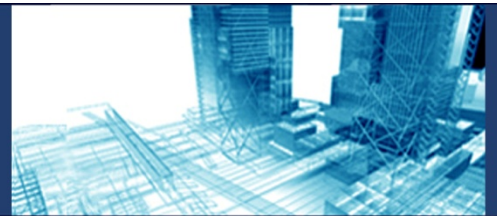


A Broadcom Company

Smart / RESTART



***Relational
Architects
International***



User Guide

Version 12.1.0

March 2017

Publication SJS-004-45

Relational Architects Intl

This Guide:

RAI Publication SJS-004-45

This document applies to **Smart/RESTART** Version 12 Release 1.0 (March 2017) and all subsequent releases, unless otherwise indicated in new editions or technical newsletters. Specifications contained herein are subject to change and will be reported in subsequent revisions or editions.

Requests for publications should be addressed to:

Documentation Coordinator
Relational Architects Intl.
Riverview Historic Plaza
33 Newark Street
Hoboken NJ 07030 USA

Tel: (USA) 201 420-0400
Fax: (USA) 201 420-4080

www.relarc.com

Reader comments regarding the product, its documentation, and suggested improvements are welcome. A reader comment form is provided at the back of this publication.

Copyright (c) 1991, 2017 by Relational Architects International (RAI) as an unpublished work -- all rights reserved. The software product(s) described herein and the documentation thereof are trade secrets and property of Relational Architects International. All use, disclosure, duplication, electronic storage, reproduction or transmission without specific written authorization from RAI is strictly prohibited. These products may also be protected under the trade-secret and copyright laws of countries other than the United States and international agreements.

DB2, ISPF, IMS, WebSphere MQ, z/OS and OS/390 are software products of International Business Machines Corporation. CA-7 and CA-11 are trademarks of Computer Associates, Inc.

The Smart Jobstream Series, Smart/RRSAF, Smart/CAF, Smart/RESTART, Smart/PRECOMPILER, Smart/MONITOR, Smart/PREDICTOR, Smart/LOGGER and Smart/QBF are trademarks of Relational Architects Intl.

Ed 17C9

Preface

This Smart/RESTART User Guide is intended for analysts, programmers and operators concerned with developing and running restartable batch applications. Readers are referred as well to the following Smart/RESTART companion publications:

Smart/RESTART Reference	SJS-005
Smart/RESTART Messages Reference	SJS-006
Smart/RESTART General Information	SJS-009
Smart Jobstream Series Installation Guide	SJS-001

In addition, DB2 application developers should be familiar with

Smart/RRSAF User Guide and Reference	DCA-002
--------------------------------------	---------

Smart/RESTART delivers restart capability that's robust, reliable and simