | 180 |
| | Running the application..... | 181 |
| | Building the sample applications | 181 |
| | Building on Windows platforms | 182 |
| | Compiling the application | 182 |
| | Linking the application..... | 183 |
| | Running the application..... | 183 |
| | Building the sample applications | 184 |
| CHAPTER 5 | Monitor Client Library Configuration Instructions | 187 |
| | Loading Monitor Client Library | 187 |
| | Using Studio Installer | 187 |
| | Results of the load | 188 |
| | Confirming your login account and permissions | 188 |
| | Modifying the interfaces file..... | 188 |
| | Setting up the user environment | 190 |
| | Setting the SYBASE environment variable | 190 |
| | Overriding the default location of the interfaces file | 190 |
| | Using Monitor Client Library | 191 |
| APPENDIX A | Examples of Views | 193 |
| | Cache performance summary..... | 194 |
| | Current statement summary..... | 195 |
| | Database object lock status | 195 |
| | Database object page I/O | 196 |
| | Data cache activity for individual caches | 197 |
| | Data cache statistics for session | 197 |
| | Data cache statistics for sample interval..... | 198 |
| | Device I/O for session | 198 |
| | Device I/O for sample interval..... | 199 |
| | Device I/O performance summary | 199 |
| | Engine activity | 200 |
| | Lock performance summary..... | 200 |
| | Network activity for session..... | 201 |
| | Network activity for sample interval..... | 202 |
| | Network performance summary | 202 |

| | |
|--|-----|
| Procedure cache statistics for session | 203 |
| Procedure cache statistics for sample interval | 203 |
| Procedure page I/O | 204 |
| Process activity | 205 |
| Process database object page I/O | 205 |
| Process detail for locks | 206 |
| Process detail page I/O | 207 |
| Process locks | 208 |
| Process page I/O | 208 |
| Process state summary | 209 |
| Process stored procedure page I/O | 209 |
| Server performance summary | 210 |
| Stored procedure activity | 211 |
| Transaction activity | 211 |

APPENDIX B

| | |
|---------------------------------------|------------|
| Datatypes and Structures | 227 |
| Summary of datatypes | 227 |
| Enum: SMC_ALARM_ACTION_TYPE | 230 |
| Enum: SMC_CLOSE_TYPE | 230 |
| Enum: SMC_DATAITEM_NAME | 230 |
| Enum: SMC_DATAITEM_STATTYPE | 230 |
| Structure: SMC_DATAITEM_STRUCT | 231 |
| Enum: SMC_DATAITEM_TYPE | 231 |
| Enum: SMC_ERR_SEVERITY | 231 |
| Enum: SMC_FILTER_TYPE | 232 |
| Enum: SMC_HS_ESTIM_OPT | 232 |
| Enum: SMC_HS_MISSDATA_OPT | 232 |
| Enum: SMC_HS_PLAYBACK_OPT | 232 |
| Enum: SMC_HS_SESS_DELETE_OPT | 233 |
| Enum: SMC_HS_SESS_ERR_OPT | 233 |
| Enum: SMC_HS_SESS_PROT_LEVEL | 233 |
| Enum: SMC_HS_SESS_SCRIPT_OPT | 234 |
| Enum: SMC_HS_TARGET_OPT | 234 |
| Enum: SMC_HS_TARGET_OPT | 234 |
| Enum: SMC_INFO_TYPE | 234 |
| Enum: SMC_LOCK_RESULT | 235 |
| Enum: SMC_LOCK_RESULT_SUMMARY | 235 |
| Enum: SMC_LOCK_STATUS | 236 |
| Enum: SMC_LOCK_TYPE | 236 |
| Enum: SMC_OBJ_TYPE | 236 |
| Enum: SMC_PROC_STATE | 236 |
| Enum: SMC_PROP_ACTION | 237 |
| Enum: SMC_PROP_TYPE | 237 |
| Enum: SMC_RETURN_CODE | 238 |

| | | |
|------------|---|------------|
| | Enum: SMC_SERVER_MODE | 239 |
| | Enum: SMC_SOURCE | 239 |
| | Union: SMC_VALUE_UNION | 240 |
| APPENDIX C | Backward Compatibility..... | 241 |
| | Obsolete and replacement functions..... | 241 |
| | New functions..... | 242 |
| | Rules for functions and callbacks compatibility..... | 242 |
| APPENDIX D | Troubleshooting Information and Error Messages | 243 |
| | Troubleshooting | 243 |
| | Error messages..... | 244 |
| | Communication failure: check if server is running..... | 244 |
| | Configuration failure: possibly missing interfaces file or bad login parameters. | 244 |
| | Don't know how to build example.h..... | 245 |
| | error L2029: 'SMC_CONNECT' : unresolved external..... | 245 |
| | error L2029: 'SMC_CREATE_VIEW' : unresolved external.. | 245 |
| | fatal error C1083: Cannot open include file: 'cstypes.h': No such file or directory | 245 |
| | fatal error C1083: Cannot open include file: 'mcpublic.h': No such file or directory | 246 |
| | LINK: fatal error L4051: smcapi32.lib : cannot find library..... | 246 |
| | Index | 247 |

About This Book

Sybase® Adaptive Server™ Enterprise Monitor™ Client Library Programmer's Guide describes how to write Sybase Adaptive Server Enterprise Monitor Client Library (Monitor Client Library) applications that access Sybase Adaptive Server Enterprise performance data.

Audience

This guide is for programmers who use Adaptive Server Enterprise Monitor Server or Adaptive Server Enterprise Monitor Historical Server.

How to use this book

When writing a Monitor Client Library application, use this book as a source of general information on how to construct Monitor Client Library programs.

- Chapter 1, “Getting started with Monitor Client Library” explains how to structure a basic Monitor Client Library program and includes a simple, complete Monitor Client Library application.
- Chapter 2, “Data Items and Statistical Types” describes data items, statistical types, and valid data item combinations of data items used in Monitor Client Library applications to gather performance data.
- Chapter 3, “Monitor Client Library Functions” describes each function including syntax, parameter values, examples, permissions, and related functions.
- Chapter 4, “Building a Monitor Client Library Application” describes how to compile and link a Monitor Client Library program.
- Chapter 5, “Monitor Client Library Configuration Instructions” explains how to configure Monitor Client Library on UNIX or Windows NT.
- Appendix A, “Examples of Views” provides examples of valid views.
- Appendix B, “Datatypes and Structures” summarizes datatypes used by Monitor Client Library and describes the datatypes that have no equivalent in C or Open-Client Client Library.
- Appendix C, “Backward Compatibility” lists obsolete functions and their replacement functions.

-
- Appendix D, “Troubleshooting Information and Error Messages” explains how to respond to problems that you might have with Monitor Client Library and lists error messages that may be reported.

Related documents

The following documents comprise the Sybase Adaptive Server Enterprise documentation:

- The release bulletin for your platform – contains last-minute information that was too late to be included in the books.

A more recent version of the release bulletin may be available on the World Wide Web. To check for critical product or document information that was added after the release of the product CD, use the Sybase Technical Library.

- The *Installation Guide* for your platform – describes installation, upgrade, and configuration procedures for all Adaptive Server and related Sybase products.
- *Configuring Adaptive Server Enterprise* for your platform – provides instructions for performing specific configuration tasks for Adaptive Server.
- *What’s New in Adaptive Server Enterprise?* – describes the new features in Adaptive Server version 12.5, the system changes added to support those features, and the changes that may affect your existing applications.
- *Transact-SQL User’s Guide* – documents Transact-SQL, Sybase’s enhanced version of the relational database language. This manual serves as a textbook for beginning users of the database management system. This manual also contains descriptions of the pubs2 and pubs3 sample databases.
- *System Administration Guide* – provides in-depth information about administering servers and databases. This manual includes instructions and guidelines for managing physical resources, security, user and system databases, and specifying character conversion, international language, and sort order settings.
- *Reference Manual* – contains detailed information about all Transact-SQL commands, functions, procedures, and datatypes. This manual also contains a list of the Transact-SQL reserved words and definitions of system tables.

- *Performance and Tuning Guide* – explains how to tune Adaptive Server for maximum performance. This manual includes information about database design issues that affect performance, query optimization, how to tune Adaptive Server for very large databases, disk and cache issues, and the effects of locking and cursors on performance.
- The *Utility Guide* – documents the Adaptive Server utility programs, such as `isql` and `bcpl`, which are executed at the operating system level.
- The *Quick Reference Guide* – provides a comprehensive listing of the names and syntax for commands, functions, system procedures, extended system procedures, datatypes, and utilities in a pocket-sized book. Available only in print version.
- The *System Tables Diagram* – illustrates system tables and their entity relationships in a poster format. Available only in print version.
- *Error Messages and Troubleshooting Guide* – explains how to resolve frequently occurring error messages and describes solutions to system problems frequently encountered by users.
- *Component Integration Services User's Guide* – explains how to use the Adaptive Server Component Integration Services feature to connect remote Sybase and non-Sybase databases.
- *Java in Adaptive Server Enterprise* – describes how to install and use Java classes as datatypes, functions, and stored procedures in the Adaptive Server database.
- *Using Sybase Failover in a High Availability System* – provides instructions for using Sybase's Failover to configure an Adaptive Server as a companion server in a high availability system.
- *Using Adaptive Server Distributed Transaction Management Features* – explains how to configure, use, and troubleshoot Adaptive Server DTM features in distributed transaction processing environments.
- *EJB Server User's Guide* – explains how to use EJB Server to deploy and execute Enterprise JavaBeans in Adaptive Server.
- *XA Interface Integration Guide for CICS, Encina, and TUXEDO* – provides instructions for using Sybase's DTM XA interface with X/Open XA transaction managers.
- *Glossary* – defines technical terms used in the Adaptive Server documentation.

-
- *Sybase jConnect for JDBC Programmer's Reference* – describes the jConnect for JDBC product and explains how to use it to access data stored in relational database management systems.
 - *Full-Text Search Specialty Data Store User's Guide* – describes how to use the Full-Text Search feature with Verity to search Adaptive Server Enterprise data.
 - *Historical Server User's Guide* – describes how to use Historical Server to obtain performance information for SQL Server and Adaptive Server.
 - *Monitor Server User's Guide* – describes how to use Monitor Server to obtain performance statistics from SQL Server and Adaptive Server.
 - *Monitor Client Library Programmer's Guide* – describes how to write Monitor Client Library applications that access Adaptive Server performance data.

Other sources of information

Use the following books for more information about Adaptive Server Enterprise configuration and tuning:

- *Managing and Monitoring Sybase Adaptive Server Enterprise* describes how to manage Adaptive Server Enterprise and monitor its activities using Sybase Central™.
- *Adaptive Server Enterprise Performance and Tuning Guide* describes how to analyze Adaptive Server Enterprise performance and tune it to improve performance.
- *Adaptive Server Enterprise Reference Manual* describes SQL commands, functions, and stored procedures used with Sybase Adaptive Server Enterprise.
- *Adaptive Server Enterprise System Administration Guide* contains information about administering Adaptive Server Enterprise.

Use the Sybase Technical Library CD and the Technical Library Product Manuals Web site to learn more about your product:

- Technical Library CD contains product manuals and is included with your software. The DynaText browser (downloadable from Product Manuals at <http://www.sybase.com/detail/1,3693,1010661,00.html>) allows you to access technical information about your product in an easy-to-use format.

Refer to the *Technical Library Installation Guide* in your documentation package for instructions on installing and starting the Technical Library.

- Technical Library Product Manuals Web site is an HTML version of the Technical Library CD that you can access using a standard Web browser. In addition to product manuals, you will find links to the Technical Documents Web site (formerly known as Tech Info Library), the Solved Cases page, and Sybase/Powersoft newsgroups.

To access the Technical Library Product Manuals Web site, go to Product Manuals at <http://www.sybase.com/support/manuals/>.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ For the latest information on product certifications

- 1 Point your Web browser to Technical Documents at <http://www.sybase.com/support/techdocs/>.
- 2 Select Products from the navigation bar on the left.
- 3 Select a product name from the product list.
- 4 Select the Certification Report filter, specify a time frame, and click Go.
- 5 Click a Certification Report title to display the report.

❖ For the latest information on EBFs and Updates

- 1 Point your Web browser to Technical Documents at <http://www.sybase.com/support/techdocs/>.
- 2 Select EBFs/Updates. Enter user name and password information, if prompted (for existing Web accounts) or create a new account (a free service).
- 3 Specify a time frame and click Go.
- 4 Select a product.
- 5 Click an EBF/Update title to display the report.

❖ To create a personalized view of the Sybase Web site (including support pages)

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

- 1 Point your Web browser to Technical Documents at <http://www.sybase.com/support/techdocs/>
- 2 Click MySybase and create a MySybase profile.

Text conventions

Italics indicates emphasis or indicates that a definition accompanies the highlighted term.

Variable names, document titles, file names, directory names, and Adaptive Server Enterprise table names are in italic typeface:

-Umyname
Adaptive Server Enterprise Monitor Server User's Guide
SQLMon.mrg file
\$SYBASE directory
systables

Command syntax

Command syntax statements use the following notational conventions:

Table 1: Command syntax conventions

| Example | Description |
|-------------------|---|
| monserver | Command keywords appear in lowercase. |
| <i>option</i> | Variables (words that stand for values that you supply in the command) appear in italic. |
| [<i>option</i>] | Brackets mean that including the enclosed items in the command is <i>optional</i> . Do not include the brackets in your command. |
| <i>option</i> ... | Ellipses indicate that you may repeat the preceding item as many times as you like in the command. Do not include ellipses in your command. |

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

Getting started with Monitor Client Library

This chapter describes the following topics:

| Topic | Page |
|--|------|
| Overview | 1 |
| What is Adaptive Server Enterprise Monitor | 1 |
| Writing a Basic Monitor Client Library program | 4 |
| A sample Monitor Client Library program | 14 |

Overview

Monitor Client Library is part of Adaptive Server Enterprise Monitor™. It is an application programming interface (API) that enables you to write client applications that connect to Adaptive Server, Adaptive Server Enterprise Monitor Server (Monitor Server), and Adaptive Server Enterprise Historical Server (Historical Server) to gather performance data. This chapter describes Adaptive Server Enterprise Monitor, explains the components of a Monitor Client Library application, and lists a sample Monitor Client Library application.

What is Adaptive Server Enterprise Monitor

Adaptive Server Enterprise Monitor provides a way to monitor Adaptive Server performance in real time or in a historical data-gathering mode. System administrators can use this information to identify potential resource bottlenecks, to research current problems, and to tune for better performance. Adaptive Server Enterprise Monitor provides feedback for tuning at several levels:

- Adaptive Server configuration

- Table and index design
- SQL statements in applications and stored procedures

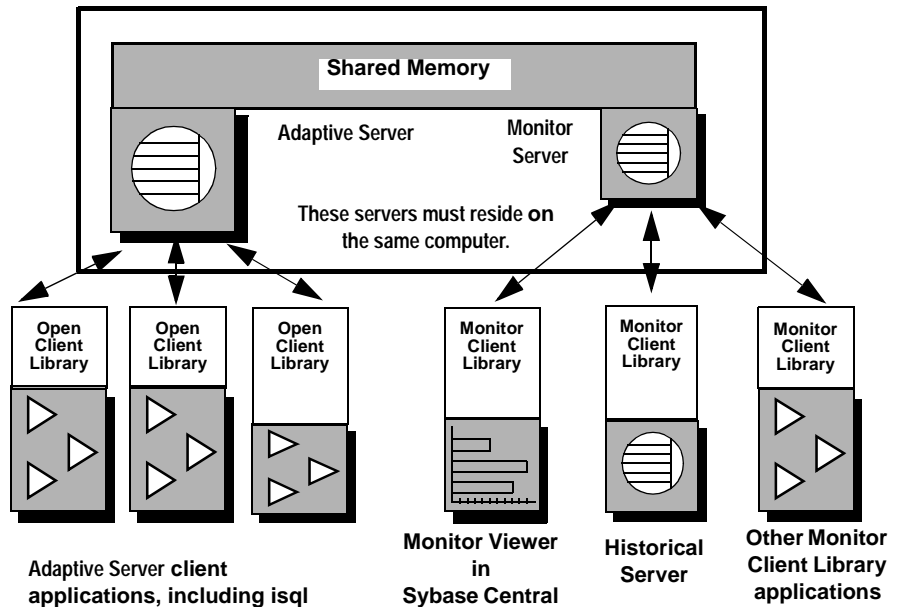
Adaptive Server Enterprise Monitor components

Adaptive Server Enterprise Monitor consists of four components that gather or display Adaptive Server performance data:

- Monitor Server – a server that collects Adaptive Server performance data in real time and makes the data available to the other Adaptive Server Enterprise Monitor components. Monitor Server is a Sybase Open Server™ application.
- Historical Server – a server that obtains Adaptive Server performance data from Monitor Server and saves the data in files for deferred analysis. Historical Server is a Sybase Open Server application.
- Monitors in the Adaptive Server plug-in for Sybase Central (Monitor Viewer) – the monitors provide a graphical user interface to Monitor Server. They obtain Adaptive Server performance data from Monitor Server and display the data in real time in tables and graphs.
- Monitor Client Library – an application programming interface to Monitor Server available to users for developing monitoring applications. Monitor Viewer and Historical Server are Monitor Client Library applications.

Adaptive Server Enterprise Monitor architecture

Figure 1-1 shows the relationships between Adaptive Server and the various components of Adaptive Server Enterprise Monitor.

Figure 1-1: Adaptive Server Enterprise Monitor architecture

Adaptive Server saves performance data in a shared *memory area* that Monitor Server reads. Because of this shared memory technique, Monitor Server must be installed and running on the same machine as the Adaptive Server installation being monitored. A one-to-one relationship exists between Adaptive Server and Monitor Server. For more information about Monitor Server, see the *Sybase Adaptive Server Enterprise Monitor Server User's Guide*.

Monitor Client Library applications obtain Adaptive Server performance statistics from Monitor Server. These applications are clients of Monitor Server. For performance reasons, Sybase recommends running Monitor Client Library applications on machines other than the ones where Adaptive Server/Monitor Server pairs are running.

Monitor Viewer in Sybase Central includes a set of monitors showing different aspects of Adaptive Server resource usage at various levels of detail. Each open monitor is a separate application, with a unique client connection to Monitor Server. In Sybase Central, each Adaptive Server installation has its own Monitors folder containing the set of monitor objects.

Historical Server collects performance information from Monitor Server and saves the information in files for deferred analysis. Historical Server interfaces let users specify the data to collect and the time period desired. They also include a historical data playback feature. The interfaces are:

- A command interface in isql. For more information, see the *Sybase Adaptive Server Enterprise Monitor Historical Server User's Guide*.
- A programming interface using Monitor Client Library. For more information, see Chapter 3, "Monitor Client Library Functions" and the *Sybase Adaptive Server Enterprise Monitor Historical Server User's Guide*.

Writing a Basic Monitor Client Library program

A basic Monitor Client Library application:

- 1 Defines error handling.
- 2 Connects to a server using the following steps:
 - Allocates a connection.
 - Sets properties on a connection.
 - Connects to a server.
- 3 Creates one or more views that define the performance data to be monitored.
- 4 Optionally, targets specific performance data values with filters.
- 5 Optionally, sets alarms on performance data values.
- 6 Requests performance data values.
- 7 Processes the results.
- 8 Closes the connection to the server.
- 9 Deallocates the connection or reuse it by reconnecting.

Note You must have the System Administrator role on Adaptive Server or execute permission on the stored procedure `mon_rpc_connect` to perform monitoring.

Application logic flow

Most Monitor Client Library applications exhibit a logic flow similar to the following:

```
allocate a connection
set properties on the connection
connect
loop to create views on the connection
loop to create filters (optional)
loop to create alarms (optional)
loop to refresh connection
  for each view
    get the row count
    for each row
      for each column
        get the data
        display the data
loop to drop alarms (optional)
loop to drop filters (optional)
loop to drop views (optional)
close monitor connection
deallocate or reuse connection
```

where:

- An application can have any number of connections.
- A connection can have one or more views.
- A view must have one or more data items.
- A view can have one filter per data item.
- A view can have any number of alarms and can have multiple alarms per data item in the view.

The following sections describe the steps for a basic Monitor Client Library program. The steps are cross referenced to the sample program that follows them.

Step 1: define error handling

An application uses one or more callback routines to handle Monitor Client Library and Server error and informational messages.

See the code sample on page 34.

Step 2: connect to a server

The Monitor Client Library functions require establishing an Adaptive Server Enterprise Monitor connection. The Adaptive Server Enterprise Monitor connection uses one or more Open Client connections depending upon the connection type.

The two types of Monitor connections are *Live* mode and *Historical* mode:

- Live mode connects to Monitor Server and Adaptive Server. It provides access to performance data.
- Historical mode connects to Historical Server and either records performance data for later access or plays back recorded data.

Connecting to a server is a three-step process. An application:

- Allocates a connection structure
- Sets properties for the connection, if necessary
- Logs in to a server

Allocating a connection structure

An application calls `smc_connect_alloc` to allocate a connection structure.

See the code sample on page 16.

Setting connection structure properties

An application calls `smc_connect_props` to set, retrieve, or clear connection structure properties.

Connection properties define various aspects of a connection's behavior. For example:

- `SMC_PROP_USERNAME` defines the *username* that a connection will use when logging into a server.
- `SMC_PROP_PASSWORD` specifies the *password* for the *username*.
- `SMC_PROP_SERVERNAME` defines the server for this connection.
- `SMC_PROP_IFILE` defines the *interfaces* file name for this connection. If you do not specify this property on a UNIX system, the default *interfaces* file in the *SYBASE* environment variable directory is used. On Windows NT, the default *interfaces* file is *sql.ini*.

- `SMC_PROP_SERVERMODE` defines the type of connection: live or historical.

Required connection properties

At a minimum, an application must set the connection properties that specify the connection's *username* (`SMC_PROP_USERNAME`) and allow the server to authenticate the user's identity by requiring a valid password. If the server requires a password, then the application must set the `SMC_PROP_PASSWORD` property to the value of the user's server password.

See the code sample on page 17.

Connecting to a server

An application calls `smc_connect_ex` to connect to a server. When establishing a connection, `smc_connect_ex` sets up communication with the network, logs in to the server, and communicates any connection-specific property information to the server. A connection to Adaptive Server writes dbcc traceon messages to the Adaptive Server error log. You can ignore these messages.

For example, if the server supports network-based user authentication and the client application requests it, then Client Library and the server query the network's security system to see if the user (whose name is specified by `SMC_PROP_USERNAME`) is logged in to the network.

See the code sample on page 22.

Step 3: create a view

Views are defined groups of data items. The data items specified determine how the data is summarized. Since you can specify multiple views, the application has full flexibility in the gathering of data. For example, a view consisting of two data items (device name, value for sample and device I/O, rate for sample) returns the device I/O rate for each database device.

For details on valid combinations of data items and information about how data items are summarized, see Chapter 2, "Data Items and Statistical Types"

For examples of views, see Appendix A, "Examples of Views".

Data items

A data item is a particular piece of data that can be obtained from the Monitor Client Library, for example, page I/O, login name, device reads, and so on. For each data item in a view, you must specify a statistical type.

Statistical types

The *statistic type* defines the duration of the data item (sample or session) and whether the server performs calculations on the data item.

The six statistic types are:

- **SMC_STAT_VALUE_SAMPLE** – this statistic type returns a count of activity or some type of information that applies to the most recent sample interval. No calculations are performed.
 - Activity counts – for data items that represent activity counts, **SMC_STAT_VALUE_SAMPLE** returns the number of occurrences of an activity during the most recent sample interval. For example, **SMC_STAT_VALUE_SAMPLE** for **SMC_NAME_PAGE_IO** is the number of page I/Os that occurred during the most recent sample interval.
 - Other information – this is the only statistic type valid for data items that represent character strings. For example, **SMC_STAT_VALUE_SAMPLE** for **SMC_NAME_OBJECT_NAME** returns the name of a database object. This statistic type is also the only one valid for data items that represent values such as IDs and values for configured parameters, on which calculations are never performed.
- **SMC_STAT_VALUE_SESSION** – this statistic type returns a cumulative count of activity since the start of gathering the data (since the connection was opened). No calculations are performed. For example, **SMC_STAT_VALUE_SESSION** for **SMC_NAME_PAGE_IO** is the number of page I/Os that occurred since the session started.
- **SMC_STAT_RATE_SAMPLE** – this statistic type calculates a rate per second. It returns the average number of occurrences per second of an activity during the most recent sample interval. For example, **SMC_STAT_RATE_SAMPLE** for **SMC_NAME_PAGE_IO** is the average number of page I/Os that occurred each second during the most recent sample interval.

The calculation is *count for the most recent sample interval* divided by *number of seconds in the sample interval*.

- **SMC_STAT_RATE_SESSION** – this statistic type calculates a rate per second. It returns the average number of occurrences per second of an activity during the current session. For example, **SMC_STAT_RATE_SESSION** for **SMC_NAME_PAGE_IO** is the average number of page I/Os that occurred per second since the session started.

The calculation is *count for the session* divided by *number of seconds in the session*.

- **SMC_STAT_AVG_SAMPLE** – this statistic type calculates an average value per occurrence of an activity over the most recent sample interval. Only a few data items can use this statistic type. The meaning of the returned value depends on the data item name. For example, **SMC_STAT_AVG_SAMPLE** for **SMC_NAME_STP_ELAPSED_TIME** is the average execution time per execution of a stored procedure during the most recent sample interval.
- **SMC_STAT_AVG_SESSION** – this statistic type calculates an average value per occurrence of an activity over the session. Only a few data items can use this statistic type. The meaning of the returned value depends on the data item name. For example, **SMC_STAT_AVG_SESSION** for **SMC_NAME_STP_ELAPSED_TIME** is the average execution time per execution of a stored procedure during the recording session.

Note Not all statistical types are valid for all data items. See Chapter 2, “Data Items and Statistical Types” for more information about data items and the rules for using them.

Creating views for a connection

smc_create_view creates a view on a particular Monitor connection. A connection must have at least one view.

For details on valid combinations of data items and information about how data items are summarized, see Chapter 2, “Data Items and Statistical Types”.

You can think of a view as a table. The data items in a view are represented by the columns in that table. The number of rows returned for a particular view depend upon the particular data items in the view. For example, a view with server-wide data returns a single row, whereas a view with per-device data returns one row for each device.

For example:

A view consisting of two data items:

```
SMC_NAME_LOCK_TYPE, SMC_STAT_VALUE_SAMPLE  
SMC_NAME_LOCK_COUNT, SMC_STAT_RATE_SAMPLE
```

returns the rate of requested locks for each lock type during the sample interval.

A view consisting of one data item:

```
SMC_NAME_LOCK_COUNT, SMC_STAT_RATE_SAMPLE
```

returns the rate of requested locks summarized for all lock types during the sample interval.

For complete details on valid combinations of data items and understanding of how data items are summarized, refer to Chapter 2, “Data Items and Statistical Types”

See the code sample on page 22.

Step 4: create filters

`smc_create_filter` creates a filter on a data item. Filters limit the number of rows of performance data returned by a view. A filter can be applied to any data item specified in a view. A view can contain one filter per data item. If you include more than one filter in a view, Monitor Client Library uses ANDs to include those filters.

The types of filters available are:

- Equal to
 - returns only values equal to one of the specified values (logical OR of each Equal comparison)
- Not Equal to
 - returns only values equal to none of the specified value (logical AND of each Not-Equal comparison)
- Greater than or equal to

- returns values greater than or equal to the specified value
- Less than or equal to
 - returns values less than or equal to the specified value
- Range: bottom is less than or equal to value which is less than or equal to top
 - returns values between the top and bottom values, inclusive
- Top N
 - returns the N highest values

A view may contain more than one filter, but any particular data item may only have one filter bound to it. When a view contains more than one filter, the filters are combined with an AND.

You can add or drop filters at any time. The change in filtering takes effect as of the next refresh.

See the code sample on page 24.

Step 5: set alarms

`smc_create_alarm_ex` sets an alarm on any numeric data item (except for IDs) in a view. When specifying an alarm for a particular data item in a live connection, an application supplies a callback function that is invoked when the alarm is triggered.

The Historical Server cannot call a callback function, but it can write to a log file or execute a procedure each time an alarm is triggered.

An example of the type of actions an application can execute upon the triggering of an alarm is to log a message, which is one of the features provided by Historical Server.

You can add or drop an alarm at any time. The change in alarm specification takes effect as of the next refresh.

Note Monitor Client Library applies alarms after it applies filters.

See the code sample on page 25.

Step 6: request performance data and process results

After all of the connections, views, alarms, and filters are created, an application requests values for performance data. Retrieving performance data is a three-step process:

- Refresh the data.
- Check the row count.
- Look at each data item in the view.

When a Monitor Client Library application needs to retrieve data, it initiates a refresh. The refresh causes Monitor Client Library to obtain fresh data. After each refresh, the application retrieves the data in each view on an item-by-item basis (that is, for each column of a table).

After calling `smc_refresh_ex` on a given connection, the application retrieves the data. The user retrieves data on an item-by-item basis.

Depending on the number of events being collected, frequent refreshes might be necessary. A view that contains many keys needs more frequent refreshes than views with one or a few keys. The following symptoms might indicate an application that is not refreshing frequently enough:

- Very large numbers of lost events reported in the Monitor Server error log. The *Sybase Adaptive Server Enterprise Monitor Server User's Guide* discusses configuration changes that can also help to reduce event loss.
- The application appears to hang in a call to `smc_refresh_ex`. A large number of keys in a view can cause a condition in which Monitor Server can not keep up with the number of events being collected and does not return control. Because of this, Monitor Server begins to consume large amounts of CPU time.

`smc_get_row_count` determines how many rows of results are available for a view. A view returns results in what is essentially a table with potentially many "rows" of result data, but in some cases, possibly zero rows.

`smc_get_dataitem_value` retrieves performance data values for a single column of a single row of a view.

Filters and alarms are applied during the refresh of the data.

Polling for new performance data is client-driven and is limited only by the speed of the data-providing system and the data-gathering system.

See the code sample on page 26.

Step 7: close and deallocate connections

Before exiting, a Monitor Client Library application must:

- Close all open connections.
- Deallocate each connection.

Closing and deallocating connections

An application calls `smc_close` to close a connection and `smc_connect_drop` to deallocate a connection structure. It is an error to deallocate a connection that has not been closed. A call to `smc_close` results in the following implicit Monitor Client Library calls:

- One or more calls to `smc_drop_alarm` to remove alarms, if necessary.
- One or more calls to `smc_drop_filter` to remove filters, if necessary.
- One or more calls to `smc_drop_view` to remove views.

See the code sample on page 33.

Reopening connections

After an application closes a connection, but before it deallocates the connection structure, it can call `smc_connect_ex` to reopen the connection.

Playing back recorded data

To retrieve recorded data from Historical Server, the steps are similar to the above, except:

- The application must connect to Historical Server. Set `smc_prop_servermode` to `SMC_SERVER_M_HISTORICAL` before making the connection.
- The application must call `smc_create_playback_session` after connecting, but before creating views.
- The application must call `smc_initiate_playback` after creating all views.
- Alarms are not allowed on playback of recorded historical data.
- Views and filters cannot be dropped.
- After the last refresh, the application must call `smc_terminate_playback`.

A sample Monitor Client Library program

This section contains a listing for a sample Monitor Client Library program that connects to a server, sends a query, processes the results, then exits.

Example program

The following example program, *monitor.c*, demonstrates the steps outlined in the previous section. Commentary for each step follows the example.

```
/*monitor.c
** Example program showing logic flow of Monitor Client Library
** application. This example assumes the use of an ANSI C
** compliant compiler. This program creates two connections
** to the Monitor Server. Data is extracted from one connection
** at the beginning and end of the monitoring session.
** Data is extracted from the other connection every
** SAMPLE_INTERVAL seconds NUM_OF_SAMPLES times.
**/
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
/* The mcpublic.h header file contains function prototypes, etc.
** for monitor client library functions. It also includes a
** header file called mctypes.h, which defines the datatypes
** used for monitor client library applications.
**/
#include "mcpublic.h"
#define NUM_OF_SAMPLES 10
#define SAMPLE_INTERVAL 5
#define NUM_SERVER_DATA_ITEMS 3
#define NUM_DB_INFO_ITEMS 14
#define NUM_NW_INFO_ITEMS 6
#define OPTIONAL_CALLS -1

/*Error signals*/
#define VIEW_NONEXISTENT -1
#define CONNECT_NONEXISTENT -1

SMC_RETURN_CODE main (SMC_INT argc, SMC_CHARP argv[])
{
    SMC_VALUE_UNION serverNameUnion;
    SMC_VALUE_UNION userNameUnion;
    SMC_VALUE_UNION passwordUnion;
    SMC_VALUE_UNION interfacesFileUnion;
```

```

SMC_VALUE_UNION workUnion;
SMC_VALUE_UNION returnedDataUnion;
SMC_CONNECT_ID connect1_id;
SMC_CONNECT_ID connect2_id;
SMC_VIEW_ID server_view_id;
SMC_VIEW_ID db_info_view_id;
SMC_VIEW_ID nw_info_view_id;

SMC_RETURN_CODE ret;
SMC_DATAITEM_TYPE dataitem_type; /*Holds data item type
                                   returned by get_dataitem_type
                                   function call*/
/*Needed if alarms and filters are used */
#ifdef OPTIONAL_CALLS
    SMC_ALARM_ID    alarm_id;
    SMC_FILTER_ID   filter_id;
    SMC_CHARP       filter_strings[2]; /*datatype is pointer to
                                       string. This is an array
                                       of pointers.*/
#endif
    SMC_SIZET    row,num_of_rows,item; /*This is an integer data
                                       type*/
    SMC_SIZET    outputLength;          /*Length of output returned
                                       by smc_connect_props
                                       function call*/
/*
** Definition of SMC_DATAITEM_STRUCT datatype
*/
    SMC_DATAITEM_STRUCT  server_info_view[NUM_SERVER_DATA_ITEMS];
    SMC_DATAITEM_STRUCT  db_info_view[NUM_DB_INFO_ITEMS];
    SMC_DATAITEM_STRUCT  nw_bytes_view[NUM_NW_INFO_ITEMS];

    SMC_VALUE_UNION      server_data[NUM_SERVER_DATA_ITEMS];
    SMC_VALUE_UNION      db_data[NUM_DB_INFO_ITEMS];
    SMC_VALUE_UNION      nw_data[NUM_NW_INFO_ITEMS];

/*Callback function prototypes. Actual functions are defined
** below.
*/
    SMC_VOID errorCallback(SMC_CONNECT_ID,SMC_COMMAND_ID,SMC_VOIDP);
    SMC_VOID alarmCallback(SMC_CONNECT_ID,SMC_COMMAND_ID,SMC_VOIDP);

    SMC_BOOL explicitInterfacesFile = FALSE;

    int index,iterations;

```

```
/*
** These are labels used when printing out data returned by the
** database info view.
*/
SMC_CHARP db_info_labels[NUM_DB_INFO_ITEMS] = {
    "Database ID: ",
    "Object ID: ",
    "Database name: ",
    "Object name: ",
    "Page hit percent: ",
    "Page I/O: ",
    "Page logical reads this sample: ",
    "Page logical reads this session: ",
    "Page logical read rate this sample: ",
    "Page logical read rate this session: ",
    "Page physical reads this sample: ",
    "Page physical reads this session: ",
    "Page physical read rate this sample: ",
    "Page physical read rate this session: "
};

/*
** These are labels used when printing out data returned by
** network info view.
*/
SMC_CHARP nw_info_labels[NUM_NW_INFO_ITEMS] = {
    "Network bytes received this sample: ",
    "Network bytes received this session: ",
    "Network bytes sent this sample: ",
    "Network bytes sent this session: ",
    "Network byte I/O rate this sample: ",
    "Network byte I/O rate this session: "
};

if (argc < 5){
    printf("Usage <%s> -U <user_name> [-P <password>]\n"
        "-S <monserver name> [-I <interfaces_file>]\n", argv[0]);
    exit(1);
}

/*
** Connect to a server.
*/
```

For commentary, see “Step 2: connect to a server” on page 6

```
/*
```

```

** Allocate first connection
*/
ret=smc_connect_alloc(errorCallback,
                      &connect1_id /*Pointer to connect_id!*/
                      );
if (ret != SMC_RET_SUCCESS) {
    printf("Attempt to allocate first connection failed \
          with error %d.\n",ret);
    exit(1);
}
/*
** Allocate second connection
*/
ret=smc_connect_alloc(errorCallback,
                      &connect2_id /*Pointer to connect_id!*/
                      );
if (ret != SMC_RET_SUCCESS) {
    printf("Attempt to allocate second connection failed \
          with error %d.\n",ret);
    exit(1);
}
/*

** Set mandatory and some optional connection properties.

** Mandatory connection properties are user name, server name,
** and password if user password is not NULL. If interfaces
** file name is not set, default is "interfaces" in directory
** pointed to by $SYBASE environment variable.

    For commentary, see "Required connection properties" on page 7.

*/

for (index=1;index<argc;index++) {
/*User name*/
    if (strcmp(argv[index],"-U",2) == 0) {
        userNameUnion.stringValue = argv[index+1];
        ret=smc_connect_props(connect1_id,
                               SMC_PROP_ACT_SET, /*Property action*/
                               SMC_PROP_USERNAME,/*Property*/
                               &userNameUnion, /*Note that union,
                                                not member of union,
                                                is used for
                                                property value*/
                               SMC_NULLTERM, /*Indicates null-
                                                terminated string
                                                for buffer length*/

```

```

                                NULL                /*Use NULL when
                                                    setting a property*/
                                );
                                /*End if argument is user name*/
    }
    if (ret != SMC_RET_SUCCESS) {
        printf("Could not set user name.\n");
        exit(SMC_RET_FAILURE);
    }
    /*Password. Default password is a null string*/
    if (strncmp(argv[index], "-P", 2) == 0) {
        passwordUnion.stringValue = argv[index+1];
        ret=smc_connect_props(connect1_id,
                                SMC_PROP_ACT_SET, /*Property action*/
                                SMC_PROP_PASSWORD, /*Property*/
                                &passwordUnion, /*Note that union,
                                                    not member of union,
                                                    is used for
                                                    property value*/
                                SMC_NULLTERM, /*Indicates null-
                                                    terminated string
                                                    for buffer length*/
                                NULL /*Use NULL when
                                                    setting a property*/
                                );
    }
    /*End if argument is password*/
    if (ret != SMC_RET_SUCCESS) {
        printf("Could not set password.\n");
        exit(SMC_RET_FAILURE);
    }
    /*Server name*/
    if (strncmp(argv[index], "-S", 2) == 0) {
        serverNameUnion.stringValue = argv[index+1];
        ret=smc_connect_props(connect1_id,
                                SMC_PROP_ACT_SET, /*Property action*/
                                SMC_PROP_SERVERNAME, /*Property*/
                                &serverNameUnion, /*Note that union,
                                                    not member of union,
                                                    is used for
                                                    property value*/
                                SMC_NULLTERM, /*Indicates null-
                                                    terminated string
                                                    for buffer length*/
                                NULL /*Use NULL when
                                                    setting a property*/
                                );
    }
    /*End if argument is server name*/
}
```

```

    if (ret != SMC_RET_SUCCESS) {
        printf("Could not set server name.\n");
        exit(SMC_RET_FAILURE);
    }
/*Interfaces file. If unspecified, $SYBASE/interfaces is used*/
    if (strcmp(argv[index],"-I",2) == 0) {
        interfacesFileUnion.stringValue = argv[index+1];
        ret=smc_connect_props(connect1_id,
                               SMC_PROP_ACT_SET, /*Property action*/
                               SMC_PROP_IFILE,    /*Property*/
                               &interfacesFileUnion, /*Note that
                                                         pointer to union,
                                                         not member of
                                                         union,is used for
                                                         property value*/
                               SMC_NULLTERM,      /*Indicates null-
                                                         terminated string
                                                         for buffer length*/
                               NULL                 /*Use NULL when
                                                         setting a property*/
                               );
        explicitInterfacesFile = TRUE;
    } /*End if argument is interfaces file pathname*/
    if (ret != SMC_RET_SUCCESS) {
        printf("Could not set interfaces file name.\n");
        printf("Using default interfaces file.\n");
    }
    /*End for loop getting connection properties
    from command-line arguments*/
/*
** Optional smc_get_connect_props call that sets a pointer to be
** passed to error callback. In this case, the pointer is to a
** string that tells which connection encountered the error.
**/
    workUnion.voidpValue = "first connection"; /*Call to set user
                                                data handle looks
                                                for value to set in
                                                void pointer member
                                                of union.*/
    ret=smc_connect_props(connect1_id,SMC_PROP_ACT_SET,\
                          SMC_PROP_USERDATA,&workUnion,SMC_NULLTERM,NULL);
    if (ret != SMC_RET_SUCCESS){
        printf("smc_connect_props call failed to \
              set userDataHandle.\n");
    }
/*

```

```
** Demonstration of "get" mode for smc_get_connect_props
*/
/*Check if user name has been set*/
ret=smc_connect_props(connect1_id,
                      SMC_PROP_ACT_GET,/*Property action is "get"*/
                      SMC_PROP_USERNAME,
                      &workUnion,
                      SMC_UNUSED,      /*Length parameter ignored
                                         on "get" operations*/
                      &outputLength    /*Note this is a pointer!*/
                      );
if (ret != SMC_RET_SUCCESS) {
    printf ("Could not get user name. Execution continuing.\n");
}
else {
    if (outputLength == 0) {
        printf("User name not set. Quitting execution.\n");
        exit(SMC_RET_FAILURE);
    }
    else {
/*
** Application is responsible for freeing
** memory allocated to string member of SMC_VALUE_UNION by
** library.
*/
        free(workUnion.stringValue);
    }
}
/*Check if server name has been set*/
ret=smc_connect_props(connect1_id,
                      SMC_PROP_ACT_GET,/*Property action is "get"*/
                      SMC_PROP_SERVERNAME,
                      &workUnion,
                      SMC_UNUSED,      /*Length parameter ignored
                                         on "get" operations*/
                      &outputLength    /*Note this is a pointer!*/
                      );
if (ret != SMC_RET_SUCCESS) {
    printf ("Could not get server name. Execution continuing.\n");
}
else {
    if (outputLength == 0) {
        printf("Server name not set. Quitting execution.\n");
        exit(SMC_RET_FAILURE);
    }
    else {
```

```
        free(workUnion.stringValue);
    }
}
/*
** Allocate properties for second connection. No need to
** repeat error checking.
*/
ret=smc_connect_props(connect2_id,SMC_PROP_ACT_SET, \
    SMC_PROP_USERNAME,&userNameUnion,SMC_NULLTERM, NULL);
if (ret != SMC_RET_SUCCESS) {
    printf("Could not set user name for second connection.\n");
    exit(SMC_RET_FAILURE);
}
ret=smc_connect_props(connect2_id,SMC_PROP_ACT_SET, \
    SMC_PROP_PASSWORD,&passwordUnion,SMC_NULLTERM,NULL);
if (ret != SMC_RET_SUCCESS) {
    printf("Could not set password for second connection.\n");
    exit(SMC_RET_FAILURE);
}
ret=smc_connect_props(connect2_id,SMC_PROP_ACT_SET, \
    SMC_PROP_SERVERNAME,&serverNameUnion,SMC_NULLTERM,NULL);
if (ret != SMC_RET_SUCCESS) {
    printf("Could not set server name for second connection.\n");
    exit(SMC_RET_FAILURE);
}
if (explicitInterfacesFile) {
    ret=smc_connect_props(connect2_id,SMC_PROP_ACT_SET, \
        SMC_PROP_IFILE,&interfacesFileUnion,SMC_NULLTERM,NULL);
    if (ret != SMC_RET_SUCCESS) {
        printf("Could not set server name for second connection.\n");
        exit(SMC_RET_FAILURE);
    }
}
/*
** Optional smc_connect_props call to set user-defined pointer to
** be passed to error callback. This pointer points to a
** string that tells where the error callback was triggered.
*/
workUnion.voidpValue = "second connection"; /*Call to set user
                                             data handle looks for
                                             value to set in void
                                             pointer member
                                             of union.*/
ret=smc_connect_props(connect2_id,SMC_PROP_ACT_SET, \
    SMC_PROP_USERDATA,&workUnion,SMC_NULLTERM,NULL);
if (ret != SMC_RET_SUCCESS){
```

```
printf("smc_connect_props call failed to set userDataHandle.\n");
}
/*
** Connect to monitor server

                For commentary, see "Connecting to a server" on page 7

*/
/*
** First connection
*/
ret=smc_connect_ex(connect1_id);
if (ret != SMC_RET_SUCCESS) {
    printf("First connection failed to connect to \
            monitor server.\n");
    exit(SMC_RET_FAILURE);
}
/*
** Second connection
*/
ret=smc_connect_ex(connect2_id);
if (ret != SMC_RET_SUCCESS) {
    printf("Second connection failed to connect to \
            monitor server.\n");
    exit(SMC_RET_FAILURE);
}
/*
** Create views on connections.
*/
```

For commentary, see "Step 3: create a view" on page 7

```
** Define views.

/*
** Each data item must be paired with a
** statistic type . View definitions are used in create_view
** calls after connecting to monitor server.
*/
/*This is a server-wide view that returns one row of data*/
server_info_view[0].dataItemName =SMC_NAME_SQL_SERVER_NAME;
server_info_view[0].dataItemStatType = SMC_STAT_VALUE_SAMPLE;
server_info_view[1].dataItemName = SMC_NAME_SQL_SERVER_VERSION;
server_info_view[1].dataItemStatType = SMC_STAT_VALUE_SAMPLE;
server_info_view[2].dataItemName = SMC_NAME_TIMESTAMP;
server_info_view[2].dataItemStatType = SMC_STAT_VALUE_SAMPLE;
/*
```

```
** This is a view with key and result data items that returns
** multiple rows of data.
*/
db_info_view[0].dataItemName = SMC_NAME_DB_ID; /*Key data items*/
db_info_view[0].dataItemStatType = SMC_STAT_VALUE_SAMPLE;
db_info_view[1].dataItemName = SMC_NAME_OBJ_ID;
db_info_view[1].dataItemStatType = SMC_STAT_VALUE_SAMPLE;
db_info_view[2].dataItemName = SMC_NAME_DB_NAME; /*Result data
items*/
db_info_view[2].dataItemStatType = SMC_STAT_VALUE_SAMPLE;
db_info_view[3].dataItemName = SMC_NAME_OBJ_NAME;
db_info_view[3].dataItemStatType = SMC_STAT_VALUE_SAMPLE;
db_info_view[4].dataItemName = SMC_NAME_PAGE_HIT_PCT;
db_info_view[4].dataItemStatType = SMC_STAT_VALUE_SAMPLE;
db_info_view[5].dataItemName = SMC_NAME_PAGE_IO;
db_info_view[5].dataItemStatType = SMC_STAT_VALUE_SAMPLE;
db_info_view[6].dataItemName = SMC_NAME_PAGE_LOGICAL_READ;
db_info_view[6].dataItemStatType = SMC_STAT_VALUE_SAMPLE;
db_info_view[7].dataItemName = SMC_NAME_PAGE_LOGICAL_READ;
db_info_view[7].dataItemStatType = SMC_STAT_VALUE_SESSION;
db_info_view[8].dataItemName = SMC_NAME_PAGE_LOGICAL_READ;
db_info_view[8].dataItemStatType = SMC_STAT_RATE_SAMPLE;
db_info_view[9].dataItemName = SMC_NAME_PAGE_LOGICAL_READ;
db_info_view[9].dataItemStatType = SMC_STAT_RATE_SESSION;
db_info_view[10].dataItemName = SMC_NAME_PAGE_PHYSICAL_READ;
db_info_view[10].dataItemStatType = SMC_STAT_VALUE_SAMPLE;
db_info_view[11].dataItemName = SMC_NAME_PAGE_PHYSICAL_READ;
db_info_view[11].dataItemStatType = SMC_STAT_VALUE_SESSION;
db_info_view[12].dataItemName = SMC_NAME_PAGE_PHYSICAL_READ;
db_info_view[12].dataItemStatType = SMC_STAT_RATE_SAMPLE;
db_info_view[13].dataItemName = SMC_NAME_PAGE_PHYSICAL_READ;
db_info_view[13].dataItemStatType = SMC_STAT_RATE_SESSION;
/*
** Another server-wide view
*/
nw_bytes_view[0].dataItemName = SMC_NAME_NET_BYTES_RCVD;
nw_bytes_view[0].dataItemStatType = SMC_STAT_VALUE_SAMPLE;
nw_bytes_view[1].dataItemName = SMC_NAME_NET_BYTES_RCVD;
nw_bytes_view[1].dataItemStatType = SMC_STAT_VALUE_SESSION;
nw_bytes_view[2].dataItemName = SMC_NAME_NET_BYTES_SENT;
nw_bytes_view[2].dataItemStatType = SMC_STAT_VALUE_SAMPLE;
nw_bytes_view[3].dataItemName = SMC_NAME_NET_BYTES_SENT;
nw_bytes_view[3].dataItemStatType = SMC_STAT_VALUE_SESSION;
nw_bytes_view[4].dataItemName = SMC_NAME_NET_BYTE_IO;
nw_bytes_view[4].dataItemStatType = SMC_STAT_RATE_SAMPLE;
nw_bytes_view[5].dataItemName = SMC_NAME_NET_BYTE_IO;
```

```
    nw_bytes_view[5].dataItemStatType = SMC_STAT_RATE_SESSION;

ret=smc_create_view (connect1_id,      /*Connect ID assigned when
                                       connect allocated*/
                    server_info_view, /*This is a pointer to
                                       array of SMC_DATAITEM_STRUCTS
                                       which defines the view*/
                    NUM_SERVER_DATA_ITEMS, /*No. of items in
                                       the view*/
                    "server info view", /*Ignored on a live
                                       connection*/
                    &server_view_id    /*Value is assigned
                                       by this call*/
                    );
if (ret != SMC_RET_SUCCESS) {          /*Cleanup from failed
                                       create_view call*/
    ret=smc_connect_drop(connect1_id); /*Create view failed
                                       so no further use for
                                       this connection*/

    connect1_id = CONNECT_NONEXISTENT;
}
/*
** The second connection will have two views
*/
ret=smc_create_view(connect2_id,db_info_view,NUM_DB_INFO_ITEMS,
                    "db info view",&db_info_view_id);
if (ret != SMC_RET_SUCCESS) {
    db_info_view_id = VIEW_NONEXISTENT;
}
ret=smc_create_view(connect2_id,nw_bytes_view,NUM_NW_INFO_ITEMS,
                    "nw bytes view",&nw_info_view_id);
if (ret != SMC_RET_SUCCESS) {
    nw_info_view_id = VIEW_NONEXISTENT;
}
/*
** Create a filter.
*/
```

For commentary, see “Step 4: create filters” on page 10

```
/*
** Filters and alarms may be applied to data items within a view.
** This is optional.
** In this case, we only want to see I/O activity for a
** particular database and tempdb. If any physical reads occur,
** an alarm is triggered that posts a message to the screen.
```

```
*/
#ifdef OPTIONAL_CALLS
    filter_strings[0] = "my_db";      /*Change to db of interest*/
    filter_strings[1] = "tempdb";
    workUnion.voidpValue = filter_strings;
    ret=smc_create_filter(connect2_id,      /*Connection id*/
                        db_info_view_id,   /*View id*/
                        &db_info_view[2], /*Pointer to a data
                                           item within the view
                                           to be filtered*/
                        SMC_FILT_T_EQ,     /*Type of filter*/
                        &workUnion,       /*Filter value*/
                        2,                 /*Number of elements
                                           in array of filter
                                           values*/
                        SMC_DI_TYPE_CHARP, /*datatype of filter
                                           values*/
                        &filter_id         /*Value is assigned by
                                           this function call*/
                    );
    if (ret != SMC_RET_SUCCESS) {
        printf("Filters were not applied. Continuing execution.\n");
    }
/*
** Set alarms.
*/
```

For commentary, see “Step 5: set alarms” on page 11

```
workUnion.longValue = 1;          /*Value above which
                                   alarm is triggered*/
ret=smc_create_alarm_ex(connect2_id,  /*Connection id*/
                        db_info_view_id, /*View id*/
                        &db_info_view[11], /*Pointer to a data
                                           item within the view
                                           to which the alarm
                                           is applied*/
                        &workUnion,     /*Where value that
                                           triggers the alarm
                                           is located*/
                        SMC_DI_TYPE_LONG, /*datatype of item
                                           to which alarm is
                                           applied*/
                        SMC_ALARM_A_NOTIFY, /*Trigger alarm
                                           callback function.
                                           This is the only
```

```

                                action possible when
                                the server mode is
                                LIVE.*/
        NULL,    /*For server mode HISTORICAL,
                  this is where log file to be
                  written to or program to be
                  run is specified. For server
                  mode LIVE, this field is
                  ignored.*/

/*The following is a string that is passed to the alarm callback function.*/
        "Physical read occurred in database.",
        alarmCallback,    /*Alarm callback
                           function*/
        &alarm_id        /*Variable into which
                           alarm id is placed.*/
    );
    if (ret != SMC_RET_SUCCESS) {
        printf("Alarm was not applied. Execution continuing.\n");
    }
#endif
/*
** Request data and process results.
*/
```

For commentary, see “Step 6: request performance data and process results”
on page 12

```
/*
** Get data from first connection. As server name and version
** do not change during the connection, we only get it once.
** Post the time when the refresh was done.
*/
    if (connect1_id != CONNECT_NONEXISTENT) { /*If the connect is
                                              not successful, the
                                              error callback is
                                              triggered. For a
                                              friendlier display,
                                              we check first.*/
        ret=smc_refresh_ex(connect1_id,    /*ID of connect*/
                           0              /*STEP not used in
                                              live connection*/
                           );
        if (ret != SMC_RET_SUCCESS) {
            printf("refresh call failed on first connect ID.\n");
        }
    }
```

```

else {
    /*Check row count even though only one
    row is expected in this case. If no
    rows are returned, get_dataitem_value
    calls will return errors.*/
    ret=smc_get_row_count(connect1_id,
                          server_view_id,
                          #_of_rows);
    if (ret != SMC_RET_SUCCESS){
        printf("Get row count call failed.\n");
    }
    else {
        if (num_of_rows > 0){
/*
** A get_dataitem_value call is made for each item in the view.
** The retrieved data is stored in an array of SMC_VALUE_UNIONS.
**/
            for (index=0;index <NUM_SERVER_DATA_ITEMS;index++){
                ret=smc_get_dataitem_value(connect1_id,
                                           server_view_id,
                                           &server_info_view[index],/*Look at
                                           each data
                                           item in
                                           the view*/
                                           0,
                                           /*Only one row of
                                           data is returned for
                                           this particular view,
                                           so the value for row
                                           is hard-coded in this
                                           case.*/
                                           &server_data[index] /*Retrieved
                                           data stored
                                           here*/
                );
            }
            /*End for loop*/
/*
** Display the returned data.
**/
            printf("Adaptive Server Enterprise name is: \
                %s.\n",server_data[0].stringValue);
            printf("Adaptive Server Enterprise version is: \
                %s.\n",server_data[1].stringValue);
            printf("Date and time is: \
                %s.\n",server_data[2].stringValue);
/*
** The application is responsible for freeing memory allocated
** by the Monitor Client Library for string members of

```

```
** SMC_VALUE_UNIONS. This also illustrates the use of the
** smc_get_dataitem_type function call.
*/
for (index=0;index <NUM_SERVER_DATA_ITEMS;index++){
    ret=smc_get_dataitem_type(&server_info_view[index], \
                             &dataitem_type);
    if (ret != SMC_RET_SUCCESS) {
        printf("Get dataitem type failed for item %d \
              in server_info_view.\n");
    }
    else {
        if (dataitem_type == SMC_DI_TYPE_CHARP) {
            free(server_data[index].stringValue);
        }
    }
}
/*End for loop*/
}
/*End if number of rows > 0*/
}
/*End case get_row_count was successful*/
}
/*End case smc_refresh_ex call was successful*/
}
/*End case connect still valid*/
/*
** Get the data from the views in the second connection to see
** how the data changes over time. To do this, we sample
** NUM_OF_SAMPLES times, pausing SAMPLE_INTERVAL times between
** each sample. The process of retrieving data is within a loop.
*/
for (iterations=0;iterations<NUM_OF_SAMPLES;iterations++){
    sleep(SAMPLE_INTERVAL);
    ret=smc_refresh_ex(connect2_id,      /*Note second connection
                                         specified for refresh*/
                      0,                /*Step not used in live
                                         connection*/
                      );
    if (ret == SMC_RET_SUCCESS) {
        if (db_info_view_id != VIEW_NONEXISTENT){ /*Attempting
                                                    get_row_count for
                                                    nonexistent view
                                                    will cause errors
                                                    so check if view
                                                    was actually
                                                    created*/
            ret=smc_get_row_count(connect2_id,
                                   db_info_view_id,
                                   #_of_rows /*Multiple rows will
                                             be returned. For
                                             each row of data
```

```

                                returned, use
                                get_dataitem_value
                                loop. Function call
                                puts number of rows
                                returned into
                                variable.*/
                                );
for(row=0;row<num_of_rows;row++){
    for (index=0;index <NUM_DB_INFO_ITEMS;index++){
        ret=smc_get_dataitem_value(connect2_id,
                                db_info_view_id, /*View specified for
                                                get_dataitem_value.*/
                                &db_info_view[index],
                                row,                /*Multiple rows in
                                                this case */
                                &db_data[index]
                                );
        if (ret != SMC_RET_SUCCESS) {
            printf("Get dataitem value failed for data item \
                %s.\n",db_info_labels[index]);
        }
        else {
            printf("%s",db_info_labels[index]);
            ret=smc_get_dataitem_type(&db_info_view[index],\
                                    &dataitem_type);
            if (ret != SMC_RET_SUCCESS){
                printf("Get data item type failed for data item \
                    %s.\n",db_info_view[index]);
            }
            else {
                switch (dataitem_type) {
                    case SMC_DI_TYPE_CHARP:
                        printf("%s.\n",db_data[index].stringValue);
                        free(db_data[index].stringValue);
                        /*Application is responsible for freeing
                        memory allocated for strings by library*/
                        break;
                    case SMC_DI_TYPE_LONG:
                        printf("%d.\n",db_data[index].longValue);
                        break;
                    case SMC_DI_TYPE_DOUBLE: /*Rates are generally
                                                floating point variables*/
                        printf("%f.\n",db_data[index].doubleValue);
                        break;
                    default:
                        printf("Unknown datatype encountered.\n");
                }
            }
        }
    }
}
```

```
        break;
    } /*End switch*/
} /*End case get_dataitem_type successful*/
} /*End case get_dataitem_value successful*/
} /*End for loop to get each data item value*/
} /*End for loop to get each row of data*/
} /*End case view exists*/
/*
** Retrieve data from second view in refresh.
** Processing is much the same.
*/
if (nw_info_view_id != VIEW_NONEXISTENT){ /*Attempting
                                           get_row_count for
                                           nonexistent view
                                           causes errors, so
                                           check to see if
                                           view was actually
                                           created*/

ret=smc_get_row_count(connect2_id,
                      nw_info_view_id,
                      #_of_rows /*This is a server-
                                wide view so only
                                one row should be
                                returned*/

                      );
if (num_of_rows > 0 ){
    for (index=0;index < NUM_NW_INFO_ITEMS;index++){
        ret=smc_get_dataitem_value(connect2_id,
                                    nw_info_view_id, /*Note view
                                                    specified for
                                                    get_dataitem_value*/
                                    &nw_bytes_view[index],
                                    0, /*One row in this case*/
                                    &nw_data[index]
                                    );
        if (ret != SMC_RET_SUCCESS) {
            printf("Get dataitem value failed for data item \
                    %s.\n",nw_info_labels[index]);
        }
        else {
            printf("%s",nw_info_labels[index]);
            ret=smc_get_dataitem_type(&nw_bytes_view[index],\
                                     &dataitem_type);
            if (ret != SMC_RET_SUCCESS){
                printf("Get data item type failed for data item \
                        %s.\n",nw_bytes_view[index]);
            }
        }
    }
}
```

```

    }
    else {
        switch (dataitem_type) {
            case SMC_DI_TYPE_CHARP:
                printf("%s.\n",nw_data[index].stringValue);
                free(nw_data[index].stringValue);
                /*Application is responsible for freeing
                memory allocated for strings by library*/
                break;
            case SMC_DI_TYPE_LONG:
                printf("%d.\n",nw_data[index].longValue);
                break;
            case SMC_DI_TYPE_DOUBLE:    /*Rates are generally
                                        floating point
                                        variables*/
                printf("%f.\n",nw_data[index].doubleValue);
                break;
            default:
                printf("Unknown datatype encountered.\n");
                break;
        }    /*End switch*/
    }    /*End case get_dataitem_type successful*/
}    /*End case get_dataitem_value successful*/
}    /*End for loop to get each data item value*/
}    /*End if any rows of data returned*/
else {
    printf("No data returned for network info view.\n");
}
}    /*End case view exists*/
}    /*End case refresh successful*/
else {
    printf("Refresh of second connect failed. \
        Return code is %d.\n",ret);
}
}    /*End for loop for number of iterations*/
/*
** This shows how to drop filters and alarms. It is not necessary
** to do this prior to closing a connection, as it is done
** automatically when the connection is closed. Filters may be
** dropped, for example, to see the filtered results of a query
** followed by the unfiltered results.
*/
#ifdef OPTIONAL_CALLS
    ret=smc_drop_filter(connect2_id,db_info_view_id,filter_id);
    if (ret != SMC_RET_SUCCESS) {
        printf("Attempt to drop filter failed.\n");
    }
}

```

```
    }
    ret=smc_drop_alarm(connect2_id,db_info_view_id,alarm_id);
    if (ret != SMC_RET_SUCCESS) {
        printf("Attempt to drop alarm failed.\n");
    }
#endif
/*
** Get another time stamp before disconnecting. To do this,
** do a refresh on the first connection again and only display
** the time stamp data returned.
*/
if (connect1_id != CONNECT_NONEXISTENT) {
    ret=smc_refresh_ex(connect1_id,0 );
    if (ret != SMC_RET_SUCCESS) {
        printf("refresh call failed on first connect ID.\n");
    }
    else {
        /*Check row count even though
        only one row is expected. If
        no rows are returned,
        get_dataitem_value calls
        will return errors.*/
        ret=smc_get_row_count(connect1_id,
                               server_view_id,
                               #_of_rows);
        if (ret != SMC_RET_SUCCESS){
            printf("Get row count call on first connection \
                failed.\n");
        }
        else {
            if (num_of_rows > 0){
                ret=smc_get_dataitem_value(connect1_id,
                                             server_view_id,
                                             &server_info_view[2], /*In this case
                                             we are only
                                             interested in
                                             the third data
                                             item*/
                                             0, /*Only one row of data
                                             is returned for this
                                             particular view, so the
                                             value for row is hard-
                                             coded in this case.*/
                                             &server_data[2]
                                             );
                printf("Date and time on conclusion of monitoring:\
                    %s\n",server_data[2].stringValue);
            }
        }
    }
}
```

```

        free(server_data[2].stringValue);
        /*Application must free string memory returned
        by library*/
    }
    }
}
/*
** Close and deallocate the connection.
*/

```

For commentary, see “Step 7: close and deallocate connections” on page 13

```

/*
** Cleanup. This consists of closing all connections, then
** de-allocating them. Alternatively, connections can be re-used.
*/
ret=smc_close(connect1_id,
              SMC_CLOSE_REQUEST           /*Close only if no
                                          outstanding commands
                                          (only close request type
                                          currently supported)*/
              );
if (ret != SMC_RET_SUCCESS) {
    printf("Attempt to close first connection failed. \
          Return code is %d.\n",ret);
}
ret=smc_close(connect2_id,SMC_CLOSE_REQUEST);
if (ret != SMC_RET_SUCCESS) {
    printf("Attempt to close second connection failed. \
          Return code is %d.\n",ret);
}
/*
** Connections can be re-used at this point, for example, to
** connect to different servers. However, we de-allocate them.
*/
ret=smc_connect_drop(connect1_id);
if (ret != SMC_RET_SUCCESS){
    printf("Attempt to drop first connection failed. \
          Return code is %d.\n",ret);
}
ret=smc_connect_drop(connect2_id);
if (ret != SMC_RET_SUCCESS){
    printf("Attempt to drop second connection failed. \
          Return code is %d.\n",ret);
}

```

```
    }
    return(SMC_RET_SUCCESS);
}
/*
** Callback functions

                For commentary, see "Step 1: define error handling" on page 5.

*/
SMC_VOID errorCallback(
    SMC_CONNECT_ID connectID,
    SMC_COMMAND_ID commandID,          /*Value internal to Monitor
                                        Client Library*/
    SMC_VOIDP userDataHandle          /*User-defined pointer. Set by
                                        smc_connect_propscall*/
)
{
    SMC_SIZET      ret;
    SMC_VALUE_UNION  errorInfo;        /*Used for getting information
                                        from smc_get_command_info
                                        function call*/

    SMC_SIZET      returned_msg_length;
    printf ("Inside new error callback.\n");
}
/*
** Use smc_get_command_info function call to get information
** from error and alarm callbacks.
*/
ret=smc_get_command_info(connectID,
                        commandID,
                        SMC_INFO_ERR_MAPSEVERITY, /*Information
                                                    requested about
                                                    command*/
                        &errorInfo,              /*Where information
                                                    returned about
                                                    command is placed*/
                        NULL                       /*Value is numeric
                                                    so length of returned
                                                    data not needed*/
                        );
if (ret != SMC_RET_SUCCESS){
    printf("get_command_info call requesting error map \
          severity failed. Error returned is:  %d\n",ret);
}
else{
    printf("Monitor Client Library error severity level is: \
          %d\n",errorInfo.sizetValue);
}
```

```
ret=smc_get_command_info(connectID,
                          commandID,
                          SMC_INFO_ERR_MSG,
                          &errorInfo,
                          &returned_msg_length    /*Find string
                                                    length */
                          );

if (ret != SMC_RET_SUCCESS){
    printf("get_command_info call requesting error message \
          failed. Error returned is:  %d\n",ret);
}
else{
    printf("Error message text is:  %s\n",errorInfo.stringValue);
    free(errorInfo.stringValue);
    /*Application is responsible for freeing string buffer
    memory allocated by library*/
}
ret=smc_get_command_info(connectID,
                          commandID,
                          SMC_INFO_ERR_NUM,
                          &errorInfo,
                          NULL
                          );
if (ret != SMC_RET_SUCCESS){
    printf("get_command_info call requesting error number \
          failed. Error returned is:  %d\n",ret);
}
else{
    printf("Error number is:  %d\n",errorInfo.sizetValue);
}
ret=smc_get_command_info(connectID,
                          commandID,
                          SMC_INFO_ERR_SEVERITY,
                          &errorInfo,
                          NULL
                          );
if (ret != SMC_RET_SUCCESS){
    printf("get_command_info call requesting error severity \
          failed. Error returned is:  %d\n",ret);
}
else{
    printf("Error severity level is:  %d\n",errorInfo.sizetValue);
}
ret=smc_get_command_info(connectID,
                          commandID,
                          SMC_INFO_ERR_SOURCE,
```

```

        &errorInfo,
        NULL
    );
    if (ret != SMC_RET_SUCCESS){
        printf("get_command_info call requesting error source \
            failed. Error returned is: %d\n",ret);
    }
    else{
        printf(" Error source is: %d\n",errorInfo.sizetValue);
    }
    ret=smc_get_command_info(connectID,
        commandID,
        SMC_INFO_ERR_STATE,
        &errorInfo,
        NULL
    );
    if (ret != SMC_RET_SUCCESS){
        printf("get_command_info call requesting state failed. \
            Error returned is: %d\n",ret);
    }
    else{
        printf(" Error state is: %d\n",errorInfo.sizetValue);
    }
}
/*
** Demonstrate use of userDataHandle. This value was set as a
** connection property for the connection in the main program and
** is passed to this function.
*/
    if (userDataHandle != NULL){
        printf("Connection on which error occurred is \
            %s.\n",userDataHandle);
    }
}
/*End errorCallback */
/*Alarm callback*/
SMC_VOID alarmCallback(
    SMC_CONNECT_ID connectID,
    SMC_COMMAND_ID commandID,
    SMC_VOIDP userDataHandle
)
{
#define MSG_BUFFER_LENGTH 80
    SMC_SIZET    ret;
    SMC_VALUE_UNION    alarmInfo;
    SMC_SIZET    returned_msg_length;
    /*Union into which requested
    data is placed*/
}
```

```
    printf ("Alarm callback triggered.\n");
/*
** Use smc_get_command_info function call to get information
** from error and alarm callbacks.
*/
    ret=smc_get_command_info(connectID,
                             commandID,
                             SMC_INFO_ALARM_ALARMID,
                             &alarmInfo,
                             NULL
                             );

    if (ret != SMC_RET_SUCCESS){
        printf("get_command_info call failed. \
              Error returned is:  %d",ret);
    }
    else{
        printf("Alarm ID is:  %d\n",alarmInfo.sizetValue);
    }
/*
** This demonstrates the use of the SMC_INFO_ALARM_VALUE_DATATYPE
** information that might be useful in a generic alarm callback
** function.
*/
    ret=smc_get_command_info(connectID,
                             commandID,
                             SMC_INFO_ALARM_VALUE_DATATYPE,
                             &alarmInfo,
                             NULL
                             );

    if (ret != SMC_RET_SUCCESS){
        printf("get_command_info call failed. \
              Error returned is:  %d",ret);
    }
    else{
        switch(alarmInfo.intValue){
        case SMC_DI_TYPE_INT:
            ret=smc_get_command_info(connectID,
                                     commandID,
                                     SMC_INFO_ALARM_CURRENT_VALUE,
                                     &alarmInfo,
                                     NULL
                                     );

            if (ret != SMC_RET_SUCCESS){
                printf("get_command_info call failed. \
                      Error returned is:  %d",ret);
            }
        }
```

```
else {
    printf("Current value of alarmed data item is:\n
           %d.\n",alarmInfo.intValue);
}
break;
case SMC_DI_TYPE_LONG:
    ret=smc_get_command_info(connectID,
                             commandID,
                             SMC_INFO_ALARM_CURRENT_VALUE,
                             &alarmInfo,
                             NULL
                             );
if (ret != SMC_RET_SUCCESS){
    printf("get_command_info call failed. \
           Error returned is:  %d",ret);
}
else {
    printf("Current value of alarmed data item is:\n
           %d.\n",alarmInfo.longValue);
}
break;
case SMC_DI_TYPE_DOUBLE:
    ret=smc_get_command_info(connectID,
                             commandID,
                             SMC_INFO_ALARM_CURRENT_VALUE,
                             &alarmInfo,
                             NULL
                             );
if (ret != SMC_RET_SUCCESS){
    printf("get_command_info call failed.  Error returned is:  %d",ret);
}
else {
    printf("Current value of alarmed data item is: \
           %f.\n",alarmInfo.doubleValue);
}
break;
default:
    printf("Invalid value returned for datatype of \
           current alarm value.\n");
    break;
}
/*End switch*/
}
ret=smc_get_command_info(connectID,
                         commandID,
                         SMC_INFO_ALARM_ROW,
                         &alarmInfo,
```

```
        NULL
    );
if (ret != SMC_RET_SUCCESS){
    printf("get_command_info call failed. \
        Error returned is:  %d",ret);
}
else{
    printf("Row of data which triggered alarm is: \
        %d\n",alarmInfo.sizetValue);
}
ret=smc_get_command_info(connectID,
    commandID,
    SMC_INFO_ALARM_VALUE_DATATYPE,
    &alarmInfo,
    NULL
);
if (ret != SMC_RET_SUCCESS){
    printf("get_command_info call failed. \
        Error returned is:  %d",ret);
}
else{
    switch(alarmInfo.intValue){
    case SMC_DI_TYPE_INT:
        ret=smc_get_command_info(connectID,
            commandID,
            SMC_INFO_ALARM_THRESHOLD_VALUE,
            &alarmInfo,
            NULL
        );
        if (ret != SMC_RET_SUCCESS){
            printf("get_command_info call failed. \
                Error returned is:  %d",ret);
        }
        else {
            printf("Value of data item exceeded alarm-triggering \
                value of:  %d.\n",alarmInfo.intValue);
        }
        break;
    case SMC_DI_TYPE_LONG:
        ret=smc_get_command_info(connectID,
            commandID,
            SMC_INFO_ALARM_THRESHOLD_VALUE,
            &alarmInfo,
            NULL
        );
        if (ret != SMC_RET_SUCCESS){
```

```
    printf("get_command_info call failed. \
        Error returned is:  %d",ret);
}
else {
    printf("Value of data item exceeded alarm-triggering \
        value of:  %d.\n",alarmInfo.longValue);
}
break;
case SMC_DI_TYPE_DOUBLE:
    ret=smc_get_command_info(connectID,
                            commandID,
                            SMC_INFO_ALARM_THRESHOLD_VALUE,
                            &alarmInfo,
                            NULL
                            );
if (ret != SMC_RET_SUCCESS){
    printf("get_command_info call failed. \
        Error returned is:  %d",ret);
}
else {
    printf("Value of data item exceeded alarm-triggering\
        value of:  %f.\n",alarmInfo.doubleValue);
}
break;
default:
    printf("Invalid value returned for datatype of \
        THRESHOLD alarm value.\n");
    break;
}
/*End switch*/
}
ret=smc_get_command_info(connectID,
                        commandID,
                        SMC_INFO_ALARM_TIMESTAMP,
                        &alarmInfo,
                        &returned_msg_length
                        );
if (ret != SMC_RET_SUCCESS){
    printf("get_command_info call failed. \
        Error returned is:  %d",ret);
}
else{
    printf("Time when alarm was triggered is: \
        %s\n",alarmInfo.stringValue);
    free(alarmInfo.stringValue); /*Application is responsible
                                for freeing string buffer memory
                                allocated by library.*/
}
```

```
    }

    ret=smc_get_command_info(connectID,
                             commandID,
                             SMC_INFO_ALARM_VIEWID,
                             &alarmInfo,
                             NULL
                             );
    if (ret != SMC_RET_SUCCESS){
        printf("get_command_info call failed. \
              Error returned is:  %d",ret);
    }
    else{
        printf("ID of view which triggered alarm is: \
              %d.\n",alarmInfo.sizetValue);
    }
}                                     /*End newAlarmCallback*/
```


Data Items and Statistical Types

This chapter contains the following topics:

| Topics | Page |
|---------------------------|------|
| Overview | 43 |
| Result and key data items | 43 |
| Data items and views | 44 |
| Data item definitions | 47 |

Overview

A data item is a particular piece of performance data that can be obtained by using Monitor Client Library. A statistical type specifies the calculations to be performed and the duration for which to report the data collected by the data item.

This chapter describes the types of data items and statistical types. It also describes each data item and its characteristics.

Monitor Client Library terminology is defined in “Overview” on page 1.

Result and key data items

Data items are classified as keys or results:

- A *key data item* refines the amount of detail in a view and usually results in additional rows returned when a view is refreshed. With the inclusion of each successive key, envision adding the word “per” to a view definition. For example, start with the Page I/O result data item. Refine the granularity by adding the Database key data item, Page I/Os “per” Database. Further refine the granularity by adding the Object key data item, Page I/Os “per” Database “per” Object.

- A *result data item* returns performance data at the level of detail determined by the key data items in a view. If no key data items are specified, only one row of data is returned.

Note A data item’s designation as a result or key is a characteristic of the data item and is independent of the statistical type associated with the data item in a view.

Data items and views

A view usually contains a mix of key and result data items. This mixture of keys and results provides flexibility in determining the amount of detail of the data to be returned. The exception is server-wide data, such as transaction or network activity data. For server-wide data, no key data items are specified and only one row of data is returned.

Table 2-1 shows examples of data returned by views.

Table 2-1: Examples of data returned by views

| View defined with | Returns | | |
|--|--------------------------------|------------|----------|
| SMC_NAME_PAGE_IO | page I/Os for the whole server | | |
| | Row results: | | |
| | Page I/O | | |
| | ----- | | |
| | 145 | | |
| SMC_NAME_SPID, SMC_NAME_LOGIN_NAME, SMC_NAME_PAGE_IO | page I/O per process | | |
| (where SPID is a key data item) | Row results: | | |
| | SPID | Login Name | Page I/O |
| | ----- | | |
| | 3 | sa | 45 |
| | 5 | joe | 100 |

| View defined with | Returns | | | | | |
|-----------------------|---|------|-----------|--------|---------|--------|
| SMC_NAME_SPID, | page I/O per database table per process | | | | | |
| SMC_NAME_DB_ID, | Row results: | | | | | |
| SMC_NAME_OBJ_ID, | SPID | DBID | ObjID | DBName | ObjName | PageIO |
| SMC_NAME_DB_NAME, | ----- | | | | | |
| SMC_NAME_OBJ_NAME, | 1 | 5 | 208003772 | pubs2 | titles | 10 |
| and | 1 | 5 | 336004228 | pubs2 | blurbs | 5 |
| SMC_NAME_PAGE_IO | 5 | 5 | 22003430 | pubs2 | sales | 100 |
| (where SMC_NAME_SPID, | | | | | | |
| SMC_NAME_DB_ID, and | | | | | | |
| SMC_NAME_OBJID are | | | | | | |
| key data items) | | | | | | |

Rows with no data versus no rows in views

When there is no activity to report, some data items cause an empty row (that is, a row with zero values for result data items) to appear in a view, and other data items cause the row to be omitted. The rules controlling whether empty rows appear in a view are:

- Server level data items always return a row, even when there is no activity to report.
- Views that contain the key data item SMC_NAME_SPID or SMC_NAME_APPLICATION_NAME only report on processes that are active as of the end of the sample period.
- Views that contain the key data items SMC_NAME_OBJ_ID or SMC_NAME_ACT_STP_ID omit the row when there is no activity to report during the sample period.
- Views that contain keys other than those listed in the previous bullets return rows when there is no activity.

Server-level status

Some data items are available only at the server level. Views with server-level data items contain only result data items and provide performance data summarized over Adaptive Server.

Combining data items

Data items cannot be combined indiscriminately. The absence or presence of a key data item in a view determines which other data items are allowed in the view.

If a view contains a key data item, all result data items in the view must be valid for the key data item. Also, for each result data item in a view, all required keys for that result data item must be in the view.

If a view does not contain a key data item, it can include any data item that does not require a key.

Result and key combinations

In some cases, if you use an optional key data item, you must also use one or more others. In the data item descriptions in this chapter, data items that have this requirement are grouped with the other required data items in brackets and separated by a plus sign (+).

Not all result data items require a key data item. If a view contains only result data items, by default the summary is at the server level. The result data items that have only optional keys can be used with server-level data items when no key data item is included in the view.

To combine various result data items within a view, match common key data items.

Connection summaries

Some views consume Monitor Server connection summaries. For information about Monitor Server connection summaries, refer to the *Adaptive Server Enterprise Monitor Server User's Guide*.

Current statement and application name data items

If you want to get data for a current statement data item (SMC_NAME_CUR_STMT_x) or SMC_NAME APPLICATION NAME, the monitor client application must connect to the monitor server and create the view before you start the application you are monitoring.

Data item definitions

This section lists data items in alphabetical order with the following information:

- Description
- Server level status
- Result or key designation
- For result data items, required keys and optional keys
- For key data items, result data items that require the key data item and result data items that can use the key data item, but do not require it
- Version compatibility: either SQL Server 11.0 or Adaptive Server 11.5 and later
- Valid statistical types

The valid statistical types are as follows:

- SMC_STAT_VALUE_SAMPLE
- SMC_STAT_VALUE_SESSION
- SMC_STAT_RATE_SAMPLE
- SMC_STAT_RATE_SESSION
- SMC_STAT_AVG_SAMPLE
- SMC_STAT_AVG_SESSION

The possible datatypes for a data item are:

- LONG – long
- ENUMS – integer
- DOUBLE – double
- CHARP – character
- DATIM – date/time

For more information about enumerated types, see Appendix B, “Datatypes and Structures”.

Note Not all statistical types are available for each data item.

You cannot use SMC_NAME_SPID and SMC_NAME_APPLICATION_NAME in the same view.

Deciphering the names of data items

The syntax of a data item's name is an abbreviation of a description of the information it reports. All data items start with SMC_NAME. The remaining components of the name are either English words, abbreviations, or both. The abbreviations and their meanings are:

- ACT – active
- APP – application
- CNT – count (number of)
- CUR – current
- DATIM – date and time
- DB – database
- DEV – device
- ID – identification number
- IMMED – immediate
- IO – input/output (page reads and writes)
- KPID – a persistent process ID
- MAX – maximum
- MEM – memory
- NET – network
- NUM – number
- OBJ – database object
- PCT – percent
- PKT – packet
- PROC – process
- RCVD – received
- REF – referenced

- SPID – server process ID
- STMT – statement
- STP – stored procedure
- XACT – transaction

The data items described in *Historical Server User's Guide* are equivalent to these data items, but use a natural language naming convention.

SMC_NAME_ACT_STP_DB_ID

Description Reports the database identification number of the active stored procedure.

Version compatibility 11.0 and later

Data item type Key

Server level No

Result data items that require this key

| |
|---------------------------------|
| SMC_NAME_ACT_STP_DB_NAME |
| SMC_NAME_ACT_STP_NAME |
| SMC_NAME_ACT_STP_OWNER_NAME |
| SMC_NAME_STP_CPU_TIME |
| SMC_NAME_STP_ELAPSED_TIME |
| SMC_NAME_STP_EXECUTION_CLASS |
| SMC_NAME_STP_LINE_TEXT |
| SMC_NAME_STP_NUM_TIMES_EXECUTED |

Result data items for which this key is optional

| |
|-----------------------------------|
| SMC_NAME_LOCKS_GRANTED_IMMED |
| SMC_NAME_LOCKS_GRANTED_WAITED |
| SMC_NAME_LOCKS_NOT_GRANTED |
| SMC_NAME_PAGE_INDEX_LOGICAL_READ |
| SMC_NAME_PAGE_INDEX_PHYSICAL_READ |
| SMC_NAME_PAGE_HIT_PCT |
| SMC_NAME_PAGE_IO |
| SMC_NAME_PAGE_LOGICAL_READ |
| SMC_NAME_PAGE_PHYSICAL_READ |
| SMC_NAME_PAGE_WRITE |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_ACT_STP_DB_NAME

| | |
|-----------------------|---|
| Description | Reports the database name of the active stored procedure. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_ACT_STP_DB_ID |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_ACT_STP_ID

| | |
|-----------------------|---|
| Description | Reports the identification number of the active stored procedure. |
| Version compatibility | 11.0 and later |
| Data item type | Key |
| Server level | No |
| Required keys | SMC_NAME_ACT_STP_DB_ID |

Result data items that require this key

| |
|---------------------------------|
| SMC_NAME_ACT_STP_NAME |
| SMC_NAME_ACT_STP_OWNER_NAME |
| SMC_NAME_STP_CPU_TIME |
| SMC_NAME_STP_ELAPSED_TIME |
| SMC_NAME_STP_EXECUTION_CLASS |
| SMC_NAME_STP_LINE_TEXT |
| SMC_NAME_STP_NUM_TIMES_EXECUTED |

Result data items for which this key is optional

| |
|-----------------------------------|
| SMC_NAME_LOCKS_GRANTED_IMMED |
| SMC_NAME_LOCKS_GRANTED_WAITED |
| SMC_NAME_LOCKS_NOT_GRANTED |
| SMC_NAME_PAGE_INDEX_LOGICAL_READ |
| SMC_NAME_PAGE_INDEX_PHYSICAL_READ |
| SMC_NAME_PAGE_HIT_PCT |
| SMC_NAME_PAGE_IO |
| SMC_NAME_PAGE_LOGICAL_READ |
| SMC_NAME_PAGE_PHYSICAL_READ |
| SMC_NAME_PAGE_WRITE |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_ACT_STP_NAME

Description Reports the name of the active stored procedure.

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys SMC_NAME_ACT_STP_DB_ID, SMC_NAME_ACT_STP_ID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_ACT_STP_OWNER_NAME

Description Reports the name of the owner of the active stored procedure.

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys SMC_NAME_ACT_STP_DB_ID, SMC_NAME_ACT_STP_ID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_APPLICATION_NAME

Description Reports the name of each application for which other statistics are being accumulated. Views that contain SMC_NAME_APPLICATION_NAME only report on processes that are active as of the end of the sample period.

SMC_NAME_APPLICATION_NAME is mutually exclusive with SMC_NAME_SPID in a view.

Version compatibility 11.0 and later

Data item type Key

Server level No

Result data items that require this key

SMC_NAME_APP_EXECUTION_CLASS

Result data items for which this key is optional

SMC_NAME_CPU_PCT

SMC_NAME_CPU_TIME

SMC_NAME_LOCKS_GRANTED_IMMED

SMC_NAME_LOCKS_GRANTED_WAITED

SMC_NAME_LOCKS_NOT_GRANTED

SMC_NAME_NUM_PROCESSES

SMC_NAME_PAGE_INDEX_LOGICAL_READ

SMC_NAME_PAGE_INDEX_PHYSICAL_READ

SMC_NAME_PAGE_LOGICAL_READ

SMC_NAME_PAGE_PHYSICAL_READ

SMC_NAME_PAGE_WRITE

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_APP_EXECUTION_CLASS

| | |
|-------------------------------|--|
| Description | <p>Reports the configured execution class, if any, for a given application name. The name is returned in one of the following formats:</p> <ul style="list-style-type: none"> • If the application is bound to the execution class only with scope NULL, the name of the execution class is returned. • If the application is bound to the execution class with a scope of NULL and a scope of one or more logins, an asterisk (*) is appended to the name of the execution class. • If the application is bound to the execution class only with a scope of one or more logins, an asterisk is returned. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_APPLICATION_NAME |
| Optional keys | None |
| Statistic types and datatypes | |

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_BLOCKING_SPID

| | |
|-----------------------|---|
| Description | <p>Reports the identification number of the process that holds a lock that the process indicated by the SMC_NAME_SPID data item is waiting for. If a process is not blocked, the Blocking SPID is zero.</p> |
| Version Compatibility | 11.0 and later |
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_SPID, SMC_NAME_DB_ID, SMC_NAME_OBJ_ID, SMC_NAME_LOCK_STATUS |
| Optional keys | SMC_NAME_LOCK_TYPE, SMC_NAME_PAGE_NUM |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_CONNECT_TIME

Description Reports the time elapsed (in seconds) since the process was started. If the process was active before you began monitoring it, connect time is the time you have monitored this process.

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | | | | |

SMC_NAME_CPU_BUSY_PCT

Description Reports the percentage of the time when Adaptive Server is in a Busy state.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys SMC_NAME_ENGINE_NUM

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | DOUBLE | | | | |

SMC_NAME_CPU_PCT

| | |
|-----------------------|--|
| Description | Reports the percentage of time that a process or the set of processes running a given application was in the Running state of the time that all processes were in the Running state. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_SPID or SMC_NAME_APPLICATION_NAME |

Note SMC_NAME_SPID and SMC_NAME_APPLICATION_NAME are mutually exclusive.

Optional keys SMC_NAME_ENGINE_NUM

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | DOUBLE | | | | |

SMC_NAME_CPU_TIME

| | |
|-----------------------|---|
| Description | At server level (with no keys), reports the total CPU “busy” time on the server. When used with keys, reports on how much of that busy time was used by each process, application, or engine. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | SMC_NAME_ENGINE_NUM, SMC_NAME_SPID or SMC_NAME_APPLICATION_NAME |

Note SMC_NAME_SPID and SMC_NAME_APPLICATION_NAME are mutually exclusive.

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | DOUBLE | | | | |

SMC_NAME_CPU_YIELD

| | |
|-----------------------|---|
| Description | Reports the number of times that Adaptive Server yielded to the operating system. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |
| Required key | None |
| Optional keys | SMC_NAME_ENGINE_NUM |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_CUR_APP_NAME

| | |
|-----------------------|--|
| Description | Reports the name of the application that is executing on a particular process. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_SPID |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_CUR_ENGINE

| | |
|-------------------------------|--|
| Description | Reports the number of the Adaptive Server engine on which a process was currently running. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_SPID |
| Optional keys | None |
| Statistic types and datatypes | |

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_CUR_EXECUTION_CLASS

| | |
|-------------------------------|---|
| Description | Reports the name of the execution class under which a process is currently running. |
| Version compatibility | 11.5 and later |
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_SPID |
| Optional keys | None |
| Statistic types and datatypes | |

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_CUR_PROC_STATE

| | |
|-------------|--|
| Description | Reports the current state of a process. The possible states are: <ul style="list-style-type: none"> • None • Alarm Sleep • Background |
|-------------|--|

- Bad Status
- Infected
- Lock Sleep
- Received Sleep
- Remote I/O
- Runnable
- Running
- Send Sleep
- Sleeping
- Stopped
- Sync Sleep
- Terminating
- Yielding

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| ENUMS | | | | | |

Enum SMC_PROC_STATE

SMC_NAME_CUR_STMT_ACT_STP_DB_ID

Description Reports the database ID of the stored procedure (including triggers, a special kind of stored procedure) that contains the currently executing SQL statement for a particular process. If the currently executing SQL statement is not contained in a stored procedure, this ID is zero.

Version compatibility 11.5 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_CUR_STMT_ACT_STP_DB_NAME

Description Reports the database name of the stored procedure (including triggers, a special kind of stored procedure) that contains the currently executing SQL statement for a particular process. If the currently executing SQL statement is not contained in a stored procedure, this name is "***NoDatabase**".

Version compatibility 11.5 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_CUR_STMT_ACT_STP_ID

Description Reports the ID of the stored procedure (including triggers, a special kind of stored procedure) that contains the currently executing SQL statement for a particular process. If the currently executing SQL statement is not contained in a stored procedure, this ID is zero.

Version compatibility 11.5 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_CUR_STMT_ACT_STP_NAME

Description Reports the name of the stored procedure (including triggers, a special kind of stored procedure) that contains the currently executing SQL statement for a particular process. If the currently executing SQL statement is not contained in a stored procedure, this name is "***NoObject**".

Version compatibility 11.5 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_CUR_STMT_ACT_STP_OWNER_NAME

Description Reports the owner name of the stored procedure (including triggers, a special kind of stored procedure) that contains the currently executing SQL statement for a particular process. If the currently executing SQL statement is not contained in a stored procedure, this name is "***NoOwner**".

Version compatibility 11.5 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_CUR_STMT_ACT_STP_TEXT

Description Reports the text of a particular stored procedure (including triggers, a special kind of stored procedure) being executed for a particular process. If both CUR_STMT_ACT_STP_DB_ID is equal to 0 and CUR_STMT_ACT_STP_ID is equal to 0 then a stored procedure is not currently executing and this text is a null-terminated empty string ("").

If the text is not available (because this stored procedure was compiled and its text was discarded, or because the text is stored in an encrypted format), then this text is a null-terminated empty string ("").

Version compatibility 11.5 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_CUR_STMT_BATCH_ID

Description Reports the ID of a particular query batch being executed for a particular process.

Version compatibility 11.5 and later

Data item type Result

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and
datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_CUR_STMT_BATCH_TEXT

Description Reports the text of a particular query batch being executed for a particular process. This text can only be an initial substring of the complete text in a query batch. The maximum amount of text stored in this field is determined by the Adaptive Server configuration option max SQL text monitored and can be monitored using SMC_NAME_CUR_STMT_BATCH_TEXXT ENABLED.

Version compatibility 11.5 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and
datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_CUR_STMT_BATCH_TEXT_ENABLED

Description Reports whether Adaptive Server is saving the SQL text of the currently executing query batches, and if so, how much.

Value of 0 = saving SQL text disabled.

Value of 1 or more = maximum number of bytes of batch text per server process that can be saved.

Version compatibility 11.5 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_CUR_STMT_CONTEXT_ID

Description Reports the ID that uniquely identifies a stored procedure invocation within a particular query batch being executed for a particular process.

Version compatibility 11.5 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_CUR_STMT_CPU_TIME

Description Reports the amount of time (in seconds) that the currently executing SQL statement has spent in the running state.

Version compatibility 11.5 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes:

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | DOUBLE | | | | |

SMC_NAME_CUR_STMT_ELAPSED_TIME

Description Reports the amount of time (in seconds) that the currently executing SQL statement has been running.

Version compatibility 11.5 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_CUR_STMT_LINE_NUM

Description Reports the number of the line (within a query batch or stored procedure) that contains the beginning of the currently executing SQL statement for a particular process. The currently executing SQL statement is in the query batch if CUR_STMT_ACT_STP_DB_ID is equal to 0 and CUR_STMT_ACT_STP_ID is equal to 0. Otherwise, the currently executing SQL statement is in the stored procedure uniquely identified by these two IDs.

Version compatibility 11.5 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_CUR_STMT_LOCKS_GRANTED_IMMED

Description Reports the number of lock requests by the currently executing SQL statement that were granted immediately or were not needed (because sufficient locking was already held by the requestor).

Version compatibility 11.5 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_CUR_STMT_LOCKS_GRANTED_WAITED

Description Reports the number of lock requests by the currently executing SQL statement that were granted after waiting.

Version compatibility 11.5 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_CUR_STMT_LOCKS_NOT_GRANTED

Description Reports the number of lock requests by the currently executing SQL statement that were denied.

Version compatibility 11.5 and later

| | |
|----------------|---------------|
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_SPID |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_CUR_STMT_NUM

| | |
|-------------|--|
| Description | <p>Reports the number of the statement (appearing in a query batch or stored procedure) that is the currently executing SQL statement for a particular process. The currently executing SQL statement is in the query batch if both CUR_STMT_ACT_STP_DB_ID is equal to 0 and CUR_STMT_ACT_STP_ID is equal to 0. Otherwise, the currently executing SQL statement is in the stored procedure uniquely identified by these two IDs.</p> <p>A value of zero indicates partial data for the currently executing SQL statement (that is, this SQL statement began executing before monitoring began. Performance metrics are available but numbers reflect only the time period since the start of monitoring).</p> |
|-------------|--|

| | |
|-----------------------|----------------|
| Version compatibility | 11.5 and later |
|-----------------------|----------------|

| | |
|----------------|--------|
| Data item type | Result |
| Server level | No |
| Required keys | None |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_CUR_STMT_PAGE_IO

| | |
|-------------|---|
| Description | Reports the number of combined logical page reads and page writes accumulated by the currently executing SQL statement. |
|-------------|---|

| | |
|-----------------------|----------------|
| Version compatibility | 11.5 and later |
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_SPID |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_CUR_STMT_PAGE_LOGICAL_READ

Description Reports the number of data page reads (satisfied from cache or from device reads) accumulated by the currently executing SQL statement.

| | |
|-----------------------|----------------|
| Version compatibility | 11.5 and later |
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_SPID |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_CUR_STMT_PAGE_PHYSICAL_READ

Description Reports the number of data page reads that could not be satisfied from the data cache, accumulated by the currently executing SQL statement.

| | |
|-----------------------|----------------|
| Version compatibility | 11.5 and later |
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_SPID |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_CUR_STMT_PAGE_WRITE

Description Reports the number of data pages written to a database device, accumulated by the currently executing SQL statement.

Version compatibility 11.5 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_CUR_STMT_QUERY_PLAN_TEXT

Description Reports the text of the query plan for a particular query being executed for a particular connection.

If the text is not available (because Adaptive Server has removed this plan from its catalog of query plans), then this text is a null-terminated empty string ("").

Version compatibility 11.5 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_CUR_STMT_START_TIME

Description Reports the date and time, in the time zone of Adaptive Server, when the currently executing SQL statement began running.

If this SQL statement began running before monitoring began, then this is the date and time that activity was first encountered for this statement.

Version compatibility 11.5 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DATM | | | | | |

SMC_NAME_CUR_STMT_TEXT_BYTE_OFFSET

Description Reports the byte offset to the beginning of a statement within the query batch or stored procedure being executed for a particular process. If both CUR_STMT_ACT_STP_DB_ID is equal to 0 and CUR_STMT_ACT_STP_ID is equal to 0, then the statement is the currently executing SQL statement in the query batch. Otherwise, the statement is the currently executing SQL statement is in the stored procedure uniquely identified by these two IDs (above).

Version compatibility 11.5 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_DATA_CACHE_CONTENTION

Description Reports the fraction of the requests for a data cache's spinlock that were forced to wait (*spinlock_waits* divided by *spinlock_requests*).

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys DATA_CACHE_ID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | DOUBLE | | | | |

SMC_NAME_DATA_CACHE_EFFICIENCY

Description Reports the number of cache hits per second per megabyte of a particular data cache.

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys DATA_CACHE_ID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | DOUBLE | | | | |

SMC_NAME_DATA_CACHE_HIT

Description Reports the number of times a page read was satisfied from a particular data cache.

Version compatibility 11.0 and later

Data item type Result

| | |
|---------------|---------------|
| Server level | No |
| Required keys | DATA_CACHE_ID |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_DATA_CACHE_HIT_PCT

Description Reports the fraction of the page reads satisfied, which is computed from the following formula:

$$cache_hits / (cache_hits + cache_misses) * 100$$

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys DATA_CACHE_ID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | DOUBLE | | | | |

Note When SMC_NAME_DATA_CACHE_MISS overstates the number of physical page reads, SMC_NAME_DATA_CACHE_HIT_PCT understates the percentage of cache hits.

SMC_NAME_DATA_CACHE_ID

Description Reports the ID of a data cache. Tables or indexes or both can be bound to a specific data cache, or all objects in a database can be bound to the same data cache. No object can be bound to more than one data cache.

Version compatibility 11.0 and later

Data item type Key

Server level No

Result data items that
require this key

SMC_NAME_DATA_CACHE_CONTENTION

SMC_NAME_DATA_CACHE_EFFICIENCY

SMC_NAME_DATA_CACHE_HIT

SMC_NAME_DATA_CACHE_HIT_PCT

SMC_NAME_DATA_CACHE_LARGE_IO_DENIED

SMC_NAME_DATA_CACHE_LARGE_IO_PERFORMED

SMC_NAME_DATA_CACHE_LARGE_IO_REQUESTED

SMC_NAME_DATA_CACHE_MISS

SMC_NAME_DATA_CACHE_NAME

SMC_NAME_DATA_CACHE_PREFETCH_EFFICIENCY

SMC_NAME_DATA_CACHE_REF_AND_REUSE

SMC_NAME_DATA_CACHE_REUSE

SMC_NAME_DATA_CACHE_SIZE

Result data items for
which this key is
optional

SMC_NAME_DATA_CACHE_REUSE_DIRTY

Statistic types and
datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_DATA_CACHE_LARGE_IO_DENIED

Description Reports the number of times the Adaptive Server buffer manager did not satisfy requests (of the optimizer) to load data into a buffer in this data cache by fetching more than one contiguous page from disk at a time.

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys DATA_CACHE_ID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_DATA_CACHE_LARGE_IO_PERFORMED

Description Reports the number of times the Adaptive Server buffer manager satisfied requests (of the optimizer) to load data into a buffer in this data cache by fetching more than one contiguous page from disk at a time.

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys DATA_CACHE_ID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_DATA_CACHE_LARGE_IO_REQUESTED

Description Reports the number of times the optimizer made requests (of the Adaptive Server buffer manager) to load data into a buffer in this data cache by fetching more than one contiguous page from disk at a time.

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys DATA_CACHE_ID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_DATA_CACHE_MISS

| | |
|-----------------------|--|
| Description | Reports the number of times that a page read was satisfied from disk rather than from a particular data cache. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | No |
| Required keys | DATA_CACHE_ID |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

Note SMC_NAME_DATA_CACHE_MISS includes failed attempts to locate pages in the data caches during page allocation. Therefore, the number of physical page reads reported may be overstated. If this occurs, the percentage of data cache misses reported by SMC_NAME_DATA_CACHE_HIT_PCT is understated.

SMC_NAME_DATA_CACHE_NAME

| | |
|-----------------------|---|
| Description | Reports the name of a data cache. Tables or indexes or both can be bound to a specific data cache, or all objects in a database can be bound to the same data cache. No object can be bound to more than one cache. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | No |
| Required keys | DATA_CACHE_ID |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_DATA_CACHE_PREFETCH_EFFICIENCY

Description Reports the ratio of pages in buffers that were both referenced and reused, relative to all pages in buffers in a given cache that were reused.

If the ratio is large, then prefetching is effective; otherwise, prefetching is not providing much benefit. This may suggest that a buffer pool should be eliminated (or it may imply that a clustered index on some table is fragmented, and that the index should be dropped and recreated).

Note SMC_NAME_DATA_CACHE_PREFETCH_EFFICIENCY ignores buffers in the default buffer pool in each cache.

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys DATA_CACHE_ID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | DOUBLE | | | | |

SMC_NAME_DATA_CACHE_REUSE

Description Reports the number of pages in buffers that were reused. A large value indicates a high rate of turnover of buffers in the cache, and suggests that a pool may be too small. A zero value suggests that a buffer pool other than the default buffer pool may be too large.

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys DATA_CACHE_ID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_DATA_CACHE_REUSE_DIRTY

| | |
|-----------------------|--|
| Description | Reports the number of times that a buffer that was reused had changes that needed to be written. A non-zero value indicates that the wash size is too small. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | No |
| Required keys | DATA_CACHE_ID |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_DATA_CACHE_REF_AND_REUSE

| | |
|-----------------------|---|
| Description | Reports the number of pages in buffers that were both referenced and reused. This count is employed when determining the efficiency of prefetching buffers (see SMC_NAME_DATA_CACHE_PREFETCH_EFFICIENCY). |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | No |
| Required keys | DATA_CACHE_ID |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_DATA_CACHE_SIZE

Description Reports the size of a data cache in megabytes.

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys DATA_CACHE_ID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | | | | | |

SMC_NAME_DB_ID

Description Reports the identification number of the database.

Version compatibility 11.0 and later

Data item type Key

Server level No

Result data items that
require this key

| |
|----------------------------------|
| SMC_NAME_BLOCKING_SPID |
| SMC_NAME_DB_NAME |
| SMC_NAME_DEMAND_LOCK |
| SMC_NAME_LOCKS_BEING_BLOCKED_CNT |
| SMC_NAME_OBJ_NAME |
| SMC_NAME_OBJ_TYPE |
| SMC_NAME_OWNER_NAME |
| SMC_NAME_TIME_WAITED_ON_LOCK |

Result data items for
which this key is
optional

| |
|-----------------------------------|
| SMC_NAME_LOCKS_GRANTED_IMMED |
| SMC_NAME_LOCKS_GRANTED_WAITED |
| SMC_NAME_LOCKS_NOT_GRANTED |
| SMC_NAME_PAGE_INDEX_LOGICAL_READ |
| SMC_NAME_PAGE_INDEX_PHYSICAL_READ |
| SMC_NAME_PAGE_HIT_PCT |

SMC_NAME_PAGE_IO

SMC_NAME_PAGE_LOGICAL_READ

SMC_NAME_PAGE_PHYSICAL_READ

SMC_NAME_PAGE_WRITE

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_DB_NAME

Description Reports the name of the database.

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys DB_ID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_DEADLOCK_CNT

Description Reports the number of deadlocks.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | | | | |

SMC_NAME_DEMAND_LOCK

| | |
|-----------------------|---|
| Description | Reports the character string (Y or N) that indicates whether or not a lock has been upgraded to demand lock status. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_SPID, SMC_NAME_DB_ID, SMC_NAME_OBJ_ID, SMC_NAME_LOCK_STATUS |
| Optional keys | SMC_NAME_LOCK_TYPE, SMC_NAME_PAGE_NUM |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_DEV_HIT

| | |
|-----------------------|---|
| Description | Reports the number of times access to a device was granted. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | SMC_NAME_DEV_NAME |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_DEV_HIT_PCT

| | |
|-----------------------|---|
| Description | Reports the fraction of device requests that were granted, which is computed by dividing SMC_NAME_DEV_HIT into the result of SMC_NAME_DEV_MISS multiplied by 100. |
| Version compatibility | 11.0 and later |

Data item type Result

Server level Yes

Required keys None

Optional keys SMC_NAME_DEV_NAME

Statistic types and
datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | DOUBLE | | | | |

SMC_NAME_DEV_IO

Description Reports the total of device reads and device writes.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys SMC_NAME_DEV_NAME

Statistic types and
datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_DEV_MISS

Description Reports the number of times that access to a device had to wait.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys SMC_NAME_DEV_NAME

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_DEV_NAME

Description Reports the name of each database device.

Version compatibility 11.0 and later

Data item type Key

Server level No

Result data items that require this key None

Result data items for which this key is optional

| |
|----------------------|
| SMC_NAME_DEV_HIT |
| SMC_NAME_DEV_HIT_PCT |
| SMC_NAME_DEV_IO |
| SMC_NAME_DEV_MISS |
| SMC_NAME_DEV_READ |
| SMC_NAME_DEV_WRITE |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_DEV_READ

Description Reports the number of reads made from a database device.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys SMC_NAME_DEV_NAME

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_DEV_WRITE

| | |
|-----------------------|---|
| Description | Reports the number of writes made to a database device. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | SMC_NAME_DEV_NAME |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_ELAPSED_TIME

| | |
|-----------------------|--|
| Description | Reports the time increment, in seconds, either from one data refresh to the next (sample) or from the creation of the view to the present session. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | | | | |

SMC_NAME_ENGINE_NUM

Description Reports the number of an Adaptive Server engine.

Version compatibility 11.0 and later

Data item type Key

Server level No

Result data items that require this key None

Result data items for which this key is optional

SMC_NAME_CPU_BUSY_PCT

SMC_NAME_CPU_PCT

SMC_NAME_CPU_TIME

SMC_NAME_CPU_YIELD

SMC_NAME_PAGE_INDEX_LOGICAL_READ

SMC_NAME_PAGE_INDEX_PHYSICAL_READ

SMC_NAME_PAGE_HIT_PCT

SMC_NAME_PAGE_IO

SMC_NAME_PAGE_LOGICAL_READ

SMC_NAME_PAGE_PHYSICAL_READ

SMC_NAME_PAGE_WRITE

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_HOST_NAME

Description Reports the name of the host computer that established a particular connection to Adaptive Server.

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_KPID

Description Reports the Adaptive Server process identification number that remains unique over long periods of time.

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_LOCK_CNT

Description Reports the number of locks. This is an accumulated value.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys SMC_NAME_SPID, SMC_NAME_LOCK_TYPE,
SMC_NAME_LOCK_RESULT,
SMC_NAME_LOCK_RESULT_SUMMARY

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_LOCK_HIT_PCT

| | |
|-----------------------|--|
| Description | Reports the percentage of successful requests for locks. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | DOUBLE | | | | |

SMC_NAME_LOCK_RESULT

| | |
|--|--|
| Description | <p>Reports the result of a logical lock request. Lock result values are:</p> <ul style="list-style-type: none"> • Granted immediately. • Not needed; requestor already held a sufficient lock. • Waited; requestor waited. • Did not wait; lock was not available immediately and the requestor did not want the lock request to be queued. • Deadlock; requestor selected as deadlock victim. • Interrupted; the lock request was interrupted by attention condition. |
| Version compatibility | 11.0 and later |
| Data item type | Key |
| Server level | No |
| Result data items that require this key | None |
| Result data items for which this key is optional | SMC_NAME_LOCK_CNT |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| ENUMS | | | | | |

Enum

SMC_LOCK_RESULT

SMC_NAME_LOCK_RESULT_SUMMARY

Description

Reports the lock results summarized at a granted or not granted level.

- The lock result summary granted includes the granted, not needed, and waited lock results.
- The lock result summary not granted includes the did not wait, deadlock, and interrupted lock results.

Version compatibility

11.0 and later

Data item type

Key

Server level

No

Result data items that require this key

None

Result data items for which this key is optional

SMC_NAME_LOCK_CNT

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| ENUMS | | | | | |

Enum

SMC_LOCK_RESULT_SUMMARY

SMC_NAME_LOCK_STATUS

Description

Reports the current status of a lock. The lock status values are:

- Held and blocking
- Held and not blocking
- Requested and blocked
- Requested and not blocked

Version compatibility 11.0 and later

Data item type Key

Server level No

Result data items that require this key

| |
|----------------------------------|
| SMC_NAME_BLOCKING_SPID |
| SMC_NAME_DEMAND_LOCK |
| SMC_NAME_LOCK_STATUS_CNT |
| SMC_NAME_LOCKS_BEING_BLOCKED_CNT |
| SMC_NAME_TIME_WAITED_ON_LOCK |

Result data items for which this key is optional

None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| ENUMS | | | | | |

Enum SMC_LOCK_STATUS

SMC_NAME_LOCK_STATUS_CNT

Description Reports the number of locks in each lock status. This is a snapshot value.

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys LOCK_STATUS

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_LOCK_TYPE

Description Reports the type of lock used by Adaptive Server. Adaptive Server protects tables or data pages being used by active transactions by locking them. Adaptive Server uses the following lock types:

- Exclusive table
- Shared table
- Exclusive intent
- Shared intent
- Exclusive page
- Shared page
- Update Page

Version compatibility 11.0 and later

Data item type Key

Server level No

Result data items that require this key None

Result data items for which this key is optional

SMC_NAME_BLOCKING_SPID

SMC_NAME_DEMAND_LOCK

SMC_NAME_LOCK_CNT

SMC_NAME_LOCKS_BEING_BLOCKED_CNT

SMC_NAME_TIME_WAITED_ON_LOCK

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| ENUMS | | | | | |

Enum SMC_LOCK_TYPE

SMC_NAME_LOCKS_BEING_BLOCKED_CNT

Description Reports the number of locks being blocked by the process that holds this “hold_and_blocking” lock.

Version compatibility 11.0 and later

| | |
|----------------|---|
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_SPID, SMC_NAME_DB_ID, SMC_NAME_OBJ_ID, SMC_NAME_LOCK_STATUS |
| Optional keys | SMC_NAME_LOCK_TYPE, SMC_NAME_PAGE_NUM |

Statistic types and
datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_LOCKS_GRANTED_IMMED

| | |
|-----------------------|--|
| Description | Reports the number of locks that were granted immediately, without having to wait for another lock to be released. |
| Version compatibility | 11.5 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | SMC_NAME_SPID, SMC_NAME_APPLICATION_NAME, [SMC_NAME_DB_ID + SMC_NAME_OBJ_ID], [SMC_NAME_CUR_STMT_ACT_STP_DB_ID + SMC_NAME_CUR_STMT_ACT_STP_ID], [SMC_NAME_ACT_STP_DB_ID + SMC_NAME_ACT_STP_ID] |

Note SMC_NAME_SPID and SMC_NAME_APPLICATION_NAME are mutually exclusive. If you use the SMC_NAME_CUR_STMT_ACT_STP_DB_ID + SMC_NAME_CUR_STMT_ACT_STP_ID key combination, you cannot use any other keys.

Statistic types and
datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_LOCKS_GRANTED_WAITED

| | |
|-----------------------|--|
| Description | Reports the number of locks that were granted after waiting for another lock to be released. |
| Version compatibility | 11.5 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | SMC_NAME_SPID, SMC_NAME_APPLICATION_NAME, [SMC_NAME_DB_ID + SMC_NAME_OBJ_ID], [SMC_NAME_CUR_STMT_ACT_STP_DB_ID + SMC_NAME_CUR_STMT_ACT_STP_ID], [SMC_NAME_ACT_STP_DB_ID + SMC_NAME_ACT_STP_ID] |

Note SMC_NAME_SPID and SMC_NAME_APPLICATION_NAME are mutually exclusive. If you use the SMC_NAME_CUR_STMT_ACT_STP_DB_ID + SMC_NAME_CUR_STMT_ACT_STP_ID key combination, you cannot use any other keys.

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_LOCKS_NOT_GRANTED

| | |
|-----------------------|--|
| Description | Reports the number of locks that were requested but not granted. |
| Version compatibility | 11.5 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |

Optional keys SMC_NAME_SPID, SMC_NAME_APPLICATION_NAME,
[SMC_NAME_DB_ID + SMC_NAME_OBJ_ID],
[SMC_NAME_CUR_STMT_ACT_STP_DB_ID +
SMC_NAME_CUR_STMT_ACT_STP_ID],
[SMC_NAME_ACT_STP_DB_ID + SMC_NAME_ACT_STP_ID]

Note SMC_NAME_SPID and SMC_NAME_APPLICATION_NAME are mutually exclusive. If you use the SMC_NAME_CUR_STMT_ACT_STP_DB_ID + SMC_NAME_CUR_STMT_ACT_STP_ID key combination, you cannot use any other keys.

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_LOG_CONTENTION_PCT

Description Reports the percentage of times, of the total times when a user log cache was flushed into the transaction log, that it had to wait for the log semaphore.

A high percentage may indicate that the user log cache size should be increased.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | DOUBLE | | | | |

SMC_NAME_LOGIN_NAME

Description Reports the login name associated with Adaptive Server processes.

| | |
|-----------------------|----------------|
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_SPID |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_MEM_CODE_SIZE

| | |
|-----------------------|--|
| Description | Reports the amount of memory in bytes allocated for Adaptive Server. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_MEM_KERNEL_STRUCT_SIZE

| | |
|-----------------------|--|
| Description | Reports the amount of memory in bytes allocated for the kernel structures. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_MEM_PAGE_CACHE_SIZE

Description Reports the amount of memory in bytes allocated for the page cache.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_MEM_PROC_BUFFER

Description Reports the amount of memory in bytes allocated for procedure buffers.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_MEM_PROC_HEADER

Description Reports the amount of memory in bytes allocated for procedure headers.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_MEM_SERVER_STRUCT_SIZE

Description Reports the amount of memory in bytes allocated for the Adaptive Server structures.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_MOST_ACT_DEV_IO

Description Reports the number of combined reads and writes against the device with the most activity during a given time interval.

Version compatibility 11.0 and later

Server level Yes

Data item type Result

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_MOST_ACT_DEV_NAME

Description Reports the name of the device with the largest number of combined reads and writes during a given time interval.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | CHARP | | | | |

SMC_NAME_NET_BYTE_IO

Description Reports the number of combined network bytes sent and received.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_NET_BYTES_RCVD

| | |
|-----------------------|---|
| Description | Reports the number of network bytes received. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_NET_BYTES_SENT

| | |
|-----------------------|---|
| Description | Reports the number of network bytes sent. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_NET_DEFAULT_PKT_SIZE

| | |
|---------------|---|
| Description | Reports the default size of a network packet. |
| Type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_NET_MAX_PKT_SIZE

Description Reports the maximum size configured for a network packet.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_NET_PKT_SIZE_RCVD

Description Reports the average size of network packets received.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | DOUBLE | | | | |

SMC_NAME_NET_PKT_SIZE_SENT

Description Reports the average size of network packets sent.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | DOUBLE | | | | |

SMC_NAME_NET_PKTS_RCVD

Description Reports the number of network packets received.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_NET_PKTS_SENT

Description Reports the number of network packets sent.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_NUM_ENGINES

Description Reports the number of engines running on Adaptive Server.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_NUM_PROCESSES

Description Reports the number of processes currently running on Adaptive Server, or, if used with the key SMC_NAME_APPLICATION_NAME, the number of processes currently running a given application.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys SMC_NAME_APPLICATION_NAME

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_OBJ_ID

Description Reports the identification number of a database object where the object returned is either a table or a stored procedure.

Version compatibility 11.0 and later

Data item type Key

Server level No

Required keys SMC_NAME_DB_ID

Result data items that require this key

| |
|----------------------------------|
| SMC_NAME_BLOCKING_SPID |
| SMC_NAME_DEMAND_LOCK |
| SMC_NAME_LOCKS_BEING_BLOCKED_CNT |
| SMC_NAME_OBJ_NAME |
| SMC_NAME_OBJ_TYPE |
| SMC_NAME_OWNER_NAME |
| SMC_NAME_TIME_WAITED_ON_LOCK |

Result data items for which this key is optional

| |
|-----------------------------------|
| SMC_NAME_LOCKS_GRANTED_IMMED |
| SMC_NAME_LOCKS_GRANTED_WAITED |
| SMC_NAME_LOCKS_NOT_GRANTED |
| SMC_NAME_PAGE_INDEX_LOGICAL_READ |
| SMC_NAME_PAGE_INDEX_PHYSICAL_READ |
| SMC_NAME_PAGE_HIT_PCT |
| SMC_NAME_PAGE_IO |
| SMC_NAME_PAGE_LOGICAL_READ |
| SMC_NAME_PAGE_PHYSICAL_READ |
| SMC_NAME_PAGE_WRITE |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

If you create a view using the SMC_NAME_OBJ_ID data item, you might see negative numbers as object IDs. Negative object IDs are an accurate reporting of IDs as assigned by Adaptive Server.

Monitor Server reports on *all* activity, including activity on temporary tables that Adaptive Server creates to perform a complex query. The object IDs that Adaptive Server assigns to temporary tables can be positive or negative. The object ID that was assigned by Adaptive Server is reported.

SMC_NAME_OBJ_NAME

| | |
|-----------------------|---|
| Description | Reports the name of a database object. In views that show SMC_NAME_OBJ_NAME, the string **TempObject** is reported for temporary tables. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_DB_ID, SMC_NAME_OBJ_ID |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_OBJ_TYPE

| | |
|-----------------------|--|
| Description | Reports the type of database object, table, or stored procedure. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_DB_ID, SMC_NAME_OBJ_ID |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| ENUMS | | | | | |

Enum SMC_OBJ_TYPE

SMC_NAME_OWNER_NAME

Description Reports the owner name of the database object.

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys SMC_NAME_DB_ID, SMC_NAME_OBJ_ID

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_PAGE_HIT_PCT

Description Reports the percentage of times that a data page read could be satisfied from cache without requiring a physical page read.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys SMC_NAME_SPID,
[SMC_NAME_DB_ID + SMC_NAME_OBJ_ID],
[SMC_NAME_ACT_STP_DB_ID + SMC_NAME_ACT_STP_ID],
SMC_NAME_ENGINE_NUM

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | DOUBLE | | | | |

SMC_NAME_PAGE_INDEX_LOGICAL_READ

Description Reports the number of index page reads satisfied from cache or from device reads.

Version compatibility 11.0 and later

| | |
|----------------|---|
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | SMC_NAME_SPID, SMC_NAME_APPLICATION_NAME, SMC_NAME_DB_ID, SMC_NAME_OBJ_ID, SMC_NAME_ENGINE_NUM, [SMC_NAME_ACT_STP_DB_ID + SMC_NAME_ACT_STP_ID] |

Note SMC_NAME_SPID and SMC_NAME_APPLICATION_NAME are mutually exclusive.

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_PAGE_INDEX_PHYSICAL_READ

| | |
|-----------------------|---|
| Description | Reports the number of index page reads that could not be satisfied from the data cache. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | No |
| Required keys | None |
| Optional keys | SMC_NAME_SPID, SMC_NAME_APPLICATION_NAME, SMC_NAME_DB_ID, SMC_NAME_OBJ_ID, SMC_NAME_ENGINE_NUM, [SMC_NAME_ACT_STP_DB_ID + SMC_NAME_ACT_STP_ID] |

Note SMC_NAME_SPID and SMC_NAME_APPLICATION_NAME are mutually exclusive.

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_PAGE_IO

| | |
|-----------------------|--|
| Description | Reports the number of combined logical page reads and page writes. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | SMC_NAME_SPID, SMC_NAME_APPLICATION_NAME, [SMC_NAME_DB_ID + SMC_NAME_OBJ_ID], [SMC_NAME_ACT_STP_DB_ID + SMC_NAME_ACT_STP_ID], SMC_NAME_ENGINE_NUM |

Note SMC_NAME_SPID and SMC_NAME_APPLICATION_NAME are mutually exclusive.

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_PAGE_LOGICAL_READ

| | |
|-----------------------|---|
| Description | Reports the number of data page reads, whether satisfied from cache or from a database device. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | SMC_NAME_SPID, SMC_NAME_APPLICATION_NAME, SMC_NAME_DB_ID, SMC_NAME_OBJ_ID, SMC_NAME_ENGINE_NUM, [SMC_NAME_ACT_STP_DB_ID + SMC_NAME_ACT_STP_ID] |

Note SMC_NAME_SPID and SMC_NAME_APPLICATION_NAME are mutually exclusive.

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_PAGE_NUM

Description Reports the number of the data page for a given lock or lock request.

Version compatibility 11.0 and later

Data item type Key

Server level No

Result data items that require this key None

Result data items for which this key is optional

SMC_NAME_BLOCKING_SPID

SMC_NAME_DEMAND_LOCK

SMC_NAME_LOCKS_BEING_BLOCKED_CNT

SMC_NAME_TIME_WAITED_ON_LOCK

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_PAGE_PHYSICAL_READ

Description Reports the number of data page reads that could not be satisfied from the data cache.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys SMC_NAME_SPID, SMC_NAME_APPLICATION_NAME,
SMC_NAME_DB_ID, SMC_NAME_OBJ_ID,
SMC_NAME_ENGINE_NUM, [SMC_NAME_ACT_STP_DB_ID +
SMC_NAME_ACT_STP_ID]

Note SMC_NAME_SPID and SMC_NAME_APPLICATION_NAME are mutually exclusive.

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_PAGE_WRITE

Description Reports the number of data pages written to a database device.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys SMC_NAME_SPID, SMC_NAME_APPLICATION_NAME,
SMC_NAME_DB_ID, SMC_NAME_OBJ_ID,
SMC_NAME_ENGINE_NUM, [SMC_NAME_ACT_STP_DB_ID +
SMC_NAME_ACT_STP_ID]

Note SMC_NAME_SPID and SMC_NAME_APPLICATION_NAME are mutually exclusive.

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_PROC_STATE

Description Reports the state of a process. The possible states are:

- None
- Alarm Sleep
- Background
- Bad Status
- Infected
- Lock Sleep
- Received Sleep
- Remote IO
- Runnable
- Running
- Send Sleep
- Sleeping
- Stopped
- Sync Sleep
- Terminating
- Yielding

Version compatibility 11.0 and later

Data item type Key

Server level No

Result data items that require this key SMC_NAME_PROC_STATE_CNT

Result data items for which this key is optional None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| ENUMS | | | | | |

Enum SMC_PROC_STATE

SMC_NAME_PROC_STATE_CNT

Description Reports the number of processes in a particular state.

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys SMC_NAME_PROC_STATE

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_SPID

Description Reports the process identification number. Views that contain SMC_NAME_SPID only report on processes that are active as of the end of the sample period. SMC_NAME_SPID is mutually exclusive with SMC_NAME_APPLICATION_NAME in a view.

Version compatibility 11.0 and later

Data item type Key

Server level No

Result data items that require this key

| |
|--------------------------------------|
| SMC_NAME_BLOCKING_SPID |
| SMC_NAME_CONNECT_TIME |
| SMC_NAME_CPU_PCT |
| SMC_NAME_CPU_TIME |
| SMC_NAME_CUR_APP_NAME |
| SMC_NAME_CUR_ENGINE |
| SMC_NAME_CUR_EXECUTION_CLASS |
| SMC_NAME_CUR_PROC_STATE |
| SMC_NAME_CUR_STMT_ACT_STP_DB_NAME |
| SMC_NAME_CUR_STMT_ACT_STP_NAME |
| SMC_NAME_CUR_STMT_ACT_STP_OWNER_NAME |
| SMC_NAME_CUR_STMT_ACT_STP_TEXT |
| SMC_NAME_CUR_STMT_BATCH_TEXT |

SMC_NAME_CUR_STMT_CPU_TIME
 SMC_NAME_CUR_STMT_ELAPSED_TIME
 SMC_NAME_CUR_STMT_LINE_NUM
 SMC_NAME_CUR_STMT_LOCKS_GRANTED_IMMED
 SMC_NAME_CUR_STMT_LOCKS_GRANTED_WAITED
 SMC_NAME_CUR_STMT_LOCKS_NOT_GRANTED
 SMC_NAME_CUR_STMT_PAGE_IO_CNT
 SMC_NAME_CUR_STMT_PAGE_CACHE_READ_CNT
 SMC_NAME_CUR_STMT_PAGE_PHYSICAL_READ_CNT
 SMC_NAME_CUR_STMT_PAGE_WRITE_CNT
 SMC_NAME_CUR_STMT_QUERY_PLAN_TEXT
 SMC_NAME_CUR_STMT_START_TIME
 SMC_NAME_CUR_STMT_TEXT_BYTE_OFFSET
 SMC_NAME_DEMAND_LOCK
 SMC_NAME_HOST_NAME
 SMC_NAME_KPID
 SMC_NAME_LOCKS_BEING_BLOCKED_CNT
 SMC_NAME_LOGIN_NAME
 SMC_NAME_TIME_WAITED_ON_LOCK

Result data items for
 which this key is
 optional

SMC_NAME_LOCK_CNT
 SMC_NAME_LOCKS_GRANTED_IMMED
 SMC_NAME_LOCKS_GRANTED_WAITED
 SMC_NAME_LOCKS_NOT_GRANTED
 SMC_NAME_PAGE_INDEX_LOGICAL_READ
 SMC_NAME_PAGE_INDEX_PHYSICAL_READ
 SMC_NAME_PAGE_LOGICAL_READ
 SMC_NAME_PAGE_PHYSICAL_READ
 SMC_NAME_PAGE_WRITE
 SMC_NAME_STP_CPU_TIME
 SMC_NAME_STP_NUM_TIMES_EXECUTED

Statistic types and
 datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_SQL_SERVER_NAME

Description Reports the name of the Adaptive Server that is being monitored as specified in the -s parameter to the start-up command of the Monitor Server to which the application is connected.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_SQL_SERVER_VERSION

Description Reports the version of the Adaptive Server that is being monitored. For more information, refer to the global @@version variable in the *Transact-SQL User's Guide*.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_STP_CPU_TIME

Description Reports the CPU time, in seconds, spent executing a stored procedure.

Version compatibility 11.0 and later

| | |
|----------------|--|
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_ACT_STP_DB_ID, SMC_NAME_ACT_STP_ID |
| Optional keys | SMC_NAME_SPID, SMC_NAME_STP_STMT_NUM, SMC_NAME_STP_LINE_NUM |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | DOUBLE | | | DOUBLE | DOUBLE |

SMC_NAME_STP_ELAPSED_TIME

| | |
|-----------------------|---|
| Description | Reports the time, in seconds, spent executing a stored procedure. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_ACT_STP_DB_ID, SMC_NAME_ACT_STP_ID |
| Optional keys | SMC_NAME_STP_STMT_NUM, SMC_NAME_STP_LINE_NUM |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | DOUBLE | | | DOUBLE | DOUBLE |

SMC_NAME_STP_EXECUTION_CLASS

| | |
|-----------------------|---|
| Description | Reports the configured execution class, if any, for a given stored procedure. |
| Version compatibility | 11.5 and later |
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_ACT_STP_DB_ID, SMC_NAME_ACT_STP_ID |
| Optional keys | SMC_NAME_STP_STMT_NUM, SMC_NAME_STP_LINE_NUM |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_STP_HIT_PCT

| | |
|-----------------------|--|
| Description | Reports the percentage of times that a stored procedure execution found the procedure's query plan in procedure cache and available for use. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | DOUBLE | | | | |

SMC_NAME_STP_LINE_NUM

| | |
|---|---|
| Description | Reports the stored procedure line number. |
| Version compatibility | 11.0 and later |
| Data item type | Key |
| Server level | No |
| Result data items that require this key | None |

Result data items for which this key is optional

SMC_NAME_STP_CPU_TIME

SMC_NAME_STP_ELAPSED_TIME

SMC_NAME_STP_NUM_TIMES_EXECUTED

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_STP_LINE_TEXT

| | |
|-----------------------|--|
| Description | Reports the entire text of the stored procedure. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | No |
| Required keys | SMC_NAME_ACT_STP_DB_ID, SMC_NAME_ACT_STP_ID |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_STP_LOGICAL_READ

| | |
|-----------------------|---|
| Description | Reports the number of requests to execute a stored procedure, whether satisfied from procedure cache or with a read from <i>sysprocedures</i> . |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_STP_NUM_TIMES_EXECUTED

| | |
|-----------------------|--|
| Description | Reports the number of times a stored procedure, or a line in a stored procedure, was executed. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | No |

Required keys SMC_NAME_ACT_STP_DB_ID, SMC_NAME_ACT_STP_ID

Optional keys SMC_NAME_SPID, SMC_NAME_STP_STMT_NUM,
SMC_NAME_STP_LINE_NUM

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_STP_PHYSICAL_READ

Description Reports the number of requests to execute a stored procedure for which a read from *sysprocedures* was necessary.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_STP_STMT_NUM

Description Reports the number within a stored procedure. A single stored procedure line may contain one or more statements.

Version compatibility 11.0 and later

Data item type Key

Server level No

Result data items that require this key None

Result data items for which this key is optional

SMC_NAME_STP_CPU_TIME

SMC_NAME_STP_ELAPSED_TIME

SMC_NAME_STP_NUM_TIMES_EXECUTED

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_THREAD_EXCEEDED_MAX

Description Reports the number of times a query plan was runtime-adjusted because of attempting to exceed the configured limit of threads in the server-wide worker thread pool in Adaptive Server.

Version compatibility 11.5 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_THREAD_EXCEEDED_MAX_PCT

Description Reports the percentage of time a query plan was adjusted at runtime because it tried to exceed the configured limit of threads in the server-wide worker thread pool in Adaptive Server.

Version compatibility 11.5 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DOUBLE | DOUBLE | | | | |

SMC_NAME_THREAD_MAX_USED

Description Reports the maximum number of threads from the server-wide worker thread pool that were concurrently in use on the server.

Version compatibility 11.5 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_TIME_WAITED_ON_LOCK

Description Reports the amount of time (in seconds) waited for a lock request to be granted.

Version compatibility 11.0 and later

Data item type Result

Server level No

Required keys SMC_NAME_SPID, SMC_NAME_DB_ID, SMC_NAME_OBJ_ID,
SMC_NAME_LOCK_STATUS

Optional keys SMC_NAME_LOCK_TYPE, SMC_NAME_PAGE_NUM

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | | | | | |

SMC_NAME_TIMESTAMP

Description Reports the date and time on Adaptive Server in its time zone. For more information, refer to the `getdate()` function in the *Transact-SQL User's Guide*.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| CHARP | | | | | |

SMC_NAME_TIMESTAMP_DATIM

Description Reports the date and time on Adaptive Server in its time zone, returned in a CS_DATETIME struct. For more information, refer to the `getdate()` function in the *Transact-SQL User's Guide*.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| DATIM | | | | | |

SMC_NAME_XACT

Description Reports the number of committed Transact-SQL statement blocks (transactions).

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_XACT_DELETE

Description Reports the number of rows deleted from database tables.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_XACT_DELETE_DEFERRED

Description Reports the number of rows deleted from a database table that were done in deferred mode.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_XACT_DELETE_DIRECT

| | |
|-----------------------|---|
| Description | Reports the number of rows deleted from a database table that were done in direct mode. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_XACT_INSERT

| | |
|-----------------------|---|
| Description | Reports the number of insertions into a database table. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_XACT_INSERT_CLUSTERED

| | |
|-----------------------|--|
| Description | Reports the number of insertions to database tables that have a clustered index. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |

Required keys None

Optional keys None

Statistic types and
datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_XACT_INSERT_HEAP

Description Reports the number of insertions to database tables that do not have a clustered index.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and
datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_XACT_SELECT

Description Reports the number of SELECT or OPEN CURSOR statements.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and
datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_XACT_UPDATE

| | |
|-----------------------|---|
| Description | Reports the updates to database tables. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_XACT_UPDATE_DEFERRED

| | |
|-----------------------|---|
| Description | Reports the updates to a database table that are performed in deferred mode rather than in direct mode. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |
| Required keys | None |
| Optional keys | None |

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_XACT_UPDATE_DIRECT

| | |
|-----------------------|--|
| Description | Reports the sum of expensive, in-place, and not-in-place updates (everything except updates deferred). Also called updates in place. |
| Version compatibility | 11.0 and later |
| Data item type | Result |
| Server level | Yes |

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_XACT_UPDATE_EXPENSIVE

Description Reports the updates to a database table that are done in expensive mode. In expensive mode, a row is deleted from its original location, and inserted at a new location.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_XACT_UPDATE_IN_PLACE

Description Reports the updates that do not require a delete and insert.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

SMC_NAME_XACT_UPDATE_NOT_IN_PLACE

Description Reports the updates that require a delete and insert.

Version compatibility 11.0 and later

Data item type Result

Server level Yes

Required keys None

Optional keys None

Statistic types and
datatypes

| VALUE_ SAMPLE | VALUE_ SESSION | RATE_ SAMPLE | RATE_ SESSION | AVG_ SAMPLE | AVG_ SESSION |
|------------------|-------------------|-----------------|------------------|----------------|-----------------|
| LONG | LONG | DOUBLE | DOUBLE | | |

Monitor Client Library Functions

In addition to giving you detailed information about Monitor Client Library functions, this chapter contains the following topics:

| Topic | Page |
|----------------|------|
| Threads | 126 |
| Error handling | 127 |

You use Monitor Client Library functions to write applications that collect Adaptive Server performance data. This chapter describes, in alphabetical order, each Monitor Client Library function. Table 3-1 lists the functions and a brief description of each.

Table 3-1: Monitor Client Library functions

| Function | Description |
|------------------------------|---|
| smc_close | Closes a connection |
| smc_connect_alloc | Creates a connection structure |
| smc_connect_drop | Deallocates a connection structure |
| smc_connect_ex | Establishes a connection |
| smc_connect_props | Sets, retrieves, or clears properties on a connection |
| smc_create_alarm_ex | Adds an alarm to a data item |
| smc_create_filter | Adds a filter to a data item |
| smc_create_playback_session | Initializes a playback session on a Historical Server connection |
| smc_create_recording_session | Initializes a recording session on a Historical Server connection |
| smc_create_view | Defines a view |
| smc_drop_alarm | Removes an alarm from a data item in a view |
| smc_drop_filter | Removes a filter from a data item in a view |
| smc_drop_view | Drop a views |
| smc_get_command_info | Retrieves detailed information about an alarm or error |
| smc_get_dataitem_type | Retrieves the type of a data item |

| Function | Description |
|--------------------------------------|---|
| <code>smc_get_dataitem_value</code> | Retrieves the data for a particular data item and row |
| <code>smc_get_row_count</code> | Retrieves the number of rows of data in a view |
| <code>smc_get_version_string</code> | Retrieves the Monitor Client Library version number |
| <code>smc_initiate_playback</code> | Concludes the definition of views for a playback session |
| <code>smc_initiate_recording</code> | Concludes the definition of views for a recording session |
| <code>smc_refresh_ex</code> | Retrieves data for all views in a given connection |
| <code>smc_terminate_playback</code> | Ends a playback session on a Historical Server connection |
| <code>smc_terminate_recording</code> | Cancels a recording session on a Historical Server connection |

Most functions work with Monitor Server and Historical Server. In this chapter, unless otherwise noted, the term connection means a connection to Monitor Server or Historical Server. See Appendix C, “Backward Compatibility” for information about obsolete functions.

Threads

Two threads cannot use Monitor Client Library functions at the same time. Use a global lock (semaphore) on Monitor Client Library calls to avoid any thread overwrites or unpredictable actions.

Monitor Client Library functions are not protected from reentrant invocation. Use the following special programming considerations when using these functions in a multithreaded environment. Be sure that:

- A call to create a client connection (`smc_connect`) is serialized with all other Monitor Client Library function calls across all threads.
- A call to disconnect a client connection (`smc_disconnect`) is serialized with all other Monitor Client Library function calls across all threads.

- Any single client connection lives in one, and only one, thread. All Monitor Client Library function calls to access this client connection occur in this thread.
- A call to refresh a client connection is serialized with all other Monitor Client Library function calls on this connection in this thread.

Error handling

A Monitor Client Library application installs an error handler when it creates a connection (`smc_connect_alloc`, described on 131). This error handler is called whenever an error occurs for that connection.

Most Monitor Client Library functions return one of the following values:

Table 3-2: Return values

| Return value | Description |
|-------------------------|--|
| SMC_RET_SUCCESS | The function completed successfully. |
| SMC_RET_FAILURE | The function failed. More detailed information is available from the error handler. |
| SMC_RET_INVALID_CONNECT | The function did not execute because it was requested against an erroneous connection. The error handler is not invoked because error handlers are available only for valid connections. |

Other return values are listed with the functions that return them.

Note The error callback function is not triggered under certain error conditions regarding data item specification in `smc_create_view` and `smc_create_alarm`. To capture these error conditions, check the return code for these functions.

Error handler

Description

An error handler is a user-defined function.

| | |
|------------|--|
| Syntax | <code>SMC_VOID errorCallback (</code> <code>SMC_CONNECT_ID clientId,</code> <code>SMC_COMMAND_ID commandId,</code> <code>SMC_VOIDP userDataHandle)</code> |
| Parameters | <i>clientId</i> identifies a Monitor connection. <i>commandId</i> identifies an instance of a command. <i>userDataHandle</i> user-supplied pointer. |
| Usage | <ul style="list-style-type: none">• An error handler can be changed at any time via either <code>smc_change_error_handler</code> or <code>smc_connect_props</code> functions. (See Callback function on page 128 for more information) |

Note C++ member functions cannot be used as callback functions.

Callback function

| | |
|-------------|---|
| Description | Callback functions are user-defined functions that notify an application when an event has occurred. These functions are registered with Monitor Client Library API calls for: <ul style="list-style-type: none">• Alarms• Error information When either of the above events occur, a callback function is executed. |
| Syntax | <code>SMC_VOID CallbackFunction</code> <code>(SMC_CONNECT_ID clientId,</code> <code>SMC_COMMAND_ID commandId,</code> <code>SMC_VOIDP userDataHandle)</code> |
| Parameters | <i>clientId</i> identifies the connection. <i>commandId</i> identifies the instance of a command. |

userDataHandle

user data pointer for a given connection. An application can set this pointer by using `smc_connect_props`.

Usage

Accessing callback data When an event triggers a callback function, you can request information about the event. Data is accessed by calling `smc_get_command_info` from within the callback function. This function takes a connection ID, a command ID, and an enumerator constant that identifies which piece of data the user is interested in. The data available depends on the type of callback. Table 3-3 describes the data available for alarm callbacks. Table 3-4 describes the data available for error callbacks.

Table 3-3: Data available for alarm callbacks

| Information type | Description |
|--------------------------------|---|
| SMC_INFO_ALARM_ACTION_DATA | String supplied for <i>alarmActionData</i> upon creation of the alarm. |
| SMC_INFO_ALARM_ALARMID | Identifies the alarm. |
| SMC_INFO_ALARM_CURRENT_VALUE | Current value that met or exceeded the alarm threshold. |
| SMC_INFO_ALARM_DATAITEM | Data item on which the alarm was set. Points to a <code>SMC_DATAITEM_STRUCT</code> . |
| SMC_INFO_ALARM_ROW | Row containing the data item value that triggered the alarm. |
| SMC_INFO_ALARM_THRESHOLD_VALUE | Threshold value defined for this alarm. |
| SMC_INFO_ALARM_TIMESTAMP | Time (in the Adaptive Server time zone) marking the end of the sample interval in whose data the alarm condition was met. |
| SMC_INFO_ALARM_VIEWID | Identifies a view created on the connection. |

Table 3-4: Data available for error callbacks

| Information type | Description |
|--------------------------|--|
| SMC_INFO_ERR_MAPSEVERITY | Monitor Client Library severity level. |
| SMC_INFO_ERR_MSG | Text of the error message. (See Appendix D, “Troubleshooting Information and Error Messages”.) |
| SMC_INFO_ERR_NUM | Number of the error. |
| SMC_INFO_ERR_SEVERITY | Severity of the error message. |

| Information type | Description |
|---------------------|--|
| SMC_INFO_ERR_SOURCE | Source of the error message. One of the following: <ul style="list-style-type: none">• SMC_SRC_UNKNOWN – not known• SMC_SRC_HS – Historical Server• SMC_SRC_SMC – Monitor Client Library• SMC_SRC_CT – Client Library• SMC_SRC_SS – Adaptive Server• SMC_SRC_SMS – Monitor Server |
| SMC_INFO_ERR_STATE | State of the error. Useful for technical support in diagnosing internal errors. |

smc_close

Description Closes a connection that was created with `smc_connect_ex`. This function terminates the connection but does not deallocate it. Use `smc_connect_drop` to deallocate a connection structure.

Syntax `SMC_RETURN_CODE smc_close
(SMC_CONNECT_ID clientId,
SMC_CLOSE_TYPE closeType)`

Parameters *clientId*
identifies the connection.

closeType
type of close: SMC_CLOSE_REQUEST

Return value

| Return value | Indicates |
|-------------------------|----------------------------|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |
| SMC_RET_INVALID_CONNECT | Connection does not exist. |

Examples This example assumes that you have created a connection and have a *clientId*.

```
if (smc_close(clientId, SMC_CLOSE_REQUEST)
    != SMC_RET_SUCCESS)
{
    printf("smc_close failed\n");
    /* do some cleanup */
}
```

```
}

```

Usage

- All views (as well as alarms and filters associated with the data items in the view) on the specified connection are also dropped.
- `smc_close` only disconnects a connection. Call `smc_connect_drop` to deallocate a connection structure.
- If `smc_close` returns a failure, the user is advised to call `smc_connect_drop`.

Valid server modes

| Mode | Availability |
|-------------------------|--------------|
| SMC_SERVER_M_LIVE | Yes |
| SMC_SERVER_M_HISTORICAL | Yes |

Errors

| Error | Indicates |
|-----------------------------------|--|
| SMC_RET_INTERNAL_ERROR | Internal error |
| SMC_RET_INVALID_API_FUNCTION | Invalid use of obsolete and replacement functions in the same connection |
| SMC_RET_INVALID_API_FUNC_SEQUENCE | Invalid calling sequence of Monitor Client Library functions |

See also

`smc_connect_drop`, `smc_connect_ex`

smc_connect_alloc

Description

Creates a connection structure with error callback, but does not establish a connection.

Syntax

```
SMC_RETURN_CODE smc_connect_alloc
(SMC_GEN_CALLBACK  ErrCallback,
 SMC_CONNECT_IDP   clientIdHandle)
```

Parameters

ErrCallback

Pointer to error callback function.

clientIdHandle

Pointer to a variable, which should be declared as type `SMC_CONNECT_ID`. If the call to `smc_connect` succeeds, this variable contains the ID for the Monitor connection.

Return value

| Return value | Indicates |
|-----------------|---------------------|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |

Examples

The following example assumes you have defined an error callback function, *myErrorHandler*.

```
SMC_CONNECT_ID  clientId;
if (smc_connect_alloc(myErrorHandler, &clientId)
    != SMC_RET_SUCCESS)
{
    printf("smc_connect_alloc failed\n");
    exit(1);
}
```

Usage

- The error handler parameter cannot be null.
- Use `smc_connect_props` to set properties on a connection.
- Use `smc_connect_ex` to establish the connection identified by *clientIdHandle*.
- Use `smc_connect_drop` to deallocate a connection structure created with `smc_connect_alloc`.

Valid server modes

| Mode | Availability |
|-------------------------|--------------|
| SMC_SERVER_M_LIVE | Yes |
| SMC_SERVER_M_HISTORICAL | Yes |

Errors

| Error | Indicates |
|-----------------------------|---------------------|
| SMC_RET_INSUFFICIENT_MEMORY | Insufficient memory |
| SMC_RET_INTERNAL_ERROR | Internal error |

See also

`smc_connect_drop`, `smc_connect_ex`, `smc_connect_props`

smc_connect_drop

Description

Deallocates a connection structure that was created with `smc_connect_alloc`.

Syntax

```
SMC_RETURN_CODE smc_connect_drop
(SMC_CONNECT_ID clientId)
```

Parameters *clientId*
identifies the connection.

Return value

| Return value | Indicates |
|-------------------------|----------------------------|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |
| SMC_RET_INVALID_CONNECT | Connection does not exist. |

Examples

This example assumes that:

- You have created a connection using `smc_connect_alloc` and have a *clientId*.
- You have successfully executed `smc_close` on the connection.

```
if (smc_connect_drop(clientId) != SMC_RET_SUCCESS) {
    printf("smc_connect_drop failed\n");
    /* do some cleanup */
}
```

Usage

- `smc_close` must be called before `smc_connect_drop`, if a connection was successfully made.

Valid server modes

| Mode | Availability |
|-------------------------|--------------|
| SMC_SERVER_M_LIVE | Yes |
| SMC_SERVER_M_HISTORICAL | Yes |

Errors

| Error | Indicates |
|-----------------------------------|--|
| SMC_RET_CONNECT_NOT_CLOSED | Connection has not been closed |
| SMC_RET_INVALID_API_FUNCTION | Invalid use of obsolete and replacement functions on the same connection |
| SMC_RET_INVALID_API_FUNC_SEQUENCE | Invalid calling sequence of Monitor Client Library functions |

See also

`smc_close`, `smc_connect_alloc`

smc_connect_ex

Description Establishes a connection for the connection structure created with `smc_connect_alloc`. Properties on the connection, such as Server Name and Server Mode, must have been set with `smc_connect_props`.

Syntax `SMC_RETURN_CODE smc_connect_ex
(SMC_CONNECT_ID clientId)`

Parameters *clientId*
identifies the connection.

Return value

| Return value | Indicates |
|-------------------------|----------------------------|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |
| SMC_RET_INVALID_CONNECT | Connection does not exist. |

Examples This example assumes you have created a connection using `smc_connect_alloc` and have a *clientId*.

```
if (smc_connect_ex(clientId) != SMC_RET_SUCCESS)
{
    printf("smc_connect_ex failed\n");
    exit(1);
}
```

Usage

- `smc_connect_alloc` and `smc_connect_props` must be called before `smc_connect_ex`.
- Each Monitor Client Library connection uses two network connections. If you are running a Monitor Client Library application on a PC and reach the limit on network connections, reconfigure your networking software to raise the limit.

Valid server modes

| Mode | Availability |
|-------------------------|--------------|
| SMC_SERVER_M_LIVE | Yes |
| SMC_SERVER_M_HISTORICAL | Yes |

Errors

| Error | Indicates |
|-----------------------------|---------------------|
| SMC_RET_INSUFFICIENT_MEMORY | Insufficient memory |
| SMC_RET_INTERNAL_ERROR | Internal error |

| Error | Indicates |
|-----------------------------------|--|
| SMC_RET_INVALID_API_FUNCTION | Invalid use of obsolete and replacement functions on the same connection |
| SMC_RET_INVALID_API_FUNC_SEQUENCE | Invalid calling sequence of Monitor Client Library functions |
| SMC_RET_INVALID_PROPERTY | Property has not been set |
| SMC_RET_UNABLE_TO_CONNECT_TO_SMS | Cannot connect to Monitor Server |
| SMC_RET_UNABLE_TO_CONNECT_TO_SS | Cannot connect to Adaptive Server |

See also

`smc_close`, `smc_connect_alloc`

smc_connect_props

Description Sets, retrieves, or clears properties on a connection.

Syntax

```
SMC_RETURN_CODE smc_connect_props
(SMC_CONNECT_ID    clientId,
 SMC_PROP_ACTION   propertyAction,
 SMC_PROP_TYPE     property,
 SMC_VALUE_UNIONP  propertyValue,
 SMC_SIZET         bufferLength,
 SMC_SIZETP        outputLengthHandle)
```

Parameters

clientId
identifies the connection.

propertyAction

Property action type. Valid types are:

- SMC_PROP_ACT_CLEAR – reset the value of the specified property to its default.
- SMC_PROP_ACT_GET – retrieve the value of the specified property.
- SMC_PROP_ACT_SET – set the value of the specified property.

property

the symbolic name of the property whose value is being set, retrieved, or cleared. See Table 3-5 on page 138 for a list of this argument's legal values.

propertyValue

if *propertyAction* is:

- SMC_PROP_ACT_CLEAR – *propertyValue* is ignored.
- SMC_PROP_ACT_GET – pointer to the union in which *smc_connect_props* will place the requested information.
- SMC_PROP_ACT_SET – pointer to the union that contains the value to which property is to be set.

bufferLength

the length of data in bytes of

*(*propertyValue*->*stringValue*). Used only if *propertyValue* is a pointer to a string. If *propertyAction* is:

- SMC_PROP_ACT_CLEAR – *bufferLength* is ignored, and must be passed SMC_UNUSED.
- SMC_PROP_ACT_GET – *bufferLength* is ignored, and must be passed SMC_UNUSED.
- SMC_PROP_ACT_SET – *bufferLength* must contain the number of bytes of *(*propertyValue*-> *stringValue*) or SMC_NULLTERM to indicate the string's length by a terminating null byte.

outputLengthHandle

a pointer to an integer variable. Used only if *propertyValue* is a pointer to a string. If *propertyAction* is:

- SMC_PROP_ACT_CLEAR – *outputLengthHandle* is ignored, and must be passed null.
- SMC_PROP_ACT_GET – the length in bytes of the requested information. Contains the number of bytes that were actually written to *propertyValue*->*stringValue* (not including the null-terminating byte). Pass null if this information is not desired.
- SMC_PROP_ACT_SET – *outputLengthHandle* is ignored, and must be passed null.

Return value

| Return value | Indicates |
|-------------------------|----------------------------|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |
| SMC_RET_INVALID_CONNECT | Connection does not exist. |

Examples

This example assumes that you have previously allocated a connection using `smc_connect_alloc` and have a *clientId*.

```
SMC_VALUE_UNION value;
value.sizeValue = 512;
if (smc_connect_props(clientId,
    SMC_PROP_ACT_SET,
    SMC_PROP_PACKETSIZE,
    &value,
    0,
    NULL) != SMC_RET_SUCCESS)
{
    printf("smc_connect_props failed\n");
    /* do some cleanup */
}
```

Usage

- A property resets to its default value when cleared.
- `smc_connect_props` must be called after `smc_connect_alloc`.
- The following properties must be set on a connection before calling `smc_connect_ex`:
 - `SMC_PROP_PASSWORD`
 - `SMC_PROP_SERVERNAME`
 - `SMC_PROP_USERNAME`
- The *serverMode* determines which other Monitor Client Library functions are applicable for the connection. For example, `smc_create_recording_session` is not applicable for a live connection.
- The *serverMode* (specified upon creation of a connection) determines the behavior of the common functions. For example, `smc_create_view` can be used to create a live view or a historical view.
- For live connections and historical connections for defining recording sessions, the property `SMC_PROP_USERNAME` must be set to either “sa”, the name of an Adaptive Server account having `sa_role`, or the name of an Adaptive Server account with execute permission on the stored procedure `master.dbo.mon_rpc_connect`.
- To retrieve only the length of a string, pass null for *propertyValue* and a valid pointer for *outputLengthHandle*.
- For the definition of a `SMC_VALUE_UNION` structure, see “Union: `SMC_VALUE_UNION`” on page 240.

- For data of type SMC_CHARP, *stringValue* points to the value. The Monitor Client Library allocates the memory for this string and the calling application must deallocate it using `free()`.
- The following properties are only valid before a connection is made:
 - SMC_PROP_APPNAME
 - SMC_PROP_IFILE
 - SMC_PROP_PASSWORD
 - SMC_PROP_SERVERMODE
 - SMC_PROP_SERVERNAME
 - SMC_PROP_USERNAME.

If these properties are changed on a connection after it has been established, they take effect during the next call to `smc_connect_ex`.

- Table 3-5 summarizes the Monitor Client Library properties, whether they can be set, retrieved, or cleared, and the datatype of each property value:

Table 3-5: Monitor Client Library connection properties

| Property | Set/ Get/ Clear | *propertyValue is | Default |
|-------------------------|-----------------------|--|---|
| SMC_PROP_APPNAME | All | SMC_CHARP | An empty string |
| SMC_PROP_ERROR_CALLBACK | Set/ Get | A function pointer (use <i>voidpValue</i> member of SMC_VALUE_UNION) | |
| SMC_PROP_IFILE | All | SMC_CHARP | Empty string, signifying the <i>interfaces</i> file in directory where the SYBASE environment variable points (on Windows, <i>sql.ini</i> in the <i>ini</i> subdirectory) |

| Property | Set/ Get/ Clear | *propertyValue is | Default |
|------------------------|-----------------------|-------------------|----------------------------|
| SMC_PROP_LOGIN_TIMEOUT | All | SMC_SIZE_T | 0 (Use the server default) |
| SMC_PROP_PACKETSIZE | All | SMC_SIZE_T | 0 (Use the server default) |
| SMC_PROP_PASSWORD | Set/ Clear | SMC_CHARP | An empty string |
| SMC_PROP_SERVERMODE | All | SMC_INT | SMC_SERVER_M_LIVE |
| SMC_PROP_SERVERNAME | All | SMC_CHARP | An empty string |
| SMC_PROP_TIMEOUT | All | SMC_SIZE_T | 0 (Use the server default) |
| SMC_PROP_USERDATA | All | SMC_VOIDP | NULL |
| SMC_PROP_USERNAME | All | SMC_CHARP | An empty string |

Properties

| Property | Description |
|-------------------------|--|
| SMC_PROP_APPNAME | The name of the application using Monitor Client Library. This property can be modified at any time, but takes effect only when <code>smc_connect_ex</code> is called. |
| SMC_PROP_ERROR_CALLBACK | The error callback function. This property can be modified at any time during the connection. |
| SMC_PROP_IFILE | The <i>interfaces</i> file. This property can be modified at any time, but takes effect only when <code>smc_connect_ex</code> is called. |
| SMC_PROP_LOGIN_TIMEOUT | The timeout value (in seconds) used during login time. This property can be modified at any time, but takes effect when only <code>smc_connect_ex</code> is called. |
| SMC_PROP_PACKETSIZE | The packet size to use for communicating to the servers. This property can be modified at any time during the connection. |
| SMC_PROP_PASSWORD | The password. This property can be modified at any time, but takes effect only when <code>smc_connect_ex</code> is called. |

| Property | Description |
|---------------------|--|
| SMC_PROP_SERVERMODE | The server mode. This property can be set only before a connection is established. It can be modified at any time, but takes effect when only <code>smc_connect_ex</code> is called. The value is an enum: <code>SMC_SERVER_MODE</code> . See “Enum: <code>SMC_SERVER_MODE</code> ” on page 239. |
| SMC_PROP_SERVERNAME | The server name. This property can be modified at any time, but takes effect only when <code>smc_connect_ex</code> is called. |
| SMC_PROP_TIMEOUT | The timeout value to use for requests sent to the servers. This property can be modified at any time during the connection. |
| SMC_PROP_USERDATA | A user-supplied pointer. This pointer is passed back to callback functions. It can be changed at any time on an available connection. |
| SMC_PROP_USERNAME | The <i>username</i> to use for this connection. This property can be modified at any time, but takes effect only when <code>smc_connect_ex</code> is called. |

Valid server modes

| Mode | Availability |
|-------------------------|--------------|
| SMC_SERVER_M_LIVE | Yes |
| SMC_SERVER_M_HISTORICAL | Yes |

Errors

| Error | Indicates |
|------------------------------|---|
| SMC_RET_INVALID_API_FUNCTION | Invalid use of obsolete and replacement functions in program. |
| SMC_RET_INVALID_PARAMETER | Invalid parameter value. |

See also

`smc_connect_alloc`, `smc_connect_ex`

smc_create_alarm_ex

Description Creates an alarm on one data item within a view on a connection.

Syntax

```
SMC_RETURN_CODE smc_create_alarm_ex
(SMC_CONNECT_ID  clientId,
 SMC_VIEW_ID     viewId,
```

```

SMC_DATAITEM_STRUCTP  dataItemHandle,
SMC_VALUE_UNIONP      alarmValueDataHandle,
SMC_DATAITEM_TYPE      alarmDatatype,
SMC_ALARM_ACTION_TYPE  alarmActionType,
SMC_CHARP              alarmActionData,
SMC_VOIDP              userDataHandle,
SMC_GEN_CALLBACK       alarmCallback,
SMC_ALARM_IDP          alarmIdHandle)

```

Parameters

clientId

identifies the connection.

viewId

identifies a view created on the connection.

dataItemHandle

pointer to data item and statistic type.

alarmValueDataHandle

pointer to threshold at or above which the alarm is triggered.

alarmDatatype

the datatype of the alarm value must be one of the following and must match the expected datatype for the given data item:

- SMC_DI_TYPE_DOUBLE
- SMC_DI_TYPE_INT
- SMC_DI_TYPE_LONG

alarmActionType

- SMC_ALARM_A_NOTIFY (SMC_SERVER_M_LIVE mode only) – invokes the alarm callback.
- SMC_ALARM_A_EXEC_PROC (SMC_SERVER_M_HISTORICAL mode only) – invokes the specified external program.
- SMC_ALARM_A_LOG_TO_FILE (SMC_SERVER_M_HISTORICAL mode only) – writes a message to the log file.

alarmActionData

pointer to null-terminated string whose contents depend on *alarmActionType*. If *alarmActionType* equals:

- SMC_ALARM_A_NOTIFY – *alarmActionData* is ignored.
- SMC_ALARM_A_EXEC_PROC – null-terminated string that contains the filename and optional parameter list of the program to invoke.

- SMC_ALARM_A_LOG_TO_FILE – null-terminated string that contains the log file name.
- These file names are on the system where Historical Server is running (which need not be where the application is running). The Historical Server must have access to the files.

userDataHandle
user-supplied pointer.

alarmCallback
identifies the notification function employed by *alarmActionType*, SMC_ALARM_A_NOTIFY.

alarmIdHandle
pointer to a variable, which should be declared as type SMC_ALARM_ID. If the call to `smc_create_alarm` succeeds, this variable contains the ID for the alarm.

Return value

| Return value | Indicates |
|-------------------------|----------------------------|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |
| SMC_RET_INVALID_CONNECT | Connection does not exist. |

Examples

- This example assumes that:
- You have created a connection using `smc_connect_ex` and have a *clientId*.
 - You have created a view on the connection and have a *viewId*.
 - The view contains the *dataItem* SMC_NAME_PAGE_LOGICAL_READ, SMC_STAT_VALUE_SAMPLE.
 - You have defined an alarm handler function, *myAlarmHandler*.

```
SMC_DATAITEM_STRUCT  dataItem =
                        { SMC_NAME_PAGE_LOGICAL_READ,
                          SMC_STAT_VALUE_SAMPLE };
SMC_DATAITEM_STRUCTP dataItemHandle = &dataItem;
SMC_VALUE_UNION alarmValue;
SMC_VALUE_UNIONP alarmValueHandle = &alarmValue;
SMC_ALARM_ID alarmId;
SMC_ALARM_IDP alarmIdHandle = &alarmId;
alarmValue.longValue = 10L;

if (smc_create_alarm_ex(clientId,
```

```
        viewId,  
        dataItemHandle,  
        alarmValueHandle,  
        SMC_DI_TYPE_LONG,  
        SMC_ALARM_A_NOTIFY,  
        NULL, /* ignored */  
        NULL, /* no user data */  
        myAlarmHandler,  
        alarmIdHandle) != SMC_RET_SUCCESS)  
{  
    printf("smc_create_alarm_ex failed\n");  
    /* do some cleanup */  
}
```

Usage

- Alarms can be created on result data items, but not on key data items.
- *alarmIds* are unique only within a given view.
- Alarms are triggered for each row of a view where the data item value meets or exceeds the threshold.
- Alarms are applied after filters, in the context of a refresh call.
- Alarms are triggered at each refresh based upon a data item's value (state) rather than the change of a data item's value (transition).
- Multiple alarms can be created on the same data item.
- When used in a Historical Server connection during the definition of a recording session, `smc_create_alarm_ex` defines an alarm that will be created during the execution of a recording session.
- Alarms cannot be defined in a Historical Server connection during a playback session.
- When creating a log-to-file alarm, if you specify a UNIX directory for the location of the log file, be sure that the directory is valid and mounted on the machine where Historical Server is running. Also be sure that you have write permissions to the directory. If the directory you specify is invalid, unmounted, or not writable, Historical Server does not create a log file, nor does it issue a message advising you that the location is invalid.

The syntax of the alarm callback is:

```
SMC_VOID AlarmCallback  
(SMC_CONNECT_ID    clientId,  
    SMC_COMMAND_ID  commandId,  
    SMC_VOIDP       userDataHandle)
```

Valid server modes

| Mode | Availability |
|-------------------------|---------------------|
| SMC_SERVER_M_LIVE | Yes |
| SMC_SERVER_M_HISTORICAL | Yes (for recording) |

Errors

| Error | Indicates |
|------------------------------------|---|
| SMC_RET_INSUFFICIENT_MEMORY | Insufficient memory |
| SMC_RET_INVALID_ALARM_VALUE | Invalid alarm value |
| SMC_RET_INVALID_API_FUNCTION | Invalid use of obsolete and replacement functions within the same program |
| SMC_RET_INVALID_DATAITEM_FOR_ALARM | Data item statistic type or alarm value mismatched |
| SMC_RET_INVALID_DATATYPE | Invalid datatype |
| SMC_RET_INVALID_DINAME | Data item does not exist |
| SMC_RET_INVALID_DISTAT | Data item statistic type does not exist |
| SMC_RET_INVALID_PARAMETER | Invalid parameter value |
| SMC_RET_INVALID_VIEWID | View does not exist |
| SMC_RET_INTERNAL_ERROR | Internal error |

Callback parameters

| Parameter | Description |
|-----------------------|---|
| <i>clientId</i> | Identifies the connection. |
| <i>commandId</i> | Identifies the instance of a command. |
| <i>userDataHandle</i> | Pointer that was set by the call to <code>smc_create_alarm</code> for this alarm. |

The alarm callback function uses `smc_get_command_info` to obtain information about the circumstances that triggered the alarm.

See also

`smc_connect_ex`, `smc_drop_alarm`, `smc_get_command_info`

smc_create_filter

Description

Creates a filter on a data item in a view. Each data item in a view can have only one filter.

This function can be used with both Monitor Server and Historical Server. When used with Historical Server (that is, when the connection mode is SMC_SERVER_M_HISTORICAL), it creates a filter for the recording session that is being defined.

Syntax

```
SMC_RETURN_CODE smc_create_filter
(SMC_CONNECT_ID      clientId,
SMC_VIEW_ID          viewId,
SMC_DATAITEM_STRUCTP dataItemHandle,
SMC_FILTER_TYPE      filterType,
SMC_VALUE_UNIONP     filterValueListHandle,
SMC_SIZET             filterValueListLength,
SMC_DATAITEM_TYPE    filterDatatype,
SMC_FILTER_IDP       filterIdHandle)
```

Parameters

clientId

identifies the connection.

viewId

identifies a view created on the connection.

dataItemHandle

data item and statistic type. The data item must be numeric if the filter type is any of the following:

- SMC_FILT_T_GE
- SMC_FILT_T_LE
- SMC_FILT_T_GE_AND_LE
- SMC_FILT_TOP_N

filterType

type of filter to apply. Valid filter types are:

- SMC_FILT_T_EQ – equal to.
- SMC_FILT_T_NEQ – not equal to.
- SMC_FILT_T_GE – greater than or equal to.
- SMC_FILT_T_LE – less than or equal to.
- SMC_FILT_T_GE_AND_LE – a lower bound followed by an upper bound.
- SMC_FILT_T_TOP_N – top N.

filterValueListHandle

pointer to an array of filter values. The number of filter values depends on the filter type:

- SMC_FILT_T_EQ – one or more.
- SMC_FILT_T_NEQ – one or more.
- SMC_FILT_T_GE – one.
- SMC_FILT_T_LE – one.
- SMC_FILT_T_GE_AND_LE – two; low bound must be first element in list and high bound second.
- SMC_FILT_T_TOP_N – one.

filterValueListLength

number of filter values listed in *filterValueListHandle*.

filterDataType

datatype of the values for the filter; one of the following:

- SMC_DI_TYPE_CHARP
- SMC_DI_TYPE_DATIM
- SMC_DI_TYPE_DOUBLE
- SMC_DI_TYPE_ENUMS
- SMC_DI_TYPE_INT
- SMC_DI_TYPE_LONG

Must match the datatype for the data item. The filter values must also be of this type, except:

- If the filter type is SMC_FILT_T_TOP_N, the filter value in the *filterValueListHandle* must be type SMC_INT.
- If the datatype is SMC_DI_TYPE_ENUMS, the filter value in the *filterValueListHandle* must be passed using the *intValue* member.

filterIdHandle

pointer to a variable, which should be declared as type SMC_FILTER_ID. If the call to `smc_create_filter` succeeds, this variable contains the ID for the filter.

Return value

| Return value | Indicates |
|-------------------------|----------------------------|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |
| SMC_RET_INVALID_CONNECT | Connection does not exist. |

Examples

The following example assumes that:

- You have created a connection and have a *clientId*.
- You have created a view on that connection and have a *viewId*.
- The view contains the *dataItem* defined in the example.

```
SMC_DATAITEM_STRUCT  dataItem =
                        { SMC_NAME_PAGE_LOGICAL_READ,
                          SMC_STAT_VALUE_SAMPLE };
SMC_DATAITEM_STRUCTP dataItemHandle = &dataItem;
SMC_VALUE_UNION  filterValue;
SMC_VALUE_UNIONP filterValueHandle = &filterValue;
SMC_FILTER_ID  filterId;
SMC_FILTER_IDP filterIdHandle = &filterId;
filterValue.longValue = 10L;

if (smc_create_filter(clientId,
                     viewId,
                     dataItemHandle,
                     SMC_FILT_T_GE,
                     filterValueHandle,
                     1, /* just one filterValue */
                     SMC_DI_TYPE_LONG,
                     filterIdHandle) != SMC_RET_SUCCESS)
{
    printf("smc_create_filter failed\n");
    /* do some cleanup */
}
```

Usage

- The application can employ wildcard (%) characters on all filters that apply to string datatypes.
- Filters are applied before alarms, in the context of a refresh call.
- Only one filter can be created on a data item.
- A filter defined for a recording session is not created until execution of the recording session.
- Not allowed during playback.
- For database objects, you can define SMC_FILT_T_EQ filters on the name of the object, that is, on a data item of SMC_NAME_OBJ_NAME or SMC_NAME_ACT_STP_NAME. The string value must include the fully qualified object name, for example, *database.owner.object*. However, you can use wildcards for each component of the name.

Valid server modes

| Mode | Availability |
|-------------------------|--------------------------|
| SMC_SERVER_M_LIVE | Yes |
| SMC_SERVER_M_HISTORICAL | Yes (for recording only) |

Errors

| Error | Indicates |
|----------------------------------|---|
| SMC_RET_INSUFFICIENT_MEMORY | Insufficient memory |
| SMC_RET_INVALID_COMPOSITE_FILTER | Invalid composite filter |
| SMC_RET_MISSING_DATAITEM | Missing data item |
| SMC_RET_INVALID_DATATYPE | Invalid datatype |
| SMC_RET_INVALID_DINAME | Invalid data item |
| SMC_RET_INVALID_DISTAT | Invalid data item statistic type |
| SMC_RET_INVALID_FILTER_VALUE | Invalid value for filter |
| SMC_RET_INVALID_FILTER_RANGE | Invalid range values |
| SMC_RET_INVALID_VALUE_COUNT | Invalid value for <i>filterValueListLength</i> |
| SMC_RET_INVALID_VIEWID | View does not exist |

See also

smc_drop_filter

smc_create_playback_session

Description Initializes a playback session on Historical Server.

Syntax

```
SMC_RETURN_CODE smc_create_playback_session
(SMC_CONNECT_ID      clientId,
  SMC_SESSION_IDP     sessionIdArray,
  SMC_SIZET            numInputSessions,
  SMC_CHARP            startTime,
  SMC_CHARP            endTime,
  SMC_HS_PLAYBACK_OPT  playbackType,
  SMC_SIZET            summarizationInterval,
  SMC_HS_ESTIM_OPT     estimationOption,
  SMC_HS_MISSDATA_OPT  missingDataOption,
  SMC_HS_TARGET_OPT    playbackTarget,
  SMC_CHARP            directoryName
  SMC_HS_SESS_PROT_LEVEL protectionLevel,
  SMC_HS_SESS_SCRIPT_OPT scriptOption,
  SMC_HS_SESS_DELETE_OPT deleteOption,
  SMC_SESSION_IDP      sessionIdHandle)
```

Parameters

clientId

identifies the connection.

sessionIdArray

array of session numbers identifying the existing recording session(s) on Historical Server that furnishes data for this playback session. If more than one input session is specified, then they all must have been defined to record data from the same Adaptive Server, and they must be ordered chronologically.

If *playbackTarget* is SMC_HS_TARGET_FILE, then there must not be any gaps between the times covered by multiple input sessions. The input sessions must contain data for all times between the *startTime* and *endTime* parameters.

numInputSessions

the number of input sessions, that is, the length of the *sessionIdArray*. Must be at least one.

startTime

null-terminated string containing the time to start playback, using the format:

```
yyyy/mm/dd hh:mm[:ss] [time zone]
```

The default is to start at the beginning of the first input session.

endTime

null-terminated string containing the time at which to stop playback, using the format:

```
yy/mm/dd hh:mm[:ss] [time zone]
```

The default is to stop at the end of the last input session.

playbackType

specifies the level of detail of the playback. Valid values are:

- SMC_HS_PBTTYPE_RAW – plays back data as it was collected, using whatever (possibly varying) intervals are contained in the input session. This option can include snapshot data such as current SQL statement data and status on locks or processes. Valid only with *playbackTarget* SMC_HS_TARGET_CLIENT.
- SMC_HS_PBTTYPE_ACTUAL – plays back data at whatever (possibly varying) intervals are contained in the input session(s). This option cannot include snapshot data.

- SMC_HS_PBTTYPE_INTERVAL – plays back data summarized into sample intervals of the length specified in *summarizationInterval*.
- SMC_HS_PBTTYPE_ENTIRE – plays back data for each input recording session summarized as a single sample. The sample interval is the time between the requested playback *startTime* and *endTime*.

If *playbackTarget* is SMC_HS_TARGET_FILE, then *playbackType* must be SMC_HS_PBTTYPE_INTERVAL or SMC_HS_PBTTYPE_ENTIRE.

summarizationInterval

if *playbackType* is SMC_HS_PBTTYPE_INTERVAL, then this specifies the length in seconds of the playback intervals over which the input data is to be summarized.

For other values of *playbackType*, applications must specify SMC_UNUSED for this parameter.

estimationOption

specifies whether playback may estimate the values of data items that cannot be calculated exactly. Valid values are:

- SMC_HS_ESTIM_ALLOW
- SMC_HS_ESTIM_DISALLOW

If SMC_HS_ESTIM_DISALLOW is specified, then a subsequent call for this playback session to *smc_create_view* will return an error if it includes data items requiring estimation.

This option is ignored if *playbackType* is SMC_HS_PBTTYPE_RAW.

missingDataOption

specifies whether the Monitor Client Library will return playback samples for periods of time when no data is available in the input session(s). Valid values are:

- SMC_HS_MISSDATA_SHOW – Monitor Client Library will return a sample for periods of time lacking data.
- SMC_HS_MISSDATA_SKIP – Monitor Client Library will not return a sample for periods of time lacking data; instead, the Library will return data for the next available time interval for which data is available.

If *playbackTarget* is SMC_HS_TARGET_FILE, this parameter is ignored.

playbackTarget

specifies whether the playback session returns data to the application or whether playback creates a new session on Historical Server. Valid values are:

- `SMC_HS_TARGET_CLIENT` – the playback session returns data to the application, by means of calls to `smc_refresh_ex`.
- `SMC_HS_TARGET_FILE` – playback creates a new session on Historical Server.

directoryName

if *playbackTarget* is `SMC_HS_TARGET_FILE`, this parameter specifies the directory in which the Historical Server creates the data file(s) and error file for the new sessions to be created.

protectionLevel

if *playbackTarget* is `SMC_HS_TARGET_FILE`, this parameter specifies the protection level of the new session to be created. Valid values are:

- `SMC_HS_SESS_PROT_PUBLIC`
- `SMC_HS_SESS_PROT_PRIVATE`

This parameter is ignored if *playbackTarget* is `SMC_HS_TARGET_CLIENT`.

scriptOption

if *playbackTarget* is `SMC_HS_TARGET_FILE`, this parameter specifies whether Historical Server must create a script that creates tables for loading results (from the new session) into Adaptive Server. The choices are:

- `SMC_HS_SESS_SCRIPT_NONE` – no script.
- `SMC_HS_SESS_SCRIPT_SYBASE` – Sybase script.

This parameter is ignored if *playbackTarget* is `SMC_HS_TARGET_CLIENT`.

deleteOption

if *playbackTarget* is `SMC_HS_TARGET_FILE`, this parameter specifies whether Historical Server must delete the input session(s) after successfully creating a new session. The choices are:

- `SMC_HS_DELETE_FILES`
- `SMC_HS_RETAIN_FILES`

This parameter is ignored if *playbackTarget* is `SMC_HS_TARGET_CLIENT`.

sessionIdHandle

if *playbackTarget* is SMC_HS_TARGET_FILE, this parameter must be a pointer to a variable of type SMC_SESSION_ID, into which the Monitor Client Library writes the identifier for the new session.

This parameter is ignored if *playbackTarget* is SMC_HS_TARGET_CLIENT.

Return value

| Return value | Indicates |
|-------------------------|----------------------------|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |
| SMC_RET_INVALID_CONNECT | Connection does not exist. |

Examples

This example assumes that you have created a connection to Historical Server and have a *clientId*.

```
SMC_SESSION_ID  inputSessions[2];

if (smc_create_playback_session(clientId,
    inputSessions,
    2,/* number of input sessions */
    "",/* default start time */
    "",/* default end time */
    SMC_HS_PBTTYPE_INTERVAL,/* summarize at */
    60,/* uniform minute intervals */
    SMC_HS_ESTIM_ALLOW,/* allow estimation */
    SMC_HS_MISSDATA_SHOW, /* produce a sample*/
    /* every minute even if no data is */
    /* available for that interval */
    SMC_HS_TARGET_CLIENT, /* do playback */
    "",/* directory name */
    SMC_HS_SESS_PROT_PUBLIC,/* so next 5 */
    SMC_HS_SESS_SCRIPT_SYBASE,/* are */
    SMC_HS_DELETE_FILES,/* unused */
    NULL)/* No output session ID */
    != SMC_RET_SUCCESS)
{
    printf("smc_create_playback_session failed\n");
    /* do some cleanup */
}
```

Usage

- In a Historical Server connection, recording sessions and playback sessions are mutually exclusive. An application that connects to a Historical Server and defines a recording session, must complete the definition of the recording session using the function `smc_initiate_recording` before creating a playback session.
- If the *playbackType* is `SMC_HS_PBTTYPE_RAW`, the application can specify only one input session. Otherwise, the application can specify any number of input sessions (but at least one), provided that all sessions were recorded against the same Adaptive Server installation and Monitor Server.
- If the *playbackType* is `SMC_HS_PBTTYPE_RAW`, different rules apply to the definition of playback views. Refer to the *Adaptive Server Enterprise Monitor Historical Server User's Guide* for more information about views.
- You cannot combine *playbackTarget* `SMC_HS_TARGET_FILE` with *playbackType* `SMC_HS_PBTTYPE_RAW` or `SMC_HS_PBTTYPE_ACTUAL`.
- Input sessions can include recording sessions that are still in the process of recording, unless *playbackTarget* is `SMC_HS_TARGET_FILE`.
- If *playbackTarget* is `SMC_HS_TARGET_FILE`, then the input session must contain performance data for the entire time from *startTime* to *endTime*, with no gaps between input sessions.
- Refer to the *Monitor Historical Server User's Guide* for more information about the `hs_create_playback_session` command.

Valid server modes

| Mode | Availability |
|--------------------------------------|--------------|
| <code>SMC_SERVER_M_LIVE</code> | No |
| <code>SMC_SERVER_M_HISTORICAL</code> | Yes |

Errors

| Error | Indicates |
|-------------------------------------|---------------------|
| <code>SMC_RET_INTERNAL_ERROR</code> | Internal error |
| <code>SMC_INVALID_SVR_MODE</code> | Invalid server mode |

See also

`smc_initiate_playback`

smc_create_recording_session

| | |
|-------------|--|
| Description | <p>Initiates the definition of a recording session on Historical Server.</p> <p>This function is applicable only if the connection mode is SMC_SERVER_M_HISTORICAL.</p> |
| Syntax | <pre>SMC_RETURN_CODE smc_create_recording_session (SMC_CONNECT_ID clientId, SMC_CHARP SMSName, SMC_INT sampleInterval, SMC_CHARP directoryName, SMC_CHARP startTime, SMC_CHARP endTime, SMC_HS_SESS_PROT_LEVEL protectionLevel, SMC_HS_SESS_ERR_OPT errOption, SMC_HS_SESS_SCRIPT_OPT scriptOption, SMC_SESSION_IDP sessionIdHandle)</pre> |
| Parameters | <p><i>clientId</i> identifies the connection.</p> <p><i>SMSName</i> null-terminated string containing the name of the Monitor Server.</p> <p><i>sampleInterval</i> the number of seconds to wait between consecutive samplings of data.</p> <p><i>directoryName</i> null-terminated string containing the full path name to the directory containing the data and error files created by Historical Server during execution of this recording session.</p> <p>The directory must be writable on the system on which Historical Server is running. This might not be the same system that is running the client application that invoked the function call.</p> <p><i>startTime</i> null-terminated string containing the time to start recording, using the format:</p> <pre>yyyy/mm/dd hh:mm[:ss] [time zone]</pre> <p>The default is to start immediately.</p> <p><i>endTime</i> null-terminated string containing the time at which to stop the recording, using the format:</p> <pre>yy/mm/dd hh:mm[:ss] [time zone]</pre> |

The default is to stop 24 hours after *startTime*.

protectionLevel

protection level of the data recorded. Valid values are:

- SMC_HS_SESS_PROT_PUBLIC
- SMC_HS_SESS_PROT_PRIVATE

errOption

indicate what Historical Server must do when encountering a non-fatal error. The choices are:

- SMC_HS_SESS_ERR_CONT – continue the session.
- SMC_HS_SESS_ERR_HALT – stop the session.

scriptOption

indicate whether Historical Server must create a script that creates tables for loading results (from this recording session) into Adaptive Server. The choices are:

- SMC_HS_SESS_SCRIPT_NONE – no script.
- SMC_HS_SESS_SCRIPT_SYBASE – Sybase script.

sessionIdHandle

pointer to a variable, which should be declared as type SMC_SESSION_ID. If the call to `smc_create_recording_session` succeeds, this variable contains the ID for the recording session.

Return value

| Return value | Indicates |
|-------------------------|----------------------------|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |
| SMC_RET_INVALID_CONNECT | Connection does not exist. |

Examples

This example assumes that you have created a connection to Historical Server and have a *clientId*.

```
SMC_SESSION_ID sessionId;
SMC_SESSION_IDP sessionIdHandle = &sessionId;
if (smc_create_recording_session(clientId,
    "myMonitorServer",
    60, /* sample interval (seconds) */
    "/usr/hist_serv_home_dir",
    "95/07/22 15:00", /* start time */
    "95/07/23 15:30", /* end time */
    SMC_HS_SESS_PROT_PUBLIC,
```

```
SMC_HS_SESS_ERR_CONT,  
SMC_HS_SESS_SCRIPT_SYBASE,  
sessionIdHandle) != SMC_RET_SUCCESS)  
{  
    printf("smc_create_recording_session failed\n");  
    /* do some cleanup */  
}
```

Usage

- In a Historical Server connection, recording sessions and playback sessions are mutually exclusive. An application that connects to Historical Server and creates a playback session must end the playback session using the function `smc_terminate_playback` before creating a recording session.
- Refer to the *Adaptive Server Enterprise Monitor Historical Server User's Guide* for more information on the `hs_create_recording_session` command.

Valid server modes

| Mode | Availability |
|-------------------------|--------------|
| SMC_SERVER_M_LIVE | No |
| SMC_SERVER_M_HISTORICAL | Yes |

Errors

| Error | Indicates |
|--------------------------|---------------------|
| SMC_RET_INTERNAL_ERROR | Internal error |
| SMC_RET_INVALID_SVR_MODE | Invalid server mode |

See also

`smc_initiate_recording`

smc_create_view

Description

Creates a view that can contain one or more data items.

For information about data items, refer to Chapter 2, “Data Items and Statistical Types”.

You can use the `smc_create_view` function with both Monitor Server and Historical Server. When used with Historical Server (SMC_SERVER_M_HISTORICAL), it creates a view for the recording or playback session that is being defined.

Syntax

```
SMC_RETURN_CODE smc_create_view  
(SMC_CONNECT_ID      clientId,
```

```

SMC_DATAITEM_STRUCTP dataItemHandle,
SMC_SIZET             dataItemListLength,
SMC_CHARP             viewName,
SMC_VIEW_IDP          viewIdHandle)

```

Parameters*clientId*

identifies the connection.

dataItemHandle

pointer to array of SMC_DATAITEM_STRUCTs.

*dataItemListLength*number of data items in the array pointed to by the *dataItemHandle*.*viewName*

null-terminated string containing a descriptive name for this view. This name can include a – z, A – Z, 0 – 9, and underscore (_) characters, or can be NULL.

Used only for a Historical Server connection. For a live connection, the view name is ignored.

viewIdHandle

pointer to a variable, which should be declared as type SMC_VIEW_ID. If the call to `smc_create_view` succeeds, this variable contains the ID for the view.

Return value

| Return value | Indicates |
|-------------------------|----------------------------|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |
| SMC_RET_INVALID_CONNECT | Connection does not exist. |

Examples

This example assumes that you have created a connection and have a *clientId*.

```

SMC_DATAITEM_STRUCT  dataItem =
    { SMC_NAME_PAGE_LOGICAL_READ,
      SMC_STAT_VALUE_SAMPLE };
SMC_DATAITEM_STRUCTP dataItemHandle = &dataItem;
SMC_VIEW_ID viewId;
SMC_VIEW_IDP viewHandle = &viewId;

if (smc_create_view(clientId,
    dataItemHandle,
    1, /* just one dataItem */
    NULL, /* this is a Monitor Server view */
    viewIdHandle) != SMC_RET_SUCCESS)

```

```
{
    printf("smc_create_view failed\n");
    /* do some cleanup */
}
```

Usage

- Refer to Chapter 2, “Data Items and Statistical Types” for rules for using views with live views.
- When called against a Historical Monitor connection, `smc_create_view` must be preceded by a call to `smc_create_recording_session` or `smc_create_playback_session`.
- When used in Historical Server during the definition of a recording session, it defines a view to be recorded by Historical Server during the recording session.
- When used in Historical Server during a playback session, it selects a view for playback from those previously recorded in recording session(s). If the playback session uses more than one input session, then the selected view must exist in all input sessions and use the same name, data items, and filters.
- Depending on whether the playback session was created for “raw” or summarizing playback, the playback view may or may not include certain data items from the original view. Refer to the *Adaptive Server Enterprise Monitor Historical Server User’s Guide* for more information on the `hs_create_playback_view` command.

Valid server modes

| Mode | Availability |
|-------------------------|--------------|
| SMC_SERVER_M_LIVE | Yes |
| SMC_SERVER_M_HISTORICAL | Yes |

Errors

| Error | Indicates |
|-----------------------------------|--|
| SMC_RET_INVALID_API_FUNC_SEQUENCE | Invalid calling sequence of Monitor Client Library functions |
| SMC_RET_INVALID_DINAME | Invalid data item |
| SMC_RET_INVALID_DI_STATTYPE | Invalid data item statistic type |
| SMC_RET_INSUFFICIENT_MEMORY | Insufficient memory |

See also

`smc_create_recording_session`, `smc_create_playback_session`,
`smc_initiate_recording`, `smc_initiate_playback`, `smc_drop_view`

smc_drop_alarm

Description Removes an alarm on a data item in a view.

Syntax SMC_RETURN_CODE smc_drop_alarm
(SMC_CONNECT_ID *clientId*,
SMC_VIEW_ID *viewId*,
SMC_ALARM_ID *alarmId*)

Parameters

clientId
identifies the connection.

viewId
identifies a view created on the connection.

alarmId
identifies the alarm.

Return value

| Return value | Indicates |
|-------------------------|--|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |
| SMC_RET_BUSY | Function not executed, connection is busy. |
| SMC_RET_INVALID_CONNECT | Connection does not exist. |

Examples The following example assumes that:

- You have created a connection and have a *clientId*.
- You have created a view on that connection and have a *viewId*.
- You have created an alarm on that view and have an *alarmId*.

```
if (smc_drop_alarm(clientId,
                  viewId,
                  alarmId) != SMC_RET_SUCCESS)
{
    printf("smc_drop_alarm_failed\n");
    exit(1);
}
```

Usage You cannot drop an alarm created while defining a Historical session (that is, when the connection mode is SMC_SERVER_M_HISTORICAL).

Valid server modes

| Mode | Availability |
|-------------------------|--------------|
| SMC_SERVER_M_LIVE | Yes |
| SMC_SERVER_M_HISTORICAL | No |

| Errors | |
|-------------------------|-----------------------|
| Error | Indicates |
| SMC_RET_INVALID_VIEWID | Function failed. |
| SMC_RET_INVALID_ALARMID | Alarm does not exist. |

See also `smc_create_alarm_ex`, `smc_drop_view`

smc_drop_filter

Description Removes a filter on a data item.

Syntax `SMC_RETURN_CODE smc_drop_filter`
`(SMC_CONNECT_ID clientId,`
`SMC_VIEW_ID viewId,`
`SMC_FILTER_ID filterId)`

Parameters *clientId*
identifies the connection.

viewId
identifies a view created on the connection.

filterId
identifies the filter to be dropped.

| Return value | Return value | Indicates |
|--------------|-------------------------|----------------------------|
| | SMC_RET_SUCCESS | Function succeeded. |
| | SMC_RET_FAILURE | Function failed. |
| | SMC_RET_INVALID_CONNECT | Connection does not exist. |

Examples The following example assumes that:

- You have created a connection and have a *clientId*.
- You have created a view on that connection and have a *viewId*.
- You have created a filter on that view and have a *filterId*.

```
if (smc_drop_filter(clientId,
                    viewId,
                    filterId) != SMC_RET_SUCCESS)
{
    printf("smc_drop_filter_failed\n");
    /* do some cleanup */
}
```

```
}

```

Usage

- Dropping a filter takes effect at the next call to `smc_refresh` following the call to `smc_drop_filter`.
- You cannot drop a filter created while defining a Historical Server session (that is, when the connection mode is `SMC_SERVER_M_HISTORICAL`).

Valid server modes

| Mode | Availability |
|--------------------------------------|--------------|
| <code>SMC_SERVER_M_LIVE</code> | Yes |
| <code>SMC_SERVER_M_HISTORICAL</code> | No |

Errors

| Error | Indicates |
|---------------------------------------|------------------------|
| <code>SMC_RET_INVALID_VIEWID</code> | View does not exist. |
| <code>SMC_RET_INVALID_FILTERID</code> | Filter does not exist. |

See also

`smc_create_filter`, `smc_drop_view`

smc_drop_view

Description

Removes a view from a connection.

Syntax

```
SMC_RETURN_CODE smc_drop_view
(SMC_CONNECT_ID clientId,
 SMC_VIEW_ID   viewId)
```

Parameters

clientId

identifies the connection.

viewId

identifies a view created on the connection.

Return value

| Return value | Indicates |
|--------------------------------------|----------------------------|
| <code>SMC_RET_SUCCESS</code> | Function succeeded. |
| <code>SMC_RET_FAILURE</code> | Function failed. |
| <code>SMC_RET_INVALID_CONNECT</code> | Connection does not exist. |

Examples

The following example assumes that:

- You have created a connection and have a *clientId*.

- You have created a view on that connection and have a *viewId*.

```
if (smc_drop_view(clientId,
                  viewId) != SMC_RET_SUCCESS)
{
    printf("smc_drop_view_failed\n");
    /* do some cleanup */
}
```

Usage

- All alarms and filters associated with the data items in the view are dropped.
- You cannot drop a view created on a Historical Server session (that is, when the connection mode is SMC_SERVER_M_HISTORICAL).

Valid server modes

| Mode | Availability |
|-------------------------|--------------|
| SMC_SERVER_M_LIVE | Yes |
| SMC_SERVER_M_HISTORICAL | No |

Error

| Error | Indicates |
|------------------------|----------------------|
| SMC_RET_INVALID_VIEWID | View does not exist. |

See also

smc_create_view, smc_drop_alarm, smc_drop_filter

smc_get_command_info

Description

Retrieves detailed information about an alarm or error notification.

Syntax

```
SMC_RETURN_CODE smc_get_command_info
(SMC_CONNECT_ID   clientId,
 SMC_COMMAND_ID   commandId,
 SMC_INFO_TYPE     infoType,
 SMC_VALUE_UNIONP  infoValue,
 SMC_SIZETP        outputLengthHandle)
```

Parameters

clientId
identifies the connection.

commandId
identifies an invocation of a callback function.

infoType

describes the type of requested information. See “Data available for alarm callbacks” on page 129.

infoValue

pointer to an `SMC_VALUE_UNION` structure receiving the value of *infoType*.

outputLengthHandle

a pointer to an integer variable. Upon a successful call to `smc_get_command_info`, the Monitor Client Library writes into this variable. The actual length, in bytes, of the data to be copied into **infoValue* (not including the null-terminator byte). If the *infoValue* datatype is not `SMC_CHARP`, this parameter is ignored. Pass null if the information is not desired.

Return value

| Return value | Indicates |
|---|--|
| <code>SMC_RET_SUCCESS</code> | Function succeeded. |
| <code>SMC_RET_FAILURE</code> | Function failed. |
| <code>SMC_RET_INVALID_API_FUNCTION</code> | Invalid use of obsolete and replacement functions within the same program. |
| <code>SMC_RET_INVALID_COMMAND</code> | Instance of command does not exist. |
| <code>SMC_RET_INVALID_CONNECT</code> | Connection does not exist. |
| <code>SMC_RET_INVALID_INFOTYPE</code> | Invalid context for requested information type. |
| <code>SMC_RET_INVALID_PARAMETER</code> | Invalid parameter value. |

Examples

This example assumes that:

- An error callback function is executing.
- You have created a connection and have a *clientId*.
- The example code is being used in the context of a Monitor Client Library API callback function, which supplies the *commandId*.

```
SMC_VALUE_UNION myValue;
SMC_VALUE_UNIONP myValuePtr = &myValue;
if (smc_get_command_info(clientId,
                        commandId,
                        SMC_INFO_ERR_NUM,
                        myValuePtr,
                        NULL) != SMC_RET_SUCCESS)
{
    printf("smc_get_command_info failed\n");
}
```

```
        /* do some cleanup */  
    }
```

Usage

- For the definition of an SMC_VALUE_UNION structure, see “Union: SMC_VALUE_UNION” on page 240.
- For data of type SMC_CHARP, *stringValue* points to the value. The Monitor Client Library allocates the memory for this string and the calling application must deallocate it using *free()*.
- To retrieve just the length in bytes of a string, pass null for *infoValue* and a valid pointer for *outputLengthHandle*.
- Table 3-6 lists the command *infoType* and associated datatype:

Table 3-6: Monitor Client Library command information types

| Information type | infoValue datatype | Available |
|--------------------------------|---|-------------------------------|
| SMC_INFO_ALARM_ACTION_DATA | SMC_CHARP | In an alarm callback function |
| SMC_INFO_ALARM_ALARMID | SMC_SIZET | In an alarm callback function |
| SMC_INFO_ALARM_CURRENT_VALUE | Depends on the data item and statistic type combination. (See Chapter 2, “Data Items and Statistical Types”.) | In an alarm callback function |
| SMC_INFO_ALARM_DATAITEM | SMC_VOIDP | In an alarm callback function |
| SMC_INFO_ALARM_ROW | SMC_SIZET | In an alarm callback function |
| SMC_INFO_ALARM_THRESHOLD_VALUE | Depends on data item/statistic type combination. (See Chapter 2, “Data Items and Statistical Types”.) | In an alarm callback function |
| SMC_INFO_ALARM_TIMESTAMP | SMC_CHARP | In an alarm callback function |
| SMC_INFO_ALARM_VALUE_DATATYPE | SMC_INT | In an alarm callback function |

| Information type | infoValue datatype | Available |
|---------------------------|--------------------|-------------------------------|
| SMC_INFO_ALARM_VIEWID | SMC_SIZET | In an alarm callback function |
| SMC_INFO_ERR_MAPSEVERITY | SMC_SIZET | In an error callback function |
| SMC_INFO_ERR_MSG | SMC_CHARP | In an error callback function |
| SMC_INFO_ERR_NUM | SMC_SIZET | In an error callback function |
| SMC_INFO_ERR_SEVERITY | SMC_SIZET | In an error callback function |
| SMC_INFO_ERR_SOURCE | SMC_SIZET | In an error callback function |
| SMC_INFO_ERR_STATE | SMC_SIZET | In an error callback function |
| Valid server modes | | |
| Mode | Availability | |
| SMC_SERVER_M_LIVE | Yes | |
| SMC_SERVER_M_HISTORICAL | Yes | |

Errors This function does not employ error callback functions.

See also

`smc_create_alarm_ex`

smc_get_dataitem_type

Description Returns the datatype for the specified data item.

Syntax `SMC_RETURN_CODE smc_get_dataitem_type
(SMC_DATAITEM_STRUCTP dataItemHandle,
SMC_DATAITEM_TYPEP ptrType)`

Parameters *dataItemHandle*
pointer to data item and statistical type.

ptrType
pointer to data value type.

Return value

| Return value | Indicates |
|-----------------|---------------------|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |

Examples

```
SMC_DATAITEM_STRUCT  dataItem =
                        { SMC_NAME_PAGE_LOGICAL_READ,
                          SMC_STAT_VALUE_SAMPLE };
SMC_DATAITEM_STRUCTP dataItemHandle = &dataItem;
SMC_DATAITEM_TYPE  dataItemType;
SMC_DATAITEM_TYPEP dataItemTypeHandle = &dataItemType;

if (smc_get_dataitem_type(dataItemHandle,
                          dataItemTypeHandle) != SMC_RET_SUCCESS)
{
    printf("smc_get_dataitem_type failed\n");
    /* do some cleanup */
}
```

Usage

- The data item types are as follows:

| Data item type | Description |
|--------------------|---|
| SMC_DI_TYPE_CHARP | Pointer to a character string. |
| SMC_DI_TYPE_DATIM | Sybase date and time. |
| SMC_DI_TYPE_DOUBLE | Double-precision floating-point number. |
| SMC_DI_TYPE_ENUMS | An enumerated datatype, specific to the data item. Enumerated types are defined in the <i>mctype.sh</i> include file and in Appendix B, “Datatypes and Structures”. |
| SMC_DI_TYPE_INT | Integer. |
| SMC_DI_TYPE_LONG | Long integer. |

- If you supply a data item and statistical type that Monitor Client Library does not support, the output parameter type is set to SMC_DI_TYPE_NONE.

See also

smc_create_view

smc_get_dataitem_value

Description Returns data after a refresh. This data is returned one data item of one row at a time.

Syntax

```
SMC_RETURN_CODE smc_get_dataitem_value
(SMC_CONNECT_ID      clientId,
 SMC_VIEW_ID         viewId,
 SMC_DATAITEM_STRUCTP dataItemHandle,
 SMC_SIZET           row,
 SMC_VALUE_UNIONP    returnVal)
```

Parameters

clientId
identifies the connection.

viewId
identifies a view created on the connection.

dataItemHandle
pointer to data item and statistic type.

row
row number of requested data.

returnVal
return value that contains the value of one data item.

Return value

| Return value | Indicates |
|-------------------------|---|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |
| SMC_RET_INVALID_CONNECT | No connection exists with the specified ID. |

Examples

The following example assumes that:

- You have created a connection and have a *clientId*.
- You have created a view on that connection and have a *viewId*.
- The view contains the *dataItem* defined in the example.
- You have successfully executed a refresh call.
- The row count is greater than zero.

```
SMC_DATAITEM_STRUCT  dataItem =
    { SMC_NAME_PAGE_LOGICAL_READ,
      SMC_STAT_VALUE_SAMPLE };
SMC_DATAITEM_STRUCTP dataItemHandle = &dataItem;
SMC_VALUE_UNION  returnValue;
```

```
SMC_VALUE_UNIONP returnValueHandle = &returnValue;
if (smc_get_dataitem_value(clientId,
    viewId,
    dataItemHandle,
    0, /* row number */
    returnValueHandle) != SMC_RET_SUCCESS)
{
    printf("smc_get_dataitem_value failed\n");
    /* do some cleanup */
}
```

Usage

- The first row of data is indexed by row number zero, the second by one, and so on.
- For data of type SMC_DI_TYPE_CHARP, the Monitor Client Library allocates the memory. The calling application must deallocate the memory using free().
- See Appendix B, “Datatypes and Structures” for a listing of members in SMC_VALUE_UNION.
- See the *mctype.sh* include file or Appendix B, “Datatypes and Structures” for the values for enumerated types.

| Errors | |
|---------------------------|-----------------------------------|
| Error | Indicates |
| SMC_RET_INVALID_VIEWID | View does not exist. |
| SMC_RET_INVALID_DINAME | Invalid data item. |
| SMC_RET_INVALID_DISTAT | Invalid data item statistic type. |
| SMC_RET_INVALID_PARAMETER | Invalid parameter. |

See also

smc_refresh_ex, smc_get_dataitem_type

smc_get_row_count

Description Returns the number of rows returned by a given view after a refresh.

Syntax SMC_RETURN_CODE smc_get_row_count
(SMC_CONNECT_ID clientId,
SMC_VIEW_ID viewId,
SMC_SIZETP rowCountHandle)

Parameters *clientId*
identifies the connection.

viewId

identifies a view created on the connection.

rowCountHandle

pointer to a variable into which Monitor Client Library writes the number of rows in a view.

Return value

| Return value | Indicates |
|-------------------------|----------------------------|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |
| SMC_RET_INVALID_CONNECT | Connection does not exist. |

Examples

The following example assumes that:

- You have created a connection and have a *clientId*.
- You have created a view on that connection and have a *viewId*.
- You have successfully executed a refresh call.

```
SMC_SIZET rowCount;
SMC_SIZETP rowCountHandle = &rowCount;
if (smc_get_row_count(clientId,
                      nviewId,
                      rowCountHandle) != SMC_RET_SUCCESS)
{
    printf("smc_get_row_count failed\n");
    /* do some cleanup */
}
```

Usage

The first row of data is indexed by row number 0, the second by 1, and so on.

Valid server modes

| Mode | Availability |
|-------------------------|-----------------------|
| SMC_SERVER_M_LIVE | Yes |
| SMC_SERVER_M_HISTORICAL | Yes (during playback) |

Error

| Error | Indicates |
|------------------------|----------------------|
| SMC_RET_INVALID_VIEWID | View does not exist. |

See also

smc_refresh_ex, smc_get_dataitem_value

smc_get_version_string

Description Returns the Monitor Client Library version number.

Syntax SMC_RETURN_CODE smc_get_version_string
(SMC_CHARPP versionBuffer)

Parameters *versionBuffer*
return value that contains the version string.

Return value

| Return value | Indicates |
|-----------------|---------------------|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |

Examples

```
SMC_CHARP versionBufferHandle;
if (smc_get_version_string(&versionBufferHandle)
    != SMC_RET_SUCCESS)
{
    printf("smc_get_version_string failed\n");
    /* do some cleanup */
}
printf("%s\n", versionBufferHandle);
free(versionBufferHandle);
```

Usage

- The Monitor Client Library allocates the memory for this string. The calling application must deallocate this memory using free().
- This function does not require a connection.

smc_initiate_playback

Description Concludes the definition of views for a playback session on Historical Server, and prepares to start playback.

Syntax SMC_RETURN_CODE smc_initiate_playback
(SMC_CONNECT_ID clientId)

Parameters *clientId*
identifies the connection.

Return value

| Return value | Indicates |
|-----------------|---------------------|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |

| Return value | Indicates |
|-------------------------|----------------------------|
| SMC_RET_INVALID_CONNECT | Connection does not exist. |

Examples

This example assumes that:

- You have created a connection to Historical Server and have a *clientId*.
- You have successfully executed `smc_create_playback_session`.
- You have created at least one view on the connection.

```
if (smc_initiate_playback(clientId) !=
    SMC_RET_SUCCESS)
{
    printf("smc_initiate_playback failed\n");
    /* do some cleanup */
}
```

Usage

- The data for a playback session is defined by calls to `smc_create_view`, made after a call to `smc_create_playback_session` and before the call to `smc_initiate_playback`.
- If this playback session was defined to create a new session from playback (that is, if `smc_create_playback_session` was called with *playbackTarget* SMC_HS_TARGET_FILE), then `smc_initiate_playback` creates the new session. The application must then call `smc_terminate_playback` to conclude the playback session.
- If the playback session was defined to play back data to the application (that is, if `smc_create_playback_session` was called with *playbackTarget* SMC_HS_TARGET_CLIENT), then the application calls `smc_refresh_ex` to retrieve each playback sample, and `smc_terminate_playback` to conclude the playback session.
- After a successful call to `smc_terminate_playback`, the Historical Server connection can be used to define another playback session, or to create a recording session.

Valid server modes

| Mode | Availability |
|-------------------------|--------------|
| SMC_SERVER_M_LIVE | No |
| SMC_SERVER_M_HISTORICAL | Yes |

Errors

| Error | Indicates |
|--------------------------|----------------------|
| SMC_RET_INVALID_SVR_MODE | Invalid server mode. |
| SMC_RET_INTERNAL_ERROR | Internal error. |

See also `smc_create_view`, `smc_create_playback_session`, `smc_refresh_ex`, `smc_terminate_playback`

smc_initiate_recording

Description Completes the definition of a recording session against Historical Server, that is, an SMC_SERVER_M_HISTORICAL connection only.

Syntax `SMC_RETURN_CODE smc_initiate_recording (SMC_CONNECT_ID clientId)`

Parameters *clientId*
identifies the connection.

Return value

| Return value | Indicates |
|-------------------------|----------------------------|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |
| SMC_RET_INVALID_CONNECT | Connection does not exist. |

Examples The following example assumes that:

- You have created a connection to Historical Server and have a *clientId*.
- You have successfully executed `smc_create_recording_session`.
- You have created at least one view on the connection.

```
if (smc_initiate_recording(clientId) !=
    SMC_RET_SUCCESS)
{
    printf("smc_initiate_recording failed\n");
    /* do some cleanup */
}
```

Usage

- The data for the recording session is defined by calls to `smc_create_view` and `smc_create_filter` that are made after a call to `smc_create_recording_session` and before the call to `smc_initiate_recording`.
- After a successful call to `smc_initiate_recording`, the Historical Server connection can be used to define another recording session, or to create a playback session.

Valid server modes

| Mode | Availability |
|-------------------------|--------------|
| SMC_SERVER_M_LIVE | No |
| SMC_SERVER_M_HISTORICAL | Yes |

Errors

| Error | Indicates |
|--------------------------|----------------------|
| SMC_RET_INVALID_SVR_MODE | Invalid server mode. |
| SMC_RET_INTERNAL_ERROR | Internal error. |

See also

`smc_create_alarm_ex`, `smc_create_filter`, `smc_create_view`,
`smc_create_recording_session`, `smc_terminate_recording_session`

smc_refresh_ex

Description Obtains a sampling of data for all views on a connection.

Syntax `SMC_RETURN_CODE smc_refresh_ex`
`(SMC_CONNECT_ID clientId,`
`SMC_SIZET step)`

Parameters *clientId*
identifies the connection.

step
during playback in a Historical Server connection, allows skipping ahead a specified number of samples. Ordinarily, on playback, *step* is +1 to retrieve the next sample (negative *step* values are not allowed).

Does not apply for live connections; use SMC_UNUSED.

Return value

| Return value | Indicates |
|-------------------------|----------------------------|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |
| SMC_RET_INVALID_CONNECT | Connection does not exist. |

Examples

This example assumes that:

- You have created a connection and have a *clientId*.
- You have created at least one view on that connection.

```
if (smc_refresh_ex(clientId, SMC_UNUSED)
```

```
        != SMC_RET_SUCCESS)
    {
        printf("smc_refresh_ex failed\n");
        /* do some cleanup */
    }
```

Usage

- In a playback session, `smc_refresh_ex` must be preceded by a call to `smc_initiate_playback`.
- If you try to refresh a view at the same time someone creates a database, the refresh may fail.
- A refresh for a view may fail if one or more databases on Adaptive Server are in single-user mode.

Valid server modes

| Mode | Availability |
|-------------------------|--------------------|
| SMC_SERVER_M_LIVE | Yes |
| SMC_SERVER_M_HISTORICAL | Yes (for playback) |

Errors

| Error | Indicates |
|------------------------------|---|
| SMC_RET_INVALID_API_FUNCTION | Invalid use of obsolete and replacement functions in program. |
| SMC_RET_INVALID_SVR_MODE | Invalid server mode. |

See also

`smc_connect_ex`

smc_terminate_playback

Description

Concludes a playback session on Historical Server.

Syntax

SMC_RETURN_CODE `smc_terminate_playback`
(SMC_CONNECT_ID *clientId*)

Parameters

clientId
identifies the connection.

Return value

| Return value | Indicates |
|-------------------------|----------------------------|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |
| SMC_RET_INVALID_CONNECT | Connection does not exist. |

- Examples

This example assumes that:

 - You have created a connection to Historical Server and have a *clientId*.
 - You have successfully executed `smc_create_playback_session`.
 - You have created at least one view on the connection.
 - You have successfully executed `smc_initiate_playback`.

```

if (smc_terminate_playback(clientId)
    != SMC_RET_SUCCESS)
{
    printf("smc_terminate_playback failed\n");
    /* do some cleanup */
}

```
- Usage

 - After a successful call to `smc_terminate_playback`, the Historical Server connection can be used to create another playback session, or to define a recording session.

| Valid server modes | |
|-------------------------|--------------|
| Mode | Availability |
| SMC_SERVER_M_LIVE | No |
| SMC_SERVER_M_HISTORICAL | Yes |

| Errors | |
|--------------------------|----------------------|
| Error | Indicates |
| SMC_RET_INVALID_SVR_MODE | Invalid server mode. |
| SMC_RET_INTERNAL_ERROR | Internal error. |

See also

`smc_create_playback_session`, `smc_initiate_playback`

smc_terminate_recording

- Description

Cancels a recording session on a Historical Server connection.
- Syntax

```

SMC_RETURN_CODE smc_terminate_playback(
    SMC_CONNECT_ID clientId,
    SMC_SESSION_ID sessionId
    SMC_HS_SESS_DELETE_OPT deleteOption,
)

```
- Parameters

clientId

identifies the Monitor connection.

sessionId
identifies the recording session to cancel.

deleteOption
specifies whether Historical Server should delete the data files, if any, associated with the session. The choices are SMC_HS_DELETE_FILES and SMC_HS_RETAIN_FILES.

This parameter is ignored if the session has not been initiated or if it has not started recording.

Return value

| Return value | Indicates |
|-------------------------|------------------------------------|
| SMC_RET_SUCCESS | Function succeeded. |
| SMC_RET_FAILURE | Function failed. |
| SMC_RET_INVALID_CONNECT | Monitor connection does not exist. |

Examples

This example assumes that:

- You have created a connection to Historical Server and have a *clientId*.
- You have successfully executed `smc_create_recording_session` and have a *sessionId*.

```
if (smc_terminate_recording(  
    clientId,  
    sessionId,  
    SMC_HS_DELETE_FILES)  
    != SMC_RET_SUCCESS)  
{  
    printf("smc_terminate_recording failed\n");  
    /* do some cleanup */  
}
```

Usage

- If the recording session had already been initiated, then `smc_terminate_recording` cancels the session. If the session had been scheduled, but had not actually started recording, then `smc_terminate_recording` causes the session to be unscheduled. If the session had actually started recording, then `smc_terminate_recording` causes the session to end prematurely, that is, before the scheduled end time.
- If the recording session had not been initiated, then `smc_terminate_recording` cancels definition of the recording session. After a successful call to `smc_terminate_recording`, the HISTORICAL connection may be used to create another recording session, or to define a playback session.

Valid server modes

| Mode | Availability |
|-------------------------|--------------|
| SMC_SERVER_M_LIVE | No |
| SMC_SERVER_M_HISTORICAL | Yes |

Errors

| Error | Indicates |
|--------------------------|----------------------|
| SMC_RET_INVALID_SVR_MODE | Invalid server mode. |
| SMC_RET_INTERNAL_ERROR | Internal error. |

See also

`smc_create_recording_session`, `smc_initiate_recording`

Building a Monitor Client Library Application

This chapter contains information about building a Monitor Client Library application on the following platforms:

| Topic | Page |
|-------------------------------|------|
| Building on UNIX platforms | 180 |
| Building on Windows platforms | 182 |

This chapter describes the steps required to build a Monitor Client Library application, including:

- Compiling
- Linking
- Running

Two sample programs are provided with the Monitor Client Library:

- *testmon*, which obtains data from a Monitor Server
- *testhist*, which creates a Historical Server recording session and places data into a file

You can use the build procedures supplied with these sample applications as a model for other applications. The sample programs are discussed separately for UNIX and Windows platforms.

Note The following instructions assume that the Monitor Client Library is installed in the Sybase root directory, and that the *SYBASE* environment variable is set to this root directory.

Building on UNIX platforms

This section explains how to compile, link, run, and build the sample applications for UNIX platforms.

Compiling the application

Each source file that uses the Monitor Client Library must include the following line:

```
#include "mcpublic.h"
```

The header files for Monitor Client Library are installed, by default, in the *monclt/include* directory of the directory indicated by the *SYBASE* environment variable.

Open Client header files, which are needed for compilation, are also installed in this directory. Include this directory in the compilation command line. For example, you could enter:

```
cc -I$SYBASE/monclt/include myprog.c
```

If the header files have been installed in directories other than the default, substitute those directories in the compilation command line.

Linking the application

The Monitor Client Library is installed in the *monclt/lib* directory of the directory indicated by the *SYBASE* environment variable. In addition, Open Client libraries, which are required for linking with the Monitor Client Library, are installed in the *monclt/lib* directory. To find the names of the libraries with which you must link your application, see the *make* files supplied with the examples.

Running the application

To run a Monitor Client Library application, set the *SYBASE* environment variable to the Open Client installation directory that contains the *locales*, *charsets*, and *lib* directories. These directories are loaded during Monitor Client Library installation.

Note Adaptive Server and Monitor Server must be configured and running on your network before you run a Monitor Client Library application.

Building the sample applications

The sample programs and the procedures to build them are installed, by default, in the *\$SYBASE/sample/monclt* directory. The two versions of the build procedure are:

- *Makefile*, which uses the native ANSI compiler and linker
- *Makefile_gcc*, which uses the GNU C compiler and linker

To build and run the sample programs, use the following steps:

- 1 If the entries for the Adaptive Server, Monitor Server, and Historical Server that you intend to use with the examples do not appear in your *interfaces* file, add the entries. You can use *monclt/bin/sybinit* to edit the *interfaces* file.
- 2 Copy the sample files from the *monclt/sample* directory to another directory to keep the original sample for future reference and enable you to edit your own copy.
- 3 If you are not already there, change your directory to the directory that contains your copies of the sample files.
- 4 Edit the *example.h* file to supply the names of:
 - Adaptive Server
 - Monitor Server
 - Historical Server
 - Login name on Adaptive Server
 - Password
 - *interfaces* file location

If you are using the default *interfaces* file located in the directory indicated by the *SYBASE* environment variable, you can accept the default null string ("") for the *interfaces* file name. If you are not using the default *interfaces* file, specify the full path name of the *interfaces* file.

- 5 Set the *MONCLTLIBDIR* environment variable to the root installation directory for Monitor Client Library, which is by default, the *monclt* directory of the Sybase root installation directory:

```
setenv MONCLTLIBDIR $SYBASE/monclt
```

- 6 You can edit the *make* files and change the value of the *SYBASE* variable to point to a different Sybase root directory. By default, it points to *\$MONCLTLIBDIR*.

- 7 Use the make utility to build the test programs.

If you use the native UNIX make utility, enter:

```
make all
```

If you use the GNU compiler, enter:

```
make -f Makefile_gcc
```

- 8 Run the sample programs.

To run the program that retrieves and displays live data from Monitor Server, enter:

```
./testmon
```

To run the program that creates a recording session using Historical Server, enter:

```
./testhist
```

Building on Windows platforms

This section describes how to compile, link, run, and build the sample applications on a Windows platform.

Compiling the application

To compile a Monitor Client Library application on a Windows platform, do the following:

- 1 Include the following line in each source file that uses Monitor Client Library:

```
#include "mcpubli.c.h"
```
- 2 Include the path of the directory that contains the Monitor Client Library and Open Client header files in the list of directories (sometimes called the Include path) in which the C compiler preprocessor looks for header files. The header files for Monitor Client Library and Open Client are installed, by default, in the *C:\SYBASE\INCLUDE* directory.
- 3 Set the compiler preprocessor option to define the *_WIN* and *WIN32* preprocessor macros.
- 4 Set the code generation option to use the *__cdecl* calling convention.

Note To use a calling convention other than the default, you must declare it in each callback function that uses it.

Linking the application

The Monitor Client Library is contained in the *smcapi32.lib* file, which is installed in the *C:\SYBASE\LIB* directory.

You can specify the full path name of the library or the *smcapi32.lib* file name in the list of libraries for the linker to use for your application. However, if you include only the file name, you must include the *C:\SYBASE\LIB* directory in the list of directories in which the linker looks for libraries.

Running the application

Refer to the release bulletin for Adaptive Server Enterprise Monitor for a list of software required to run a Monitor Client Library application.

Define the *SYBASE* environment variable to indicate the directory where the Sybase client software has been installed. The *ini* directory within this directory must contain the *sql.ini* file. Use the *SQLEDIT* utility to set up this file to include the names of any Adaptive Server installations, Monitor Servers, and (optionally) Historical Servers that your application uses.

Note Adaptive Server and Monitor Server must be configured and running on your network before you run a Monitor Client Library application.

Building the sample applications

The sample programs and the build procedures to build them are installed in the *C:\SYBASE\SAMPLE\MONCLT\TESTMON* and *C:\SYBASE\SAMPLE\MONCLT\TESTHIST* directories.

For each of the sample programs, there is a project (*.mak*) file. For applications to be built using Microsoft Visual C/C++ version 4.0 and to be run under Windows NT or Windows 95 as a console application, the two project files are *TESTMO32.MAK* and *TESTHI32.MAK*.

To build and run the sample programs, use the following steps:

- 1 Modify the *PATH* environment variable to include the *C:\SYBASE\DLL* directory in which the Sybase DLLs were installed.
- 2 If you have not already done so, set the *SYBASE* environment variable to the Sybase *\SYBASE* root installation directory.
- 3 If you do not have the appropriate server names in the *sql.ini* file, add the entries for the Adaptive Server installation, Monitor Server, and Historical Server that you intend to use to the *C:\SYBASE\INI\SQL.INI* file.
- 4 Edit the *\SYBASE\SAMPLE\MONCLT\TESTMON\EXAMPLE.H* and *\SYBASE\SAMPLE\MONCLT\TESTHIST\EXAMPLE.H* files to supply the names of the Adaptive Server, Monitor Server, Historical Server (for *TESTHIST* only), login name on Adaptive Server, and password.
- 5 Open the project (*.mak*) file for the sample application you want to build.
 - To use the program that tests a live connection to Monitor Server, enter:

```
\SYBASE\SAMPLE\MONCLT\TESTMON\TESTMO32.MAK
```
 - To use the program that tests Historical Server, enter:

`\SYBASE\SAMPLE\MONCLT\TESTHIST\TESTHI32.MAK`

- 6 If the Monitor Client Library is installed in a directory other than `\SYBASE`:
 - Modify the compiler preprocessor option to include the *INCLUDE* subdirectory of the installation directory, instead of the default `\SYBASE\INCLUDE` directory, in the list of directories in which the C compiler preprocessor looks for header files.
 - Edit the list of libraries for the linker to use for the application so that it specifies the full path name of the library, instead of the `\SYBASE\LIB\SMCAPI32.LIB` default directory path name.
- 7 Build the project.
- 8 Run the application.

To run applications under Windows NT or Windows 95, enter the name of the executable program from a Command Prompt window. For example:

`C:\SYBASE\SAMPLE\MONCLT\TESTMON\WinDebug\TESTMO32`

Monitor Client Library Configuration Instructions

This chapter describes the installation and configuration process for Monitor Client Library.

| Topic | Page |
|---|------|
| Loading Monitor Client Library | 187 |
| Results of the load | 188 |
| Confirming your login account and permissions | 188 |
| Modifying the interfaces file | 188 |
| Setting up the user environment | 190 |
| Using Monitor Client Library | 191 |

Loading Monitor Client Library

To move the Monitor Client Library files from the distribution media onto your machine, use Studio Installer. This utility allows you to load all of the products you have ordered onto one machine in one Studio Installer session or to distribute your software among different licensed machines by running separate Studio Installer sessions.

Using Studio Installer

If you have not already done so, follow the instructions in the installation guide to load Monitor Client Library onto your machine.

After loading the software, return to this chapter to complete the installation and configuration of Monitor Client Library.

Results of the load

The Studio Installer utility places the Monitor Client Library software in the load directory you specified to Studio Installer during the installation process. The default load directory is the `$SYBASE` directory.

The load directory contains all software and other files for Monitor Client Library, including the *locales* and *charsets* subdirectories at the correct version level for Monitor Client Library.

Confirming your login account and permissions

To perform the tasks described in this chapter, you must be logged in using the “sybase” account or some other account that has read, write, and search (execute) permissions on the load directory. The load directory is the directory name you supplied to Studio Installer when you loaded the Monitor Client Library software onto your machine. The default load directory is the `$SYBASE` directory.

Modifying the *interfaces* file

Before a Monitor Client Library application can run, it must have access to an *interfaces* file that contains entries for Adaptive Server Enterprise Monitor. The *interfaces* file can exist on a local or remote machine, so long as the Monitor Client Library application has access to the file system containing the *interfaces* file.

If an *interfaces* file does not exist on a machine where a Monitor Client Library application will run and an *interfaces* file is not accessible remotely, you must create one.

The *interfaces* file accessed by a Monitor Client Library application must contain entries for the following servers:

- The Adaptive Server installations being monitored
- The Monitor Server(s) that Monitor Viewer is using
- Optionally, the Monitor Historical Server if one is being used

The entries that you add to the *interfaces* file accessed by the Monitor Client Library application must match the entries that already exist in the *interfaces* file for the servers, on the server machine. Those entries define the server names, their host machine names, and their port numbers. You must use the same values on the client machine. See the person who installed Monitor Server and Monitor Historical Server to obtain the entries for the servers.

The general format for additions to a client *interfaces* file is:

```
sql_server_name
    query entry
    master entry
monitor_server_name
    query entry
    master entry
historical_server_name
    query entry
    master entry
```

Use the sybinit utility or a text editor to add entries to the *interfaces* file.

Note Before invoking sybinit, make sure that your *SYBASE* environment variable points to the directory containing the *interfaces* file that you want to change or, if the file does not yet exist on your machine, the directory where you want it to reside.

If you use a text editor to update the *interfaces* file, entries must comply with the following rules:

- The entry cannot contain blank lines.
- The *server_name* line must start in the first column of the *interfaces* file.
- The entries for query and master must have one tab preceding them. You must indent the query and master lines using the Tab key; do not use the space bar to indent these two lines.

If you use sybinit to edit the *interfaces* file, the utility enforces these rules.

For information about editing *interfaces* files, specifics about the *interfaces* file format, and details about parameters within an *interfaces* file entry, see *Configuring Adaptive Server Enterprise* for your platform.

Setting up the user environment

On start-up, a Monitor Client Library application must:

- The correct version of the *locales* and *charsets* directories
- An *interfaces* file

The *SYBASE* environment variable defines the location of the *locales* and *charsets* directories. The *SYBASE* variable also defines the default location of the *interfaces* file; however, the Monitor Client Library application might need to override that default location.

Setting the *SYBASE* environment variable

When a user starts a Monitor Client Library application, the directory pointed to by the *SYBASE* environment variable must contain the correct version of the *locales* and *charsets* directories. Therefore, users must set their *SYBASE* environment variable to point to the *monclt* subdirectory of the load directory (the directory where the Studio Installer placed Monitor Client Library software).

Overriding the default location of the *interfaces* file

The default location of the *interfaces* file is the directory pointed to by the *SYBASE* environment variable. Since the *SYBASE* environment variable must point to the load directory, then one of the following statements also must be true when users run a Monitor Client Library application:

- The *interfaces* file must be located in the load directory, or
- The Monitor Client Library application code must override the default location of the *interfaces* file.

To override the default location, the Monitor Client Library application must call the `smc_connect` function, specifying an explicit value in the *interfaceFile* parameter. In most cases, it would be appropriate to obtain the value of the *interfaceFile* parameter from the user at start-up time, as a command-line argument, from an X resource file, or from an interactive dialog box.

For more information about the `smc_connect` function, see the *Adaptive Server Enterprise Monitor Client Library Programmer's Guide*.

Using Monitor Client Library

After completing the installation and setting up the user environment, you can build and run the sample programs provided. For more details on the sample programs, see the *Adaptive Server Enterprise Monitor Client Library Programmer's Guide*.

If you have not already done so, read the *Adaptive Server Enterprise Monitor Client Library Release Bulletin* for your platform.

Notes

- Adaptive Server and Monitor Server must be configured and running on your network before you run a Monitor Client Library application.
- For maximum responsiveness, Sybase recommends that Monitor Client applications run on different machines from the one on which Adaptive Server and Monitor Server are running.

Examples of Views

This appendix contains examples of views. These views also appear in the sample views file installed with Historical Server.

You may find that some of these views collect exactly the information that you are interested in, while others can serve as templates for building the views that you need.

Some of the sample views differ from one another only in the time interval over which the data is accumulated (either the duration of the most recent sample interval or the entire session). Other views may contain similar data items, but in a different order. The order in which data items appear in a view is significant because the data is sorted according to the key field. The first key field appears in a view's definition and acts as the primary sort key, the second key field is the secondary sort key, and so on.

```
#include mcpublic.h

SMC_VOID
ErrorCallback(
    SMC_SIZET    id,
    SMC_SIZET    error_number,
    SMC_SIZET    severity,
    SMC_SIZET    map_severity,
    SMC_SIZET    source,
    SMC_CCHARP   error_msg,
    SMC_SIZET    state);

SMC_VOID
RefreshCallback(
    SMC_SIZET    id,
    SMC_VOIDP    user_msg,
    SMC_CHARP    msg);
SMC_CHARP
SMC_DATAITEM_NAME    value);

SMC_CHARP
LookupDataItemStat(
    SMC_DATAITEM_STATTYPE    value);
```

```
SMC_CHARP
LookupLockResult(
SMC_LOCK_RESULT    value);

SMC_CHARP
LookupLockResultSummary(
SMC_LOCK_RESULT_SUMMARY    value);

SMC_CHARP
LookupLockStatus(
SMC_LOCK_STATUS    value);

SMC_CHARP
LookupLockType(
SMC_LOCK_TYPE    value);

SMC_CHARP
LookupObjectType(
SMC_OBJ_TYPE    value);

SMC_CHARP
LookupProcessState(
SMC_PROCESS_STATE    value);
SMC_INT
main(
SMC_INT    argc,
SMC_CHARP    argv[])
{
```

Cache performance summary

This view shows the overall effectiveness of Adaptive Server caches during the most recent sample interval. It shows the percentage of data page reads that were satisfied from Adaptive Server data caches and the percentage of requests for procedure execution that were satisfied from Adaptive Server procedure cache.

```
SMC_SIZET cache_perf_sum_count = 2;
SMC_DATAITEM_STRUCT cache_perf_sum_view[] = {
{ SMC_NAME_PAGE_HIT_PCT,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_STP_HIT_PCT,          SMC_STAT_VALUE_SAMPLE }
};
```

Current statement summary

This view displays information about the statement that is currently being executed by Adaptive Server whether it is part of a stored procedure or batch text. Use a view such as this if you are trying to determine what an application is doing at a particular point in its execution.

```
SMC_SIZE cur_stmt_act_count = 11;
SMC_DATAITEM_STRUCT cur_stmt_act_view[] = {
{ SMC_NAME_SPID, SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_CUR_STMT_ACT_STP_DB_ID, SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_CUR_STMT_ACT_STP_DB_NAME, SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_CUR_STMT_ACT_STP_ID, SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_CUR_STMT_ACT_STP_NAME, SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_CUR_STMT_ACT_STP_TEXT, SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_CUR_STMT_BATCH_ID, SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_CUR_STMT_CONTEXT_ID, SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_CUR_STMT_NUM, SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_CUR_STMT_QUERY_PLAN_TEXT, SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_CUR_STMT_START_TIME, SMC_STAT_VALUE_SAMPLE },
};
```

Database object lock status

This view shows the status of locks on database objects that are held or being requested by Adaptive Server processes as of the end of the most recent sample interval. Each lock is identified by:

- The name and ID of the object being locked
- The name and ID of the database that contains that object
- The page number to which the lock applies (if it is a page lock)

Each Adaptive Server process associated with the lock is also identified by its login name, Process ID and Kernel Process ID. The type of lock is shown, together with the current status of the lock and an indication of whether or not this is a demand lock.

If the lock is being requested by the process, the amount of time that this process has waited to acquire the lock and the Process ID of the process that already holds the lock are shown. If the process already holds the lock, the count of other processes waiting to acquire that lock is shown.

```
SMC_SIZET object_lock_status_count = 14;
SMC_DATAITEM_STRUCT object_lock_status_view[] = {
{ SMC_NAME_DB_ID,                SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_DB_NAME,             SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_OBJ_ID,              SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_OBJ_NAME,            SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_NUM,            SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_LOGIN_NAME,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_SPID,                 SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_KPID,                 SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_LOCK_TYPE,           SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_LOCK_STATUS,         SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_DEMAND_LOCK,         SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_TIME_WAITED_ON_LOCK, SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_BLOCKING_SPID,       SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_LOCKS_BEING_BLOCKED_CNT, SMC_STAT_VALUE_SAMPLE }
};
```

Database object page I/O

This view shows the objects in Adaptive Server databases and the page I/Os associated with them. It shows the Adaptive Server database name and ID, and the object names and IDs within each database. For each object, this view shows the associated logical reads, physical reads, and page writes for both the most recent sample interval and for the session.

```
SMC_SIZET object_page_io_count = 10;
SMC_DATAITEM_STRUCT object_page_io_view[] = {
{ SMC_NAME_DB_ID,                SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_DB_NAME,             SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_OBJ_ID,              SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_OBJ_NAME,            SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_LOGICAL_READ,    SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_PHYSICAL_READ,   SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_WRITE,           SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_LOGICAL_READ,    SMC_STAT_VALUE_SESSION },
{ SMC_NAME_PAGE_PHYSICAL_READ,   SMC_STAT_VALUE_SESSION },
{ SMC_NAME_PAGE_WRITE,           SMC_STAT_VALUE_SESSION }
};
```

Data cache activity for individual caches

This view shows information about the performance of individual data caches.

For each named cache, including the default data cache, configured in Adaptive Server, this view collects the cache's name and the percentage of page reads for objects bound to the cache that were satisfied from the cache since the start of the recording session.

This view also shows the:

- Efficiency of the cache's use of space
- Percentage of times when an attempt to acquire the cache's spinlock was forced to wait, since the start of the session
- Number of cache hits and misses for the session

```
SMC_SIZET data_cache_activity_count = 7;
SMC_DATAITEM_STRUCT data_cache_activity_view[] = {
{ SMC_NAME_DATA_CACHE_NAME,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_DATA_CACHE_ID,            SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_DATA_CACHE_HIT_PCT,        SMC_STAT_VALUE_SESSION },
{ SMC_NAME_DATA_CACHE_EFFICIENCY,     SMC_STAT_VALUE_SESSION },
{ SMC_NAME_DATA_CACHE_CONTENTION,     SMC_STAT_RATE_SESSION },
{ SMC_NAME_DATA_CACHE_HIT,            SMC_STAT_VALUE_SESSION },
{ SMC_NAME_DATA_CACHE_MISS,           SMC_STAT_RATE_SESSION }

};
```

Data cache statistics for session

This view shows the effectiveness of the data caches of Adaptive Server since the start of the session. It shows the:

- Percentage of requests for page reads that were satisfied from cache for the session
- Number of logical reads, physical reads, and page writes for the session
- Rate of logical reads, physical reads, and page writes for the session

```
SMC_SIZET session_page_cache_stats_count = 7;
SMC_DATAITEM_STRUCT session_page_cache_stats_view[] = {
{ SMC_NAME_PAGE_HIT_PCT,              SMC_STAT_VALUE_SESSION },
```

```
{ SMC_NAME_PAGE_LOGICAL_READ,      SMC_STAT_VALUE_SESSION },
{ SMC_NAME_PAGE_LOGICAL_READ,      SMC_STAT_RATE_SESSION },
{ SMC_NAME_PAGE_PHYSICAL_READ,     SMC_STAT_VALUE_SESSION },
{ SMC_NAME_PAGE_PHYSICAL_READ,     SMC_STAT_RATE_SESSION },
{ SMC_NAME_PAGE_WRITE,             SMC_STAT_VALUE_SESSION },
{ SMC_NAME_PAGE_WRITE,             SMC_STAT_RATE_SESSION }
};
```

Data cache statistics for sample interval

This view shows the effectiveness of the data caches of Adaptive Server for the most recent sample interval. It shows the:

- Percentage of requests for page reads that were satisfied from cache for the most recent sample interval
- Number of logical reads, physical reads, and page writes for the most recent sample interval
- Rate of logical reads, physical reads, and page writes for the most recent sample interval

```
SMC_SIZET sample_page_cache_stats_count = 7;
SMC_DATAITEM_STRUCT sample_page_cache_stats_view[] = {
{ SMC_NAME_PAGE_HIT_PCT,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_LOGICAL_READ,     SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_LOGICAL_READ,     SMC_STAT_RATE_SAMPLE },
{ SMC_NAME_PAGE_PHYSICAL_READ,    SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_PHYSICAL_READ,    SMC_STAT_RATE_SAMPLE },
{ SMC_NAME_PAGE_WRITE,           SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_WRITE,           SMC_STAT_RATE_SAMPLE }
};
```

Device I/O for session

This view shows the I/O activity that occurred on Adaptive Server database devices since the start of the session. It identifies each device by name. Device I/O levels are presented in two ways: as counts of total device I/Os, reads and writes since the start of the session, and also as overall rates of total I/Os, reads and writes per second since the session began.

```
SMC_SIZET session_device_io_count = 7;
SMC_DATAITEM_STRUCT session_device_io_view[] = {
{ SMC_NAME_DEV_NAME,      SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_DEV_READ,      SMC_STAT_VALUE_SESSION },
{ SMC_NAME_DEV_WRITE,     SMC_STAT_VALUE_SESSION },
{ SMC_NAME_DEV_IO,        SMC_STAT_VALUE_SESSION },
{ SMC_NAME_DEV_READ,      SMC_STAT_RATE_SESSION },
{ SMC_NAME_DEV_WRITE,     SMC_STAT_RATE_SESSION },
{ SMC_NAME_DEV_IO,        SMC_STAT_RATE_SESSION }
};
```

Device I/O for sample interval

This view shows the I/O activity that occurred on Adaptive Server database devices during the most recent sample interval. It identifies each device by name. Device I/O levels are presented in two ways: as counts of total device I/Os, reads and writes during the most recent sample interval, and also as rates of total I/Os, reads and writes per second during the sample interval.

```
SMC_SIZET sample_device_io_count = 7;
SMC_DATAITEM_STRUCT sample_device_io_view[] = {
{ SMC_NAME_DEV_NAME,      SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_DEV_IO,        SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_DEV_READ,      SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_DEV_WRITE,     SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_DEV_IO,        SMC_STAT_RATE_SAMPLE },
{ SMC_NAME_DEV_READ,      SMC_STAT_RATE_SAMPLE },
{ SMC_NAME_DEV_WRITE,     SMC_STAT_RATE_SAMPLE }
};
```

Device I/O performance summary

This view shows reads and writes to database devices by Adaptive Server, since the start of the session. It shows the:

- Overall rate of reads and writes to database devices since the start of the session
- Most active database device for that time period

- Rate of reads and writes to the most active device

```
SMC_SIZET device_perf_sum_count = 3;
SMC_DATAITEM_STRUCT device_perf_sum_view[] = {
{ SMC_NAME_DEV_IO,                SMC_STAT_RATE_SESSION },
{ SMC_NAME_MOST_ACT_DEV_NAME,     SMC_STAT_VALUE_SESSION },
{ SMC_NAME_MOST_ACT_DEV_IO,       SMC_STAT_RATE_SESSION }
};
```

Engine activity

This view shows the level of activity for each active Adaptive Server engine during the most recent sample interval. This view shows, for each engine, the:

- Percentage of the sample interval when that engine used the CPU
- Number of lock requests
- Number of logical page reads, physical page reads, and page writes that were generated by the engine during the sample interval

```
SMC_SIZET engine_activity_count = 6;
SMC_DATAITEM_STRUCT engine_activity_view[] = {
{ SMC_NAME_ENGINE_NUM,            SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_CPU_BUSY_PCT,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_LOCK_CNT,              SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_LOGICAL_READ,     SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_PHYSICAL_READ,    SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_WRITE,            SMC_STAT_VALUE_SAMPLE }
};
```

Lock performance summary

This view shows the total number of locks of each type requested and granted during the most recent sample interval.

```
SMC_SIZET lock_perf_sum_count = 3;
SMC_DATAITEM_STRUCT lock_perf_sum_view[] = {
{ SMC_NAME_LOCK_TYPE,             SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_LOCK_RESULT_SUMMARY,   SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_LOCK_CNT,              SMC_STAT_VALUE_SAMPLE }
};
```

```
};
```

Network activity for session

This view shows the network activity over all Adaptive Server network connections since the start of the session. It shows the:

- Default packet size
- Maximum packet size
- Average packet sizes sent and received since the start of the session
- Number of packets sent
- Number of packets received
- The rate at which packets were sent and received
- Number of bytes sent
- Number of bytes received
- Rate at which bytes were sent and received

```
SMC_SIZET session_network_activity_count = 12;
SMC_DATAITEM_STRUCT session_network_activity_view[] = {
{ SMC_NAME_NET_DEFAULT_PKT_SIZE,    SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_NET_MAX_PKT_SIZE,        SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_NET_PKT_SIZE_SENT,       SMC_STAT_VALUE_SESSION },
{ SMC_NAME_NET_PKT_SIZE_RCVD,       SMC_STAT_VALUE_SESSION },
{ SMC_NAME_NET_PKTS_SENT,           SMC_STAT_VALUE_SESSION },
{ SMC_NAME_NET_PKTS_RCVD,           SMC_STAT_VALUE_SESSION },
{ SMC_NAME_NET_PKTS_SENT,           SMC_STAT_RATE_SESSION },
{ SMC_NAME_NET_PKTS_RCVD,           SMC_STAT_RATE_SESSION },
{ SMC_NAME_NET_BYTES_SENT,          SMC_STAT_VALUE_SESSION },
{ SMC_NAME_NET_BYTES_RCVD,          SMC_STAT_VALUE_SESSION },
{ SMC_NAME_NET_BYTES_SENT,          SMC_STAT_RATE_SESSION },
{ SMC_NAME_NET_BYTES_RCVD,          SMC_STAT_RATE_SESSION }
};
```

Network activity for sample interval

This view shows the network activity over all Adaptive Server network connections during the most recent sample interval. It shows the:

- Default packet size
- Maximum packet size
- Average packet sizes sent and received for the sample interval
- Number of packets sent
- Number of packets received
- Rate at which packets were sent and received
- Number of bytes sent
- Number of bytes received
- Rate at which bytes were sent and received

```
SMC_SIZET sample_network_activity_count = 12;
SMC_DATAITEM_STRUCT sample_network_activity_view[] = {
{ SMC_NAME_NET_DEFAULT_PKT_SIZE,    SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_NET_MAX_PKT_SIZE,        SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_NET_PKT_SIZE_SENT,        SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_NET_PKT_SIZE_RCVD,        SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_NET_PKTS_SENT,            SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_NET_PKTS_RCVD,            SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_NET_PKTS_SENT,            SMC_STAT_RATE_SAMPLE },
{ SMC_NAME_NET_PKTS_RCVD,            SMC_STAT_RATE_SAMPLE },
{ SMC_NAME_NET_BYTES_SENT,           SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_NET_BYTES_RCVD,           SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_NET_BYTES_SENT,           SMC_STAT_RATE_SAMPLE },
{ SMC_NAME_NET_BYTES_RCVD,           SMC_STAT_RATE_SAMPLE },
};
```

Network performance summary

This view shows the rate of Adaptive Server activity over all its network connections during the most recent sample interval. It shows the number of bytes per second that were received by and sent by Adaptive Server during the interval.

```
SMC_SIZET network_perf_sum_count = 2;
SMC_DATAITEM_STRUCT network_perf_sum_view[] = {
{ SMC_NAME_NET_BYTES_RCVD,          SMC_STAT_RATE_SAMPLE },
{ SMC_NAME_NET_BYTES_SENT,          SMC_STAT_RATE_SAMPLE }
};
```

Procedure cache statistics for session

This view shows the effectiveness of the procedure cache of Adaptive Server since the start of the session. It shows the:

- Percentage of requests for stored procedure executions that were satisfied by the procedure cache
- Number of logical reads and physical reads of stored procedures since the start of the session
- Overall rate of logical and physical reads of stored procedures since the start of the session

```
SMC_SIZET session_procedure_cache_stats_count = 5;
SMC_DATAITEM_STRUCT session_procedure_cache_stats_view[] = {
{ SMC_NAME_STP_HIT_PCT,          SMC_STAT_VALUE_SESSION },
{ SMC_NAME_STP_LOGICAL_READ,    SMC_STAT_VALUE_SESSION },
{ SMC_NAME_STP_LOGICAL_READ,    SMC_STAT_RATE_SESSION },
{ SMC_NAME_STP_PHYSICAL_READ,   SMC_STAT_VALUE_SESSION },
{ SMC_NAME_STP_PHYSICAL_READ,   SMC_STAT_RATE_SESSION }
};
```

Procedure cache statistics for sample interval

This view shows the effectiveness of the procedure cache of Adaptive Server for the most recent sample interval. It shows the:

- Percentage of requests for stored procedure executions that were satisfied by the procedure cache for the most recent sample interval
- Number of logical reads and physical reads of stored procedures during the most recent sample interval

- Rate of logical and physical reads of stored procedures for the most recent sample interval

```
SMC_SIZET sample_procedure_cache_stats_count = 5;
SMC_DATAITEM_STRUCT sample_procedure_cache_stats_view[] = {
{ SMC_NAME_STP_HIT_PCT,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_STP_LOGICAL_READ,    SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_STP_LOGICAL_READ,    SMC_STAT_RATE_SAMPLE },
{ SMC_NAME_STP_PHYSICAL_READ,   SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_STP_PHYSICAL_READ,   SMC_STAT_RATE_SAMPLE }
};
```

Procedure page I/O

This view shows page I/Os that occurred while running stored procedures during the most recent sample interval. For each stored procedure that generated page I/Os during the sample interval, it shows the stored procedure name and ID, together with the name and ID of the database that contains the procedure. If page I/Os were produced when no stored procedure was active, those I/Os are associated with procedure ID and database ID values of zero.

This view also shows, on a per stored procedure level:

- Total page I/Os
- Percentage of page I/O requests that could be satisfied by Adaptive Server data caches
- Number of logical reads, physical reads, and page writes generated while executing the stored procedures during the most recent sample interval.

```
SMC_SIZET procedure_page_cache_io_count = 9;
SMC_DATAITEM_STRUCT procedure_page_cache_io_view[] = {
{ SMC_NAME_ACT_STP_DB_NAME,      SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_ACT_STP_DB_ID,        SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_ACT_STP_NAME,         SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_ACT_STP_ID,           SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_IO,              SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_HIT_PCT,         SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_LOGICAL_READ,    SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_PHYSICAL_READ,   SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_WRITE,           SMC_STAT_VALUE_SAMPLE }
};
```

Process activity

This view shows the CPU use, page I/Os, and current process state for all processes in Adaptive Server.

For each process in the most recent sample interval it shows the:

- Login name
- Process ID
- Kernel Process ID
- Current process state.

The view also presents each process's connect time, total page I/Os and CPU usage time, accumulated since the start of the session.

```
SMC_SIZET process_activity_count = 7;
SMC_DATAITEM_STRUCT process_activity_view[] = {
{ SMC_NAME_LOGIN_NAME,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_SPID,                SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_KPID,                SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_CONNECT_TIME,        SMC_STAT_VALUE_SESSION },
{ SMC_NAME_PAGE_IO,             SMC_STAT_VALUE_SESSION },
{ SMC_NAME_CPU_TIME,            SMC_STAT_VALUE_SESSION },
{ SMC_NAME_CUR_PROC_STATE,      SMC_STAT_VALUE_SAMPLE }
};
```

Process database object page I/O

This view shows the page I/Os by database object for each Adaptive Server process. For each process that had page I/Os during the most recent sample interval it shows the:

- Login name
- Process ID
- Kernel Process ID

For each such process and for each database object it accessed, the view shows the:

- Object name
- Object ID

- Database name and ID
- Page I/Os

The view also shows the total page I/Os, the percentage of page I/O requests that could be satisfied by Adaptive Server cache, and the number of logical reads, physical reads, and page writes for the most recent sample interval.

```
SMC_SIZET process_object_page_io_count = 13;
SMC_DATAITEM_STRUCT process_object_page_io_view[] = {
{ SMC_NAME_LOGIN_NAME,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_SPID,                SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_KPID,                SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_DB_NAME,             SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_DB_ID,               SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_OBJ_NAME,            SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_OBJ_ID,              SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_OBJ_TYPE,            SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_IO,             SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_HIT_PCT,        SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_LOGICAL_READ,   SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_PHYSICAL_READ,  SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_WRITE,         SMC_STAT_VALUE_SAMPLE }
};
```

Process detail for locks

This view shows the status of locks held or being requested by Adaptive Server processes as of the end of the most recent sample interval. Each lock is identified by:

- Login name
- Process ID
- Kernel Process ID of the Adaptive Server process associated with the lock
- Name and ID of the object being locked
- Name and ID of the database that contains that object
- Page number to which the lock applies (if it is a page lock)
- Current status of each lock
- Indication of whether or not this is a demand lock

If the lock is being requested by the process, the amount of time that this process has waited to acquire the lock and the Process ID of the process that holds the lock are shown. If the process holds the lock, the count of other processes waiting to acquire that lock is shown.

```
SMC_SIZET process_detail_locks_count = 13;
SMC_DATAITEM_STRUCT process_detail_locks_view[] = {
{ SMC_NAME_LOGIN_NAME,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_SPID,                SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_KPID,                SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_DB_NAME,             SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_DB_ID,               SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_OBJ_NAME,            SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_OBJ_ID,              SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_NUM,            SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_LOCK_STATUS,         SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_DEMAND_LOCK,         SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_TIME_WAITED_ON_LOCK, SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_BLOCKING_SPID,       SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_LOCKS_BEING_BLOCKED_CNT, SMC_STAT_VALUE_SAMPLE }
};
```

Process detail page I/O

This view shows the page I/Os for each Adaptive Server process in detail. It shows the following as of the end of the most recent sample interval:

- Login name
- Process ID
- Kernel Process ID
- Process state and current engine are shown for each Adaptive Server process

The view shows the percentage of page I/O requests that could be satisfied by Adaptive Server data caches, both for the sample interval and since the start of the session. It also shows the number of logical reads, physical reads, and page writes since the start of the session.

```
SMC_SIZET process_detail_io_count = 12;
SMC_DATAITEM_STRUCT process_detail_io_view[] = {
{ SMC_NAME_LOGIN_NAME,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_SPID,                SMC_STAT_VALUE_SAMPLE },
```

```
{ SMC_NAME_KPID,                SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_CUR_PROC_STATE,      SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_CUR_ENGINE,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_CONNECT_TIME,        SMC_STAT_VALUE_SESSION },
{ SMC_NAME_CPU_TIME,            SMC_STAT_VALUE_SESSION },
{ SMC_NAME_PAGE_HIT_PCT,        SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_HIT_PCT,        SMC_STAT_VALUE_SESSION },
{ SMC_NAME_PAGE_LOGICAL_READ,   SMC_STAT_VALUE_SESSION },
{ SMC_NAME_PAGE_PHYSICAL_READ,  SMC_STAT_VALUE_SESSION },
{ SMC_NAME_PAGE_WRITE,          SMC_STAT_VALUE_SESSION }
};
```

Process locks

This view shows the count of lock requests for every process in Adaptive Server that generated lock requests during the most recent sample interval.

```
SMC_SIZET process_lock_count = 4;
SMC_DATAITEM_STRUCT process_lock_view[] = {
{ SMC_NAME_LOGIN_NAME,    SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_SPID,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_KPID,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_LOCK_CNT,      SMC_STAT_VALUE_SAMPLE }
};
```

Process page I/O

This view summarizes the page I/Os for each Adaptive Server process for the most recent sample. For each process in Adaptive Server that generated page I/Os during the interval, it shows the login name, Process ID, and Kernel Process ID.

This view also shows, for each process:

- Total page I/Os
- Percentage of page I/O requests that could be satisfied by Adaptive Server data caches
- Number of logical reads, physical reads, and writes for the most recent sample interval

```
SMC_SIZET process_page_io_count = 8;
SMC_DATAITEM_STRUCT process_page_io_view[] = {
{ SMC_NAME_LOGIN_NAME,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_SPID,               SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_KPID,               SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_IO,            SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_HIT_PCT,       SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_LOGICAL_READ,  SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_PHYSICAL_READ, SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_WRITE,         SMC_STAT_VALUE_SAMPLE }
};
```

Process state summary

This view shows the number of processes that were in each process state at the end of the most recent sample interval.

```
SMC_SIZET process_perf_sum_count = 2;
SMC_DATAITEM_STRUCT process_perf_sum_view[] = {
{ SMC_NAME_PROC_STATE,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PROC_STATE_CNT,      SMC_STAT_VALUE_SAMPLE }
};
```

Process stored procedure page I/O

This view shows the page I/Os associated with stored procedure executions by Adaptive Server processes. It shows the login name, Process ID, and Kernel Process ID for each process that generated page I/Os during the sample interval.

For each process and stored procedure that generated page I/Os, it shows the name and ID of the database that contains the stored procedure, and the name and ID of the procedure.

For the most recent sample interval, the view shows the:

- Total page I/Os
- Percentage of page I/O requests that could be satisfied from data caches
- Number of logical reads, physical reads, and page writes

```
SMC_SIZET process_procedure_page_io_count = 12;
SMC_DATAITEM_STRUCT process_procedure_page_io_view[] = {
{ SMC_NAME_LOGIN_NAME,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_SPID,                SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_KPID,                SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_ACT_STP_DB_NAME,     SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_ACT_STP_DB_ID,       SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_ACT_STP_NAME,        SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_ACT_STP_ID,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_IO,             SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_HIT_PCT,        SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_LOGICAL_READ,   SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_PHYSICAL_READ,  SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_PAGE_WRITE,          SMC_STAT_VALUE_SAMPLE }
};
```

Server performance summary

This view shows overall Adaptive Server performance. It shows the:

- Number of lock requests per second
- Percentage of the sample interval when Adaptive Server was busy
- Number of transactions processed per second
- Number of times Adaptive Server detected a deadlock during the most recent sample interval

```
SMC_SIZET server_perf_sum_count = 4;
SMC_DATAITEM_STRUCT server_perf_sum_view[] = {
{ SMC_NAME_LOCK_CNT,            SMC_STAT_RATE_SAMPLE },
{ SMC_NAME_CPU_BUSY_PCT,        SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_XACT,                SMC_STAT_RATE_SAMPLE },
{ SMC_NAME_DEADLOCK_CNT,        SMC_STAT_VALUE_SAMPLE }
};
```

Stored procedure activity

This view shows stored procedure activity for procedure statements. Each statement of any stored procedure that was executed during the most recent sample interval is identified by:

- Name and ID of the database that contains the procedure
- Name and ID of the procedure
- Relative number of the statement within the stored procedure
- Line of the procedure's text on which the statement begins

The view shows the:

- Number of times each statement was executed, both during the most recent sample interval and since the start of the session
- Average elapsed time needed to execute the statement, both for the sample interval and for the session so far

```
SMC_SIZET procedure_activity_count = 10;
SMC_DATAITEM_STRUCT procedure_activity_view[] = {
{ SMC_NAME_ACT_STP_DB_ID, SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_ACT_STP_DB_NAME,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_ACT_STP_ID,              SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_ACT_STP_NAME,            SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_STP_LINE_NUM,            SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_STP_STMT_NUM,            SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_STP_NUM_TIMES_EXECUTED,  SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_STP_NUM_TIMES_EXECUTED,  SMC_STAT_VALUE_SESSION },
{ SMC_NAME_STP_ELAPSED_TIME,        SMC_STAT_AVG_SAMPLE },
{ SMC_NAME_STP_ELAPSED_TIME,        SMC_STAT_AVG_SESSION }
};
```

Transaction activity

This view shows the transaction activity that occurred in the Adaptive Server, both for the sample interval and the session.

```
SMC_SIZET transaction_activity_count = 20;
SMC_DATAITEM_STRUCT transaction_activity_view[] = {
{ SMC_NAME_XACT,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_XACT_DELETE,   SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_XACT_INSERT,   SMC_STAT_VALUE_SAMPLE },
};
```

```
{ SMC_NAME_XACT_UPDATE,          SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_XACT_UPDATE_DIRECT,   SMC_STAT_VALUE_SAMPLE },
{ SMC_NAME_XACT,                 SMC_STAT_VALUE_SESSION },
{ SMC_NAME_XACT_DELETE,         SMC_STAT_VALUE_SESSION },
{ SMC_NAME_XACT_INSERT,         SMC_STAT_VALUE_SESSION },
{ SMC_NAME_XACT_UPDATE,         SMC_STAT_VALUE_SESSION },
{ SMC_NAME_XACT_UPDATE_DIRECT,   SMC_STAT_VALUE_SESSION },
{ SMC_NAME_XACT,                 SMC_STAT_RATE_SAMPLE },
{ SMC_NAME_XACT_DELETE,         SMC_STAT_RATE_SAMPLE },
{ SMC_NAME_XACT_INSERT,         SMC_STAT_RATE_SAMPLE },
{ SMC_NAME_XACT_UPDATE,         SMC_STAT_RATE_SAMPLE },
{ SMC_NAME_XACT_UPDATE_DIRECT,   SMC_STAT_RATE_SAMPLE },
{ SMC_NAME_XACT,                 SMC_STAT_RATE_SESSION },
{ SMC_NAME_XACT_DELETE,         SMC_STAT_RATE_SESSION },
{ SMC_NAME_XACT_INSERT,         SMC_STAT_RATE_SESSION },
{ SMC_NAME_XACT_UPDATE,,        SMC_STAT_RATE_SESSION },
{ SMC_NAME_XACT_UPDATE_DIRECT,   SMC_STAT_RATE_SESSION }
}; SMC_SIZET num_views = 27;
SMC_SIZET* view_count = (SMC_SIZET*) malloc (sizeof(SMC_SIZET)
    * num_views );
SMC_DATAITEM_STRUCT** view_list = (SMC_DATAITEM_STRUCT**)
    malloc (sizeof(SMC_DATAITEM_STRUCT*) * num_views );
SMC_SIZET** view_id_handle_list = (SMC_SIZET**) malloc
    (sizeof(SMC_SIZET*) * num_views );
SMC_SIZET* view_id_list = (SMC_SIZET*) malloc
    (sizeof(SMC_SIZET) * num_views );

SMC_SIZET client_id;
SMC_SIZETP client_id_handle = &client_id;

SMC_SERVER_MODE server_mode = SMC_SERVER_M_LIVE;
SMC_CHAR server_name[ 40 ];
SMC_CHAR user_name[ 40 ];
SMC_CHAR password[ 40 ];
SMC_CHAR interfaces_file[ 40 ];

SMC_RETURN_CODE ret;
SMC_SIZET refresh_num, view_num, col_num, row_num;

SMC_SIZET num_refreshes = 10;

SMC_SIZET row_count;
SMC_SIZETP row_count_handle = &row_count;

SMC_DATAITEM_STRUCTP dataitem_list;
SMC_DATAITEM_NAME dataitem_name;
```

```

SMC_CHARP dataitem_name_str;
SMC_DATAITEM_STATTYPE dataitem_stat;
SMC_CHARP dataitem_stat_str;
SMC_DATAITEM_TYPE dataitem_type;

SMC_VALUE_UNION data_union;
SMC_VALUE_UNIONNP data_union_handle = &data_union;
SMC_CHARP data_str;
SMC_INT ival;
printf("*****\n");
printf("*** Test Driver for SQL Monitor Client Library **\n");
printf("*****\n");
if (argc != 5)
{
    printf(Usage: testcli <SQLMonitorServer> <user> <password>
        <"interfaces_file">\n");
    exit(1);
}

strcpy(server_name, argv[1]);
strcpy(user_name, argv[2]);
strcpy(password, argv[3]);
strcpy(interfaces_file, argv[4]);

for(view_num=0; view_num<num_views; view_num++)
{
    view_id_handle_list[ view_num ] = &(view_id_list[ view_num ]);
}

view_count [ 0 ] = cache_perf_sum_count;
view_list  [ 0 ] = cache_perf_sum_view;
view_count [ 1 ] = object_lock_status_count;
view_list  [ 1 ] = object_lock_status_view;
view_count [ 2 ] = object_page_io_count;
view_list  [ 2 ] = object_page_io_view;
view_count [ 3 ] = session_page_cache_stats_count;
view_list  [ 3 ] = session_page_cache_stats_view;
view_count [ 4 ] = sample_page_cache_stats_count;
view_list  [ 4 ] = sample_page_cache_stats_view;
view_count [ 5 ] = session_device_io_count;
view_list  [ 5 ] = session_device_io_view;
view_count [ 6 ] = sample_device_io_count;
view_list  [ 6 ] = sample_device_io_view;
view_count [ 7 ] = device_perf_sum_count;
view_list  [ 7 ] = device_perf_sum_view;
view_count [ 8 ] = engine_activity_count;

```

```
view_list [ 8 ] = engine_activity_view;
view_count [ 9 ] = lock_perf_sum_count;
view_list [ 9 ] = lock_perf_sum_view;
view_count [ 10 ] = session_network_activity_count;
view_list [ 10 ] = session_network_activity_view;
view_count [ 11 ] = sample_network_activity_count;
view_list [ 11 ] = sample_network_activity_view;
view_count [ 12 ] = network_perf_sum_count;
view_list [ 12 ] = network_perf_sum_view;
view_count [ 13 ] = session_procedure_cache_stats_count;
view_list [ 13 ] = session_procedure_cache_stats_view;
view_count [ 14 ] = sample_procedure_cache_stats_count;
view_list [ 14 ] = sample_procedure_cache_stats_view;
view_count [ 15 ] = procedure_page_cache_io_count;
view_list [ 15 ] = procedure_page_cache_io_view;
view_count [ 16 ] = process_activity_count;
view_list [ 16 ] = process_activity_view;
view_count [ 17 ] = process_object_page_io_count;
view_list [ 17 ] = process_object_page_io_view;
view_count [ 18 ] = process_detail_locks_count;
view_list [ 18 ] = process_detail_locks_view;
view_count [ 19 ] = process_detail_io_count;
view_list [ 19 ] = process_detail_io_view;
view_count [ 20 ] = process_lock_count;
view_list [ 20 ] = process_lock_view;
view_count [ 21 ] = process_page_io_count;
view_list [ 21 ] = process_page_io_view;
view_count [ 22 ] = process_perf_sum_count;
view_list [ 22 ] = process_perf_sum_view;
view_count [ 23 ] = process_procedure_page_io_count;
view_list [ 23 ] = process_procedure_page_io_view;
view_count [ 24 ] = server_perf_sum_count;
view_list [ 24 ] = server_perf_sum_view;
view_count [ 25 ] = procedure_activity_count;
view_list [ 25 ] = procedure_activity_view;
view_count [ 26 ] = transaction_activity_count;
view_list [ 26 ] = transaction_activity_view;

printf("***** testing smc_connect() *****\n");
ret = smc_connect(server_mode,
                  server_name,
                  user_name,
                  password,
                  interfaces_file,
                  ErrorCallback,
                  0,
```

```

        0,
        client_id_handle);
if ( ret != SMC_RET_SUCCESS )
{
    printf("error returned by smc_connect()\n");
    return (int) ret;
}
else
{
    printf("smc_connect() succeeded\n");
}
printf("***** testing smc_create_view() *****\n");
for(view_num=0; view_num<num_views; view_num++)
{
    ret = smc_create_view(client_id,
        view_list[ view_num ],
        view_count[ view_num ],
        (SMC_CHARP) 0,
        view_id_handle_list[ view_num ]);
    if ( ret != SMC_RET_SUCCESS )
    {
        printf("error returned by smc_create_view( %d )\n",
            view_num);
        return (int) ret;
    }
    else
    {
        printf("smc_create_view( %d ) succeeded\n", view_num);
    }
}
printf("***** testing smc_refresh() *****\n");
for(refresh_num=0; refresh_num<num_refreshes; refresh_num++)
{
    ret = smc_refresh(client_id,
        (SMC_VOIDP) 0,
        RefreshCallback,
        0);
    if ( ret != SMC_RET_SUCCESS )
    {
        printf("error returned by smc_refresh() number %d\n",
            refresh_num);
        return (int) ret;
    }
    else
    {
        printf("smc_refresh() number %d succeeded\n", refresh_num);
    }
}

```

```
}

for(view_num=0; view_num<num_views; view_num++)
{
    printf("*****   testing smc_get_row_count()   *****\n");
    ret = smc_get_row_count(client_id,
                           view_id_list[ view_num ],
                           row_count_handle);
    if ( ret != SMC_RET_SUCCESS )
    {
        printf("error returned by smc_get_row_count()\n");
        return (int) ret;
    }
    else
    {
        printf("smc_get_row_count( view_id = %d ) = %d\n",
              view_id_list[view_num], row_count);
    }

    dataitem_list = view_list[view_num];

    /* print dataitem name headers */
    for(col_num = 0; col_num<view_count[ view_num ]; col_num++)
    {
        dataitem_name = (dataitem_list[col_num]).dataItemName;
        dataitem_name_str = LookupDataItemName( dataitem_name );
        printf("Col %d %s\t", col_num, dataitem_name_str);
    }
    printf("\n");

    /* print dataitem stattype headers */
    for(col_num = 0; col_num<view_count[ view_num ]; col_num++)
    {
        dataitem_stat = (dataitem_list[col_num]).dataItemStatType;
        dataitem_stat_str = LookupDataItemStat( dataitem_stat );
        printf("Col %d %s\t", col_num, dataitem_stat_str);
    }
    printf("\n");

    for(row_num = 0; row_num<row_count; row_num++)
    {
        for(col_num = 0; col_num<view_count[ view_num ];
            col_num++)
        {
            dataitem_name = (dataitem_list[col_num]).dataItemName;
            dataitem_stat = (dataitem_list[col_num]).dataItemStatType;
```

```

dataitem_name_str = LookupDataItemName( dataitem_name );
ret = smc_get_dataitem_value(client_id,
                             view_id_list[ view_num ],
                             &(dataitem_list[col_num]),
                             row_num,
                             data_union_handle);

if ( ret != SMC_RET_SUCCESS )
{
    printf("error returned by smc_get_dataitem_value()\n");
    return (int) ret;
}

smc_get_dataitem_type(&(dataitem_list[col_num]),
                    &dataitem_type);

switch(dataitem_type)
{
    case SMC_DI_TYPE_CHARP:
        printf("Col %d:
               \"%s\"\\t", col_num, data_union.stringValue);
        free( data_union.stringValue );
        break;
    case SMC_DI_TYPE_DOUBLE:
        printf("Col %d:
               %f\\t", col_num, data_union.doubleValue);
        break;
    case SMC_DI_TYPE_ENUMS:
        ival = data_union.intValue;
        switch (dataitem_name)
        {
            case SMC_NAME_LOCK_RESULT_SUMMARY:
                data_str = LookupLockResultSummary(
                    ((SMC_LOCK_RESULT_SUMMARY) ival) );
                printf("Col %d: \"%s\"\\t", col_num, data_str );
                break;
            case SMC_NAME_LOCK_RESULT:
                data_str = LookupLockResult(
                    ((SMC_LOCK_RESULT) ival) );
                printf("Col %d: \"%s\"\\t", col_num, data_str );
                break;
            case SMC_NAME_LOCK_STATUS:
                data_str = LookupLockStatus(
                    ((SMC_LOCK_STATUS) ival) );
                printf("Col %d: \"%s\"\\t", col_num, data_str );
                break;
            case SMC_NAME_LOCK_TYPE:

```

```
        data_str = LookupLockType( ((SMC_LOCK_TYPE)
            ival) );
        printf("Col %d: \"%s\"\\t", col_num, data_str );
        break;
    case SMC_NAME_OBJ_TYPE:
        data_str = LookupObjectType( ((SMC_OBJ_TYPE)
            ival) );
        printf("Col %d: \"%s\"\\t", col_num, data_str );
        break;
    case SMC_NAME_CUR_PROC_STATE:
    case SMC_NAME_PROC_STATE:
        data_str = LookupProcessState(
            ((SMC_PROCESS_STATE) ival) );
        printf("Col %d: \"%s\"\\t", col_num, data_str );
        break;
    default:
        printf("Col %d: \"ERR with %s\"\\t", col_num,
            dataitem_name_str );
    }
    break;
case SMC_DI_TYPE_LONG:
    printf("Col %d: %d\\t", col_num,
        data_union.longValue);
    break;
case SMC_DI_TYPE_DATIM:
case SMC_DI_TYPE_NONE:
default:
    printf("Col %d: \"ERR with %s\"\\t", col_num,
        dataitem_name_str );
}
}
printf("\\n");
}
}

printf("***** testing smc_disconnect() *****\\n");
ret = smc_disconnect(client_id);
if ( ret != SMC_RET_SUCCESS )
{
    printf("error returned by smc_disconnect()\\n");
    return (int) ret;
}
{
    printf("smc_disconnect() succeeded\\n");
}
```

```
    free(view_count);
    free(view_list);

    return 0;
}

SMC_VOID
ErrorCallback(
    SMC_SIZET    id,
    SMC_SIZET    error_number,
    SMC_SIZET    severity,
    SMC_SIZET    map_severity,
    SMC_SIZET    source,
    SMC_CCHARP   error_msg,
    SMC_SIZET    state
)
{
    printf("*****\n");
    printf("Inside ErrorCallback()\n");

    printf("id = %d\n", id);
    printf("error_number = %d\n", error_number);
    printf("err severity = %d\n", severity);
    printf("map severity = %d\n", map_severity);
    printf("source = %d\n", source);
    printf("error msg = %s\n", error_msg);
    printf("state = %d\n", state);
    printf("*****\n");
    return;
}

SMC_VOID
RefreshCallback(
    SMC_SIZET    id,
    SMC_VOIDP    user_msg,
    SMC_CHARP    msg
)
{
    printf("*****\n");
    printf("Inside RefreshCallback()\n");

    printf("id = %d\n", id);
    printf("user_msg = %s\n", (SMC_CHARP) user_msg);
```

```
printf("msg = %s\n", msg);

return;
}
```

SMC_CHARP

LookupDataItemName(

```
    SMC_DATAITEM_NAME    value
)
```

```
{
    typedef struct {
        SMC_CHARP      str_name;
        SMC_DATAITEM_NAME  enum_name;
    } DATAITEM_NAME_MAPPER;
    DATAITEM_NAME_MAPPER    dataitem_name_map[] = {
    { "Process ID",          SMC_NAME_SPID },
    { "Kernel Process ID",   SMC_NAME_KPID },
    { "Cache Name",          SMC_NAME_DATA_CACHE_NAME },
    { "Database ID",         SMC_NAME_DB_ID },
    { "Object ID",           SMC_NAME_OBJ_ID },
    { "Procedure Database ID", SMC_NAME_ACT_STP_DB_ID },
    { "Procedure ID",        SMC_NAME_ACT_STP_ID },
    { "Procedure Line Number", SMC_NAME_STP_LINE_NUM },
    { "Lock Type",           SMC_NAME_LOCK_TYPE },
    { "Lock Result",         SMC_NAME_LOCK_RESULT },
    { "Lock Results Summarized", SMC_NAME_LOCK_RESULT_SUMMARY },
    { "Lock Status",         SMC_NAME_LOCK_STATUS },
    { "Engine Number",       SMC_NAME_ENGINE_NUM },
    { "Page Number",         SMC_NAME_PAGE_NUM },
    { "Device Name",         SMC_NAME_DEV_NAME },
    { "Process State",       SMC_NAME_PROC_STATE },
    { "Login Name",          SMC_NAME_LOGIN_NAME },
    { "Database Name",       SMC_NAME_DB_NAME },
    { "Owner Name",          SMC_NAME_OWNER_NAME },
    { "Object Name",         SMC_NAME_OBJ_NAME },
    { "Object Type",         SMC_NAME_OBJ_TYPE },
    { "Procedure Database Name", SMC_NAME_ACT_STP_DB_NAME },
    { "Procedure Owner Name", SMC_NAME_ACT_STP_OWNER_NAME },
    { "Procedure Name",       SMC_NAME_ACT_STP_NAME },
    { "Blocking Process ID",  SMC_NAME_BLOCKING_SPID },
    { "Cache Efficiency",     SMC_NAME_DATA_CACHE_EFFICIENCY },
    { "Cache Hit Pct",        SMC_NAME_DATA_CACHE_HIT_PCT },
    { "Cache Hits",           SMC_NAME_DATA_CACHE_HIT },
    { "Cache Misses",         SMC_NAME_DATA_CACHE_MISS },
    { "Cache Spinlock Contention", SMC_NAME_DATA_CACHE_CONTENTION },
    }
```

```

{ "Connect Time",          SMC_NAME_CONNECT_TIME },
{ "CPU Busy Percent",      SMC_NAME_CPU_BUSY_PCT },
{ "CPU Percent",          SMC_NAME_CPU_PCT },
{ "CPU Time",              SMC_NAME_CPU_TIME },
{ "Current Engine",        SMC_NAME_CUR_ENGINE },
{ "Current Process State", SMC_NAME_CUR_PROC_STATE },
{ "Deadlock Count",        SMC_NAME_DEADLOCK_CNT },
{ "Demand Lock",           SMC_NAME_DEMAND_LOCK },
{ "Device Hits",           SMC_NAME_DEV_HIT },
{ "Device Hit Percent",    SMC_NAME_DEV_HIT_PCT },
{ "Device I/O",            SMC_NAME_DEV_IO },
{ "Device Misses",         SMC_NAME_DEV_MISS },
{ "Device Reads",          SMC_NAME_DEV_READ },
{ "Device Writes",         SMC_NAME_DEV_WRITE },
{ "Lock Count",            SMC_NAME_LOCK_CNT },
{ "Lock Hit Percent",      SMC_NAME_LOCK_HIT_PCT },
{ "Lock Status Count",     SMC_NAME_LOCK_STATUS_CNT },
{ "Locks Being Blocked Count", SMC_NAME_LOCKS_BEING_BLOCKED_CNT },
{ "Code Memory Size",      SMC_NAME_MEM_CODE_SIZE },
{ "Kernel Structures Memory Size", SMC_NAME_MEM_KERNEL_STRUCT_SIZE },
{ "Page Cache Size",       SMC_NAME_MEM_PAGE_CACHE_SIZE },
{ "Procedure Buffer Size",  SMC_NAME_MEM_PROC_BUFFER },
{ "Procedure Header Size", SMC_NAME_MEM_PROC_HEADER },
{ "Server Structures Size", SMC_NAME_MEM_SERVER_STRUCT_SIZE },
{ "Most Active Device I/O", SMC_NAME_MOST_ACT_DEV_IO },
{ "Most Active Device Name", SMC_NAME_MOST_ACT_DEV_NAME },
{ "Net I/O Bytes",         SMC_NAME_NET_BYTE_IO },
{ "Net Bytes Received",    SMC_NAME_NET_BYTES_RCVD },
{ "Net Bytes Sent",        SMC_NAME_NET_BYTES_SENT },
{ "Net Default Packet Size", SMC_NAME_NET_DEFAULT_PKT_SIZE },
{ "Net Max Packet Size",   SMC_NAME_NET_MAX_PKT_SIZE },
{ "Net Packet Size Received", SMC_NAME_NET_PKT_SIZE_RCVD },
{ "Net Packet Size Sent",  SMC_NAME_NET_PKT_SIZE_SENT },
{ "Net Packets Received",  SMC_NAME_NET_PKTS_RCVD },
{ "Net Packets Sent",      SMC_NAME_NET_PKTS_SENT },
{ "Page Hit Percent",      SMC_NAME_PAGE_HIT_PCT },
{ "Logical Page Reads",    SMC_NAME_PAGE_LOGICAL_READ },
{ "Page I/O",              SMC_NAME_PAGE_IO },
{ "Physical Page Reads",   SMC_NAME_PAGE_PHYSICAL_READ },
{ "Page Writes",           SMC_NAME_PAGE_WRITE },
{ "Process State Count",   SMC_NAME_PROC_STATE_CNT },
{ "Timestamp",             SMC_NAME_TIMESTAMP },
{ "Elapsed Time",          SMC_NAME_ELAPSED_TIME },
{ "SQL Server Name",       SMC_NAME_SQL_SERVER_NAME },
{ "SQL Server Version",    SMC_NAME_SQL_SERVER_VERSION },
{ "Procedure Elapsed Time", SMC_NAME_STP_ELAPSED_TIME },

```

```
{ "Procedure Hit Percent", SMC_NAME_STP_HIT_PCT },
{ "Procedure Line Text", SMC_NAME_STP_LINE_TEXT },
{ "Procedure Execution Count", SMC_NAME_STP_NUM_TIMES_EXECUTED },
{ "Procedure Logical Reads", SMC_NAME_STP_LOGICAL_READ },
{ "Procedure Physical Reads", SMC_NAME_STP_PHYSICAL_READ },
{ "Time Waited on Lock", SMC_NAME_TIME_WAITED_ON_LOCK },
{ "Transactions", SMC_NAME_XACT },
{ "Rows Deleted", SMC_NAME_XACT_DELETE },
{ "Rows Inserted Clustered", SMC_NAME_XACT_CINSERT },
{ "Rows Inserted", SMC_NAME_XACT_INSERT },
{ "Rows Inserted Nonclustered", SMC_NAME_XACT_NCINSERT },
{ "Rows Updated", SMC_NAME_XACT_UPDATE },
{ "Rows Updated Directly", SMC_NAME_XACT_UPDATE_DIRECT },
{ (SMC_CHARP)0, SMC_NAME_NONE }
};

SMC_INT    idx = 0;
SMC_BOOL   match = FALSE;
while( match == FALSE)
{
    if ( value == dataitem_name_map[ idx ].enum_name )
        return dataitem_name_map[ idx ].str_name;

    if (dataitem_name_map[ idx ].enum_name == SMC_NAME_NONE )
        return dataitem_name_map[ idx ].str_name;

    idx++;
}

SMC_CHARP
LookupDataItemStat(
    SMC_DATAITEM_STATTYPE    value
)
{
    typedef struct {
        SMC_CHARP            str_stat;
        SMC_DATAITEM_STATTYPE enum_stat;
    } DATAITEM_STAT_MAPPER;

    DATAITEM_STAT_MAPPER    dataitem_stat_map[] = {
    { "Value for Sample",      SMC_STAT_VALUE_SAMPLE },
    { "Value for Session",     SMC_STAT_VALUE_SESSION },
    { "Rate for Sample",       SMC_STAT_RATE_SAMPLE },
    { "Rate for Session",      SMC_STAT_RATE_SESSION },
```

```

{ "Avg for Sample",          SMC_STAT_AVG_SAMPLE },
{ "Avg for Session",        SMC_STAT_AVG_SESSION },
{ (SMC_CHARP)0,0 }
};

SMC_INT    idx = 0;
SMC_BOOL   match = FALSE;

while( match == FALSE)
{
    if ( value == dataitem_stat_map[ idx ].enum_stat )
        return dataitem_stat_map[ idx ].str_stat;

    if (dataitem_stat_map[ idx ].enum_stat == 0 )
        return dataitem_stat_map[ idx ].str_stat;

    idx++;
}
}

SMC_CHARP
LookupLockResult(
    SMC_LOCK_RESULT    value
)
{
    typedef struct {
        SMC_CHARP        str_lock_res;
        SMC_LOCK_RESULT  enum_lock_res;
    } LOCK_RESULT_MAPPER;

    LOCK_RESULT_MAPPER lock_result_map[] = {
    { "granted",          SMC_LOCK_R_GRANTED },
    { "notneeded",        SMC_LOCK_R_NOTNEEDED },
    { "waited",           SMC_LOCK_R_WAITED },
    { "didntwait",        SMC_LOCK_R_DIDNTWAIT },
    { "deadlock",         SMC_LOCK_R_DEADLOCK },
    { "interrupted",      SMC_LOCK_R_INTERRUPTED},
    { (SMC_CHARP)0,0 }
    };

    SMC_INT    idx = 0;
    SMC_BOOL   match = FALSE;

    while( match == FALSE)
    {

```

```
    if ( value == lock_result_map[ idx ].enum_lock_res )
        return lock_result_map[ idx ].str_lock_res;

    if (lock_result_map[ idx ].enum_lock_res == 0 )
        return lock_result_map[ idx ].str_lock_res;

    idx++;
}
}
SMC_CHARP
LookupLockResultSummary(
    SMC_LOCK_RESULT_SUMMARY    value
)
{
    typedef struct {
        SMC_CHARP                str_lock_ressum;
        SMC_LOCK_RESULT_SUMMARY  enum_lock_ressum;
    } LOCK_RESULT_SUMMARY_MAPPER;

    LOCK_RESULT_SUMMARY_MAPPER lock_result_summary_map[] = {
    { "granted",                SMC_LOCK_RS_GRANTED },
    { "notgranted",            SMC_LOCK_RS_NOTGRANTED },
    { (SMC_CHARP)0,0 }
    };

    SMC_INT    idx = 0;
    SMC_BOOL   match = FALSE;

    while( match == FALSE)
    {
        if ( value == lock_result_summary_map[ idx ].enum_lock_ressum )
            return lock_result_summary_map[ idx ].str_lock_ressum;

        if (lock_result_summary_map[ idx ].enum_lock_ressum == 0 )
            return lock_result_summary_map[ idx ].str_lock_ressum;

        idx++;
    }
}

SMC_CHARP
LookupLockStatus(
    SMC_LOCK_STATUS    value
)
{
    typedef struct {
```

```

    SMC_CHARP      str_lock_status;
    SMC_LOCK_STATUS enum_lock_status;
} LOCK_STATUS_MAPPER;

LOCK_STATUS_MAPPER lock_status_map[] = {
{ "held_blocking",      SMC_LOCK_S_HELD_BLOCKING },
{ "held_notblocking",   SMC_LOCK_S_HELD_NOTBLOCKING },
{ "requested_blocked",  SMC_LOCK_S_REQUESTED_BLOCKED },
{ "requested_notblocked", SMC_LOCK_S_REQUESTED_NOTBLOCKED },
{ (SMC_CHARP)0, 0 }
};

SMC_INT    idx = 0;
SMC_BOOL   match = FALSE;

while( match == FALSE)
{
    if ( value == lock_status_map[ idx ].enum_lock_status )
        return lock_status_map[ idx ].str_lock_status;

    if (lock_status_map[ idx ].enum_lock_status == 0 )
        return lock_status_map[ idx ].str_lock_status;

    idx++;
}
}

SMC_CHARP
LookupLockType(
    SMC_LOCK_TYPE    value
)
{
    typedef struct {
        SMC_CHARP      str_lock_type;
        SMC_LOCK_TYPE  enum_lock_type;
    } LOCK_TYPE_MAPPER;

    LOCK_TYPE_MAPPER lock_type_map[] = {
    { "ex_tab",          SMC_LOCK_T_EX_TAB },
    { "sh_tab",          SMC_LOCK_T_SH_TAB },
    { "ex_int",          SMC_LOCK_T_EX_INT },
    { "sh_int",          SMC_LOCK_T_SH_INT },
    { "ex_page",         SMC_LOCK_T_EX_PAGE },
    { "sh_page",         SMC_LOCK_T_SH_PAGE },
    { "upd_page",        SMC_LOCK_T_UP_PAGE },

```

```
{ (SMC_CHARP) 0, 0 }
};

SMC_INT    idx = 0;
SMC_BOOL   match = FALSE;

while( match == FALSE)
{
    if ( value == lock_type_map[ idx ].enum_lock_type )
        return lock_type_map[ idx ].str_lock_type;

    if (lock_type_map[ idx ].enum_lock_type == 0 )
        return lock_type_map[ idx ].str_lock_type;

    idx++;
}
```

Summary of datatypes

Table B-1 lists Monitor Client Library type constants with descriptions and their corresponding C or Open Client datatypes.

Table B-1: Summary of datatypes

| Monitor Client Library datatype | Description | Corresponding C or Open Client datatype |
|---------------------------------|--|---|
| SMC_ALARM_ACTION_TYPE | Specifies the type of action to take when an alarm is triggered | None |
| SMC_ALARM_ID | Alarm identifier | size_t |
| SMC_ALARM_IDP | Pointer to alarm identifier | size_t* |
| SMC_BOOL | Boolean | int |
| SMC_CHAR | Character | char |
| SMC_CHARP | Character pointer | char* |
| SMC_CHARPP | Pointer to character pointer | char** |
| SMC_CCHARP | Constant character pointer | CS_CONST char* |
| SMC_CLOSE_TYPE | Specifies an option when closing a Adaptive Server Enterprise Monitor connection | None |
| SMC_COMMAND_ID | Command identifier | size_t |
| SMC_COMMAND_IDP | Pointer to command identifier | size_t* |
| SMC_CONNECT_ID | Connection identifier | size_t |
| SMC_CONNECT_IDP | Pointer to connection identifier | size_t* |
| SMC_DATETIME | Date and time | CS_DATETIME |
| SMC_DATAITEM_NAME | Identifies a particular piece of performance data that Monitor Client Library is to obtain | None |
| SMC_DATAITEM_NAMEP | Pointer to SMC_DATAITEM_NAME | None |
| SMC_DATAITEM_STATTYPE | Identifies what normalization, if any, Monitor Client Library should perform on data | None |
| SMC_DATAITEM_STRUCT | Identifies data that Monitor Client Library is to obtain | None |
| SMC_DATAITEM_STRUCTP | Pointer to SMC_DATAITEM_STRUCT | None |

| Monitor Client Library datatype | Description | Corresponding C or Open Client datatype |
|---------------------------------|--|---|
| SMC_DATAITEM_TYPE | Identifies datatype of data that Monitor Client Library obtains | None |
| SMC_DATAITEM_TYPEP | Pointer to SMC_DATAITEM_TYPE | None |
| SMC_DOUBLE | Double precision floating point | double |
| SMC_DOUBLEP | Pointer to double precision | double* |
| SMC_ERR_SEVERITY | Indicates the degree of severity of an error | None |
| SMC_FILTER_ID | Filter identifier | size_t |
| SMC_FILTER_IDP | Pointer to filter identifier | size_t* |
| SMC_FILTER_TYPE | Specifies the type of filter to create with <code>smc_create_filter</code> | None |
| SMC_HS_ESTIM_OPT | Specifies whether, in playback of historical performance data, to authorize estimation of data that cannot be calculated reliably from the available recorded data | None |
| SMC_HS_MISSDATA_OPT | Specifies whether, in playback of historical performance data, a sample should be returned for a period of time for which no data is available | None |
| SMC_HS_PLAYBACK_OPT | Specifies whether playback of historical performance data should be normalized or summarized or both | None |
| SMC_HS_SESS_DELETE_OPT | Specifies whether to delete data files associated with a Historical Server session | None |
| SMC_HS_SESS_ERR_OPT | Specifies whether a recording session should continue after an error | None |
| SMC_HS_SESS_PROT_LEVEL | Specifies whether the data in a recording session should be accessible to other users | None |
| SMC_HS_SESS_SCRIPT_OPT | Specifies whether to create a script to create tables corresponding to the views in a recording session | None |
| SMC_HS_TARGET_OPT | Specifies whether playback of historical performance data should be sent to the client application, or used to create a new session | None |
| SMC_INFO_TYPE | Specifies the type of information to request in a call to <code>smc_get_command_info</code> | None |
| SMC_INT | Integer | int |
| SMC_INTP | Pointer to integer | int* |
| SMC_LOCK_RESULT | Identifies the possible outcomes of a lock request | None |
| SMC_LOCK_RESULT_SUMMARY | Identifies the two major categories of outcomes of a lock request | None |

| Monitor Client Library datatype | Description | Corresponding C or Open Client datatype |
|--|---|--|
| SMC_LOCK_STATUS | Identifies the possible statuses of a lock or lock request | None |
| SMC_LOCK_TYPE | Identifies the granularity and exclusivity of a lock | None |
| SMC_LONG | Long | long |
| SMC_LONGP | Pointer to long | long* |
| SMC_OBJ_TYPE | Identifies the type of an object in an Adaptive Server database | None |
| SMC_PROC_STATE | Identifies the possible statuses of an Adaptive Server process | None |
| SMC_PROP_ACTION | Specifies the action to take in a call to <code>smc_connect_props</code> | None |
| SMC_PROP_TYPE | Specifies the property that is the object of a call to <code>smc_connect_props</code> | None |
| SMC_RETURN_CODE | Indicates whether a Monitor Client Library operation succeeded, and, if not, what error occurred | None |
| SMC_SERVER_MODE | Specifies whether a Adaptive Server Enterprise Monitor connection is to obtain live performance data or whether to manipulate historical data | None |
| SMC_SESSION_ID | Session identifier | size_t |
| SMC_SESSION_IDP | Pointer to session identifier | size_t* |
| SMC_SIZET | unsigned integer | size_t |
| SMC_SIZETP | Pointer to unsigned integer | size_t* |
| SMC_SOURCE | Indicates the software layer that detected an error | None |
| SMC_VALUE_UNION | Structure containing data | None |
| SMC_VALUE_UNIONP | Pointer to SMC_VALUE_UNION | None |
| SMC_VIEW_ID | View identifier | size_t |
| SMC_VIEW_IDP | Pointer to view identifier | size_t* |
| SMC_VOID | Void | void |
| SMC_VOIDP | Pointer to void | void* |

The rest of this appendix describes individual datatypes that have no equivalent in C or Open-Client Client Library.

Enum: SMC_ALARM_ACTION_TYPE

An enum to identify the type of action taken when an alarm is triggered:

Table B-2: Alarm action type

| |
|-------------------------|
| SMC_ALARM_A_EXEC_PROC |
| SMC_ALARM_A_LOG_TO_FILE |
| SMC_ALARM_A_NOTIFY |

Enum: SMC_CLOSE_TYPE

An enum used to identify the extent of a close command:

Table B-3: Close type

| |
|-------------------|
| SMC_CLOSE_REQUEST |
|-------------------|

Enum: SMC_DATAITEM_NAME

An enum used in conjunction with `smc_create_view` to specify performance data. See Chapter 2, “Data Items and Statistical Types” for a list of the available data items.

Enum: SMC_DATAITEM_STATTYPE

An enum used in conjunction with `smc_create_view` to identify statistical type and accumulation interval of performance data.

Table B-4: Data item statistical type

| |
|------------------------|
| SMC_STAT_VALUE_SAMPLE |
| SMC_STAT_VALUE_SESSION |
| SMC_STAT_RATE_SAMPLE |
| SMC_STAT_RATE_SESSION |
| SMC_STAT_AVG_SAMPLE |
| SMC_STAT_AVG_SESSION |

Structure: SMC_DATAITEM_STRUCT

A structure used in conjunction with `smc_create_view` to identify performance data.

| typedef struct SMC_DATAITEM_STRUCT{ | |
|--|-------------------------|
| SMC_DATAITEM_NAME | <i>dataItemName</i> |
| SMC_DATAITEM_STATTYPE | <i>dataItemStatType</i> |
| } SMC_DATAITEM_STRUCT; | |

Enum: SMC_DATAITEM_TYPE

An enum used in conjunction with `smc_get_dataitem_type` to identify physical type of performance data results:

Table B-5: Data item type

| |
|--------------------|
| SMC_DI_TYPE_NONE |
| SMC_DI_TYPE_CHARP |
| SMC_DI_TYPE_DATIM |
| SMC_DI_TYPE_DOUBLE |
| SMC_DI_TYPE_ENUMS |
| SMC_DI_TYPE_INT |
| SMC_DI_TYPE_LONG |

Enum: SMC_ERR_SEVERITY

An enum used in conjunction with `smc_get_command_info` to identify the severity of an error, warning, or informational notification.

Table B-6: Error severity

| |
|-------------------|
| SMC_ERR_SEV_INFO |
| SMC_ERR_SEV_WARN |
| SMC_ERR_SEV_FATAL |

Enum: SMC_FILTER_TYPE

An enum to identify the types of filters:

Table B-7: Filter type

| |
|------------------------|
| SMC_FILTER_T_EQ |
| SMC_FILTER_T_NEQ |
| SMC_FILTER_T_GE |
| SMC_FILTER_T_LE |
| SMC_FILTER_T_GE_AND_LE |
| SMC_FILTER_T_TOP_N |

Enum: SMC_HS_ESTIM_OPT

An enum to specify whether to allow certain data to be estimated during a playback session.

Table B-8: Historical Server error action

| |
|-----------------------|
| SMC_HS_ESTIM_ALLOW |
| SMC_HS_ESTIM_DISALLOW |

Enum: SMC_HS_MISSDATA_OPT

An enum to specify what action Historical Server should take if a given sample during a playback session has no performance data to play back:

Table B-9: Historical Server missing data option

| |
|----------------------|
| SMC_HS_MISSDATA_SHOW |
| SMC_HS_MISSDATA_SKIP |

Enum: SMC_HS_PLAYBACK_OPT

An enum to specify whether data for a playback session should be normalized, summarized, or both.

Table B-10: Historical Server protection level

| |
|-------------------------|
| SMC_HS_PBTYPED_ENTIRE |
| SMC_HS_PBTYPED_ACTUAL |
| SMC_HS_PBTYPED_INTERVAL |
| SMC_HS_PBTYPED_RAW |

Enum: SMC_HS_SESS_DELETE_OPT

An enum to specify whether to delete data files associated with a Historical Server connection.

Table B-11: Historical Server file deletion option

| |
|--------------------------|
| SMC_HS_SESS_DELETE_FILES |
| SMC_HS_SESS_RETAIN_FILES |

Enum: SMC_HS_SESS_ERR_OPT

An enum to specify what action Historical Server should take if a recording session encounters non-fatal errors:

Table B-12: Historical Server error option

| |
|----------------------|
| SMC_HS_SESS_ERR_CONT |
| SMC_HS_SESS_ERR_HALT |

Enum: SMC_HS_SESS_PROT_LEVEL

An enum to specify the protection level for access to performance data recorded by Historical Server:

Table B-13: Historical Server protection level

| |
|--------------------------|
| SMC_HS_SESS_PROT_PRIVATE |
| SMC_HS_SESS_PROT_PUBLIC |

Enum: SMC_HS_SESS_SCRIPT_OPT

An enum to specify the type of script (if any) that Historical Server should create to help the user to manipulate the performance data of a recording session:

Table B-14: Historical Server script option

| |
|---------------------------|
| SMC_HS_SESS_SCRIPT_SYBASE |
| SMC_HS_SESS_SCRIPT_NONE |

Enum: SMC_HS_TARGET_OPT

An enum to specify whether the playback session will return data to the application or whether playback will create a new session on Historical Server:

Table B-15: Historical Server script option

| |
|----------------------|
| SMC_HS_TARGET_CLIENT |
| SMC_HS_TARGET_FILE |

Enum: SMC_HS_TARGET_OPT

An enum to specify the destination of data in a playback session:

Table B-16: Historical Server playback target option

| |
|----------------------|
| SMC_HS_TARGET_CLIENT |
| SMC_HS_TARGET_FILE |

Enum: SMC_INFO_TYPE

An enum to identify the various pieces of data that are available for querying from a callback function, using `smc_get_command_info`:

Table B-17: Information type

| |
|----------------------------|
| SMC_INFO_ALARM_ACTION_DATA |
| SMC_INFO_ALARM_ALARMID |

| |
|--------------------------------|
| SMC_INFO_ALARM_CURRENT_VALUE |
| SMC_INFO_ALARM_DATAITEM |
| SMC_INFO_ALARM_ROW |
| SMC_INFO_ALARM_THRESHOLD_VALUE |
| SMC_INFO_ALARM_TIMESTAMP |
| SMC_INFO_ALARM_VALUE_DATATYPE |
| SMC_INFO_ALARM_VIEWID |
| SMC_INFO_ERR_MAPSEVERITY |
| SMC_INFO_ERR_MSG |
| SMC_INFO_ERR_NUM |
| SMC_INFO_ERR_SEVERITY |
| SMC_INFO_ERR_SOURCE |
| SMC_INFO_ERR_STATE |

Enum: SMC_LOCK_RESULT

An enum to identify results of a lock request:

Table B-18: Lock result type

| |
|------------------------|
| SMC_LOCK_R_GRANTED |
| SMC_LOCK_R_NOTNEEDED |
| SMC_LOCK_R_WAITED |
| SMC_LOCK_R_DIDNTWAIT |
| SMC_LOCK_R_DEADLOCK |
| SMC_LOCK_R_INTERRUPTED |

Enum: SMC_LOCK_RESULT_SUMMARY

An enum to identify whether the lock request was granted or not granted:

Table B-19: Lock result summary type

| |
|------------------------|
| SMC_LOCK_RS_GRANTED |
| SMC_LOCK_RS_NOTGRANTED |

Enum: SMC_LOCK_STATUS

An enum to identify the status of a lock:

| Table B-20: Lock status type | |
|---------------------------------|--|
| SMC_LOCK_S_HELD_BLOCKING | |
| SMC_LOCK_S_HELD_NOTBLOCKING | |
| SMC_LOCK_S_REQUESTED_BLOCKED | |
| SMC_LOCK_S_REQUESTED_NOTBLOCKED | |

Enum: SMC_LOCK_TYPE

An enum to identify lock types:

| Table B-21: Lock type | |
|-----------------------|--|
| SMC_LOCK_T_EX_TAB | |
| SMC_LOCK_T_SH_TAB | |
| SMC_LOCK_T_EX_INT | |
| SMC_LOCK_T_SH_INT | |
| SMC_LOCK_T_EX_PAGE | |
| SMC_LOCK_T_SH_PAGE | |
| SMC_LOCK_T_UP_PAGE | |

Enum: SMC_OBJ_TYPE

An enum to identify object types:

| Table B-22: Object type | |
|-------------------------|--|
| SMC_OBJ_T_STP | |
| SMC_OBJ_T_TBL | |

Enum: SMC_PROC_STATE

An enum to identify process states:

Table B-23: Process state

| |
|----------------------------|
| SMC_PROC_STATE_ALARM_SLEEP |
| SMC_PROC_STATE_BACKGROUND |
| SMC_PROC_STATE_BAD_STATUS |
| SMC_PROC_STATE_INFECTED |
| SMC_PROC_STATE_LOCK_SLEEP |
| SMC_PROC_STATE_RECV_SLEEP |
| SMC_PROC_STATE_RUNNABLE |
| SMC_PROC_STATE_RUNNING |
| SMC_PROC_STATE_SEND_SLEEP |
| SMC_PROC_STATE_SLEEPING |
| SMC_PROC_STATE_STOPPED |
| SMC_PROC_STATE_TERMINATING |
| SMC_PROC_STATE_YIELDING |
| SMC_PROC_STATE_REMOTE_IO |
| SMC_PROC_STATE_SYNC_SLEEP |

Enum: SMC_PROP_ACTION

An enum used to identify the desired action of an `smc_connect_props` function call:

Table B-24: Connection property action

| |
|--------------------|
| SMC_PROP_ACT_SET |
| SMC_PROP_ACT_GET |
| SMC_PROP_ACT_CLEAR |

Enum: SMC_PROP_TYPE

An enum used to identify the property to operate on in a call to `smc_connect_props`:

Table B-25: Connection property

| |
|-------------------------|
| SMC_PROP_APPNAME |
| SMC_PROP_ERROR_CALLBACK |

| |
|------------------------|
| SMC_PROP_IFILE |
| SMC_PROP_LOGIN_TIMEOUT |
| SMC_PROP_PACKETSIZE |
| SMC_PROP_PASSWORD |
| SMC_PROP_SERVERMODE |
| SMC_PROP_SERVERNAME |
| SMC_PROP_TIMEOUT |
| SMC_PROP_USERDATA |
| SMC_PROP_USERNAME |

Enum: SMC_RETURN_CODE

An enum to identify the types of return codes:

Table B-26: Return codes

| |
|----------------------------------|
| SMC_RET_SUCCESS |
| SMC_RET_FAILURE |
| SMC_RET_INSUFFICIENT_MEMORY |
| SMC_RET_CONNECTION_ERROR |
| SMC_RET_UNABLE_TO_CONNECT_TO_SMS |
| SMC_RET_UNABLE_TO_CONNECT_TO_SS |
| SMC_RET_MISSING_RESULT_TABLE |
| SMC_RET_INVALID_USER_PASSWD |
| SMC_RET_INVALID_PARAMETER |
| SMC_RET_INVALID_CACHE |
| SMC_RET_INVALID_DCID |
| SMC_RET_INVALID_COMMAND |
| SMC_RET_INVALID_VIEWID |
| SMC_RET_INVALID_DINAME |
| SMC_RET_INVALID_DISTAT |
| SMC_RET_INVALID_DI_STRUCT |
| SMC_RET_DI_STAT_MISMATCH |
| SMC_RET_INVALID_DI_COMBO |
| SMC_RET_INVALID_DATATYPE |
| SMC_RET_INVALID_VALUE_COUNT |
| SMC_RET_INVALID_FILTER_VALUE |

| |
|-----------------------------------|
| SMC_RET_INVALID_FILTER_RANGE |
| SMC_RET_DATAITEM_CONTAINS_FILTER |
| SMC_RET_INVALID_COMPOSITE_FILTER |
| SMC_RET_INVALID_SVR_MODE |
| SMC_RET_MISSING_DATAITEM |
| SMC_RET_INVALID_FILTERID |
| SMC_RET_INVALID_ALARMID |
| SMC_RET_INVALID_ALARM_VALUE |
| SMC_RET_INVALID_DINAME_FOR_ALARM |
| SMC_RET_INVALID_API_FUNC_SEQUENCE |
| SMC_RET_INVALID_API_FUNCTION |
| SMC_RET_INVALID_PROPERTY |
| SMC_RET_INVALID_INFOTYPE |
| SMC_RET_CONNECT_NOT_CLOSED |
| SMC_RET_ARITHMETIC_OVERFLOW |
| SMC_RET_LOGIN_LACKS_SA_ROLE |
| SMC_RET_INTERNAL_ERROR |

Enum: SMC_SERVER_MODE

An enum to identify the types of Adaptive Server Enterprise Monitor connections:

Table B-27: Server mode type

| |
|-------------------------|
| SMC_SERVER_M_LIVE |
| SMC_SERVER_M_HISTORICAL |

Enum: SMC_SOURCE

An enum used in conjunction with ErrorCallback to identify the source of an error, warning or informational notification.

Table B-28: Error source

| |
|-----------------|
| SMC_SRC_UNKNOWN |
| SMC_SRC_HS |

SMC_SRC_SMC

SMC_SRC_CT

SMC_SRC_SS

SMC_SRC_SMS

Union: SMC_VALUE_UNION

A union used in conjunction with `smc_connect_props`, `smc_get_command_info`, and `smc_get_dataitem_value` to set and retrieve results.

typedef union SMC_VALUE_UNION {

| | |
|---------|-----------------|
| SMC_INT | <i>intValue</i> |
|---------|-----------------|

| | |
|----------|------------------|
| SMC_LONG | <i>longValue</i> |
|----------|------------------|

| | |
|------------|--------------------|
| SMC_DOUBLE | <i>doubleValue</i> |
|------------|--------------------|

| | |
|-----------|------------------|
| SMC_SIZET | <i>sizeValue</i> |
|-----------|------------------|

| | |
|-----------|--------------------|
| SMC_CHARP | <i>stringValue</i> |
|-----------|--------------------|

| | |
|-----------|-------------------|
| SMC_VOIDP | <i>voidpValue</i> |
|-----------|-------------------|

| | |
|--------------|----------------------|
| SMC_DATETIME | <i>datetimeValue</i> |
|--------------|----------------------|

| | |
|--------------------|--|
| } SMC_VALUE_UNION; | |
|--------------------|--|

Backward Compatibility

Monitor Client Library version 11.5 and later replaces several API functions. The new API and callback functions provide improved features and extensibility. Replaced API and callback functions have been preserved within the library for backwards compatibility.

Obsolete and replacement functions

Table C-1 maps obsolete Monitor Client Library functions to their replacement functions:

Table C-1: Obsolete functions and replacement functions

| Obsolete | Replacement |
|--------------------------|--|
| smc_change_error_handler | smc_connect_props |
| smc_connect | smc_connect_alloc smc_connect_props smc_connect_ex |
| smc_create_alarm | smc_create_alarm_ex |
| smc_disconnect | smc_close smc_connect_drop |
| smc_refresh | smc_refresh_ex |

The most significant syntactic difference between the obsolete and replacement functions is the callback function parameter. In earlier versions, `SMC_CALLBACK`, `SMC_ALARM_CALLBACK`, and `SMC_ERR_CALLBACK` were used to specify a callback function. These callback function types have been replaced by `SMC_GEN_CALLBACK`.

Note The new refresh function, `smc_refresh_ex`, does not use any callback function, unlike the obsolete `smc_refresh`.

In addition to changing the callback function types, `smc_connect` and `smc_disconnect` have been replaced by a set of functions that allow for greater flexibility and control.

New functions

Table C-2 lists the new functions.

Table C-2: New functions

| |
|--|
| <code>smc_create_playback_session</code> |
| <code>smc_get_command_info</code> |
| <code>smc_initiate_playback</code> |
| <code>smc_terminate_playback</code> |
| <code>smc_terminate_recording</code> |

Note New functions cannot be used with obsolete functions.

Rules for functions and callbacks compatibility

Use the following rules to decide which functions and callbacks can be used together:

- If you are using any new or replacement functions, do not use obsolete functions.
- If you are using obsolete functions, use the obsolete error callback function types.
- If you are using replacement or new functions, use the version 11.1 error callback function types.
- You can use unchanged functions with all other types of functions.

Troubleshooting Information and Error Messages

This appendix contains two sections.

- “Troubleshooting” lists problems that you may encounter using Monitor Client Library, but that do not have specific error messages.
- “Error Messages” describes error messages that you may receive.

Troubleshooting

Confusing messages from Adaptive Server

If you create a view that requires information from a database that needs to be recovered, you get error messages from Adaptive Server rather than a concise error message from Monitor Client Library.

View refreshes fail

- If you try to refresh a view at the same time as someone creates a database, the refresh may fail.
- A refresh for a view may fail if one or more databases on Adaptive Server are in single user mode.

Negative numbers as object IDs

If you create a view using the SMC_NAME_OBJ_ID data item, you might see negative numbers as object IDs. Negative object IDs are an accurate reporting of IDs as assigned by Adaptive Server.

Monitor Server reports on *all* activity, including activity on temporary tables that Adaptive Server creates to perform a complex query. The object IDs that Adaptive Server assigns to temporary tables can be positive or negative. The object ID that was assigned by Adaptive Server is reported.

In views that show SMC_NAME_OBJ_NAME, the string ****TempObject**** is reported for temporary tables.

Error messages

Monitor Client Library is an Open Server application that uses the Open Client Library to communicate with Adaptive Server and Monitor Server. Any of these components can detect and report errors conditions. Monitor Client Library also detects and reports error conditions, which it logs or reports or both to clients.

The following building, linking, and compiling error messages may be reported. They are listed here in alphabetical order.

Communication failure: check if server is running.

While running testmon.exe, one of the following conditions caused the error to be reported:

- Server names are incorrect in *example.h*.
- *sql.ini* file is missing.
- *sql.ini* file has incorrect network connection information.
- Adaptive Server is not running.
- Historical Server is not running.
- User name is incorrectly set in *example.h*.
- Password for the user name is incorrectly set in *example.h*.

Configuration failure: possibly missing *interfaces* file or bad login parameters.

While running testmon.exe, one of the following conditions caused the error to be reported:

- Server names are incorrect in *example.h*.
- *sql.ini* file is missing.
- *sql.ini* file has incorrect network connection information.
- Adaptive Server is not running.
- Historical Server is not running.
- User name is incorrectly set in *example.h*.

- Password for the user name is incorrectly set in *example.h*.

Don't know how to build *example.h*

While building *testmon.exe*, one of the following conditions caused the compile error to be reported:

- Project needs to rebuild all dependencies.
- Project's *include* file path needs the location of the file names.
- Default location would be *C:\SYBASE\INCLUDE* and *C:\SYBASE\MONCLT\SAMPLE*.

error L2029: 'SMC_CONNECT' : unresolved external

While building *testmon.exe*, the following condition caused the link error to be reported:

- *smcapi32.lib* needs to be included as one of the libraries in which to link. It is located by default in *C:\SYBASE\MONCLT\LIB*.

error L2029: 'SMC_CREATE_VIEW' : unresolved external

While building *testmon.exe*, the following condition caused the link error to be reported:

- You need to include *smcapi32.lib* as one of the libraries in which to link. It is located by default in *C:\SYBASE\MONCLT\LIB*.

fatal error C1083: Cannot open include file: 'cstypes.h': No such file or directory

While building *testmon.exe*, one of the following conditions caused the compile error to be reported:

- Project needs to rebuild all dependencies.
- Project's *include* file path needs the location of the file names.

- Default location would be *C:\SYBASE\INCLUDE* and *C:\SYBASE\MONCLT\SAMPLE*.

fatal error C1083: Cannot open include file: 'mcpublic.h': No such file or directory

While building testmon.exe, the following condition caused the compile error to be reported:

- Project's *include* path for the preprocessor needs to be edited to the correct setting. It is currently set to *C:\SYBASE\MONCLT\INCLUDE*; *C:\SYBASE\INCLUDE*.

LINK: fatal error L4051: smcapi32.lib : cannot find library

While building testmon.exe, the following condition caused the link error to be reported:

- The project's Library File's path needs to include the location of *smcapi32.lib*, which is assumed to be in *C:\SYBASE\MONCLT\LIB*.

Index

A

- activity
 - none 45
- alarm callback syntax 143
- alarms
 - adding 125
 - callback functions 11, 128
 - creating 140
 - removing 125
 - retrieve information 162
 - setting 11
- allocating
 - connection structure 6
- application programming interface 2
- architecture
 - SQL Server Monitor 2
- average
 - statistical types 8
- average, statistic type
 - definition of 9

C

- calculation
 - statistical type 8
- callback function 11, 128
- cancelling
 - recording session 175
- client connection 6
- command info types 129, 165
- command structure
 - deallocating 13
- commands
 - isql 4
- compiling 179
 - UNIX 180
 - Windows 182
- configuring

- Monitor Server 3
- SQL Server 3
- SQL Server Monitor 2
- connecting
 - server 6
- connection
 - closing 125
 - creating 125, 131
 - deallocating 125, 132
 - establishing 125, 134
 - initialize playback 125
 - Monitor 131
 - properties 135, 140
 - reusing 13
 - setting properties 125
- connection structure
 - allocating 6
 - deallocating 13
- connections
 - summaries 46
- creating
 - filters 10

D

- data item
 - defined 43
 - definition 7
- data item statistical type 8
- data item type
 - returning 125
- data items
 - list of 47
 - result and key 43
 - retrieving 126
 - SMC_NAME_ACT_STP_DB_ID 49
 - SMC_NAME_ACT_STP_DB_NAME 50
 - SMC_NAME_ACT_STP_ID 50
 - SMC_NAME_ACT_STP_NAME 51

SMC_NAME_ACT_STP_OWNER_NAME 51
 SMC_NAME_APP_EXECUTION_CLASS 52
 SMC_NAME_APPLICATION_NAME 52
 SMC_NAME_BLOCKING_SPID 53
 SMC_NAME_CONNECT_TIME 54
 SMC_NAME_CPU_BUSY_PCT 54
 SMC_NAME_CPU_PCT 55
 SMC_NAME_CPU_TIME 55
 SMC_NAME_CPU_YIELD 56
 SMC_NAME_CUR_APP_NAME 56
 SMC_NAME_CUR_ENGINE 56
 SMC_NAME_CUR_EXECUTION_CLASS 57
 SMC_NAME_CUR_PROC_STATE 57
 SMC_NAME_CUR_STMT_ACT_STP_DB_ID 58
 SMC_NAME_CUR_STMT_ACT_STP_DB_NAME
 59
 SMC_NAME_CUR_STMT_ACT_STP_ID 59
 SMC_NAME_CUR_STMT_ACT_STP_NAME 60
 SMC_NAME_CUR_STMT_ACT_STP_OWNER_NAME
 60
 SMC_NAME_CUR_STMT_ACT_STP_TEXT 61
 SMC_NAME_CUR_STMT_BATCH_ID 61
 SMC_NAME_CUR_STMT_BATCH_TEXT 62
 SMC_NAME_CUR_STMT_BATCH_TEXT_ENABLE
 D 62
 SMC_NAME_CUR_STMT_CONTEXT_ID 63
 SMC_NAME_CUR_STMT_CPU_TIME 63
 SMC_NAME_CUR_STMT_ELAPSED_TIME 63
 SMC_NAME_CUR_STMT_LINE_NUM 64
 SMC_NAME_CUR_STMT_LOCKS_GRANTED_IM
 MED 65
 SMC_NAME_DATA_CACHE_HIT_PCT 71
 SMC_NAME_DATA_CACHE_ID 71
 SMC_NAME_DATA_CACHE_NAME 74
 SMC_NAME_LOCK_RESULT_SUMMARY 86
 SMC_NAME_LOCK_STATUS 86
 SMC_NAME_LOCK_STATUS_CNT 87
 SMC_NAME_LOCKS_BEING_BLOCKED_CNT
 88
 SMC_NAME_OBJ_NAME 101
 SMC_NAME_OWNER_NAME 101
 SMC_NAME_PROC_STATE_CNT 107
 data refresh 12, 173
 deallocating
 connection structure 13
 detail

specifying in view 43
 details
 server-wide data 44

E

empty rows 45
 views, in 45
 error handler 127
 error handling 127
 error messages
 callback function 128
 Monitor Historical Server 244
 error notification 162

F

filters
 adding 125, 144
 creating 10
 removing 125, 160
 types 10
 function summary 126
 functions
 using threads 126

G

graphical user interface 2

H

Historical Server 2, 3
 cancel session 175
 isql interface to 4
 Monitor Client Library and
 playback in 4

I

information types 129, 165

callback data 129
 isql
 Historical Sever and 4

K

key data items
 defined 43

L

linking 179
 UNIX 180
 Windows 183

M

Monitor Client Library 2
 definition of 1
 Historical Server and 4
 playback 4
 properties 138
 relationship to Monitor Server 3
 Monitor Historical Server
 connection 125
 definition of 2
 messages 244
 Monitor Server 2
 summaries 46
 Monitor Viewer 2

O

Open Server 2

P

performance 3
 performance data 12
 playback 4
 conclude definition 126

conclude session 174
 creating a session 148
 ending a session 126
 initializing 125
 program structure
 closing connections 13
 connecting to a server 6
 creating filters 10
 creating views 7
 deallocating connections 13
 setting alarms 11
 properties
 clearing 140
 connection 140
 retrieving 140
 setting 140

R

rate
 statistical types 8
 recording
 conclude definition 126
 creating a session 154
 initializing 125
 initiating session 172
 recording, initiating 126
 refresh data 12, 173
 reopen a connection 13
 result data items
 defined 43
 return values 127
 row count
 retrieving 126
 rows
 empty 45

S

sample
 statistical types 8
 sample applications 179
 UNIX 182
 Windows 184

Index

- servers
 - connecting to 6
 - logging into 7
- server-wide data
 - details of 44
- session
 - cancelling 175
 - statistical types 8
- session, creating 125
- setting
 - alarms 11
- shared memory 3
- smc_close 125, 130
- smc_connect_alloc 125, 131
 - see also connection structure 6
- smc_connect_drop 125, 132
- smc_connect_ex 7, 13, 125, 133
- smc_connect_props 6, 125, 135
- smc_create_alarm 11
- smc_create_alarm_ex 125, 140
- smc_create_filter 10, 125, 144
- smc_create_playback_session 125, 148
- smc_create_recording_session 125, 153
- smc_create_view 9, 125, 156
- smc_drop_alarm 125, 158
- smc_drop_filter 125, 160
- smc_drop_view 125, 161
- smc_get_command_info 125, 162
- smc_get_dataitem_type 125, 165
- smc_get_dataitem_value 12, 126, 166
- smc_get_row_count 12, 126, 168
- smc_get_version_string 126, 169
- smc_initiate_playback 126
- smc_initiate_recording 126, 172
- SMC_NAME_ACT_STP_DB_ID 49
- SMC_NAME_ACT_STP_DB_NAME 50
- SMC_NAME_ACT_STP_ID 50
- SMC_NAME_ACT_STP_NAME 51
- SMC_NAME_ACT_STP_OWNER_NAME 51
- SMC_NAME_APP_EXECUTION_CLASS 52
- SMC_NAME_APPLICATION_NAME 52
- SMC_NAME_BLOCKING_SPID 53
- SMC_NAME_CONNECT_TIME 54
- SMC_NAME_CPU_BUSY_PCT 54
- SMC_NAME_CPU_PCT 55
- SMC_NAME_CPU_TIME 55
- SMC_NAME_CPU_YIELD 56
- SMC_NAME_CUR_APP_NAME 56
- SMC_NAME_CUR_ENGINE 56
- SMC_NAME_CUR_EXECUTION_CLASS 57
- SMC_NAME_CUR_PROC_STATE 57
- SMC_NAME_CUR_STMT_ACT_STP_DB_ID 58
- SMC_NAME_CUR_STMT_ACT_STP_DB_NAME 59
- SMC_NAME_CUR_STMT_ACT_STP_ID 59
- SMC_NAME_CUR_STMT_ACT_STP_NAME 60
- SMC_NAME_CUR_STMT_ACT_STP_OWNER_NAME 60
- SMC_NAME_CUR_STMT_ACT_STP_TEXT 61
- SMC_NAME_CUR_STMT_BATCH_ID 61
- SMC_NAME_CUR_STMT_BATCH_TEXT 62
- SMC_NAME_CUR_STMT_BATCH_TEXT_ENABLED 62
- SMC_NAME_CUR_STMT_CONTEXT_ID 63
- SMC_NAME_CUR_STMT_CPU_TIME 63
- SMC_NAME_CUR_STMT_ELAPSED_TIME 63
- SMC_NAME_CUR_STMT_LINE_NUM 64
- SMC_NAME_CUR_STMT_LOCKS_GRANTED_IMMED 65
- SMC_NAME_DATA_CACHE_HIT_PCT 71
- SMC_NAME_DATA_CACHE_ID 71
- SMC_NAME_DATA_CACHE_NAME 74
- SMC_NAME_LOCK_RESULT_SUMMARY 86
- SMC_NAME_LOCK_STATUS 86
- SMC_NAME_LOCK_STATUS_CNT 87
- SMC_NAME_LOCKS_BEING_BLOCKED_CNT 88
- SMC_NAME_OBJ_NAME 101
- SMC_NAME_OWNER_NAME 101
- SMC_NAME_PROC_STATE_CNT 107
- smc_refresh_ex 12, 126, 173
- SMC_STAT_AVG_SESSION
 - definition of 9
- SMC_STAT_RATE_SAMPLE
 - definition of 8
- SMC_STAT_RATE_SESSION
 - definition of 9
- SMC_STAT_VALUE_SAMPLE
 - definition of 8
- SMC_STAT_VALUE_SESSION
 - definition of 8
- smc_terminate_playback 126, 174

- smc_terminate_recording 175
- specifying
 - detail in view 43
- SQL Server Monitor
 - architecture 2
 - components 2
 - definition 1
- statistical type 8
- structures
 - allocating a connection structure 6
- summaries
 - connection 46
- Sybase Central 3

T

- terminating playback 174
- testhist 179
- threads 126
- triggering
 - alarms 11
- ty 140

V

- value
 - statistical type 8
- version number 126
- view
 - contents 44
 - description 9
- views 7
 - alarms 11
 - amount of detail 43
 - defining 125
 - definition 7
 - dropping 125, 161
 - empty rows 45
 - filters on views 10
 - monitor summaries 46
 - retrieving data 126
 - sampling data 173

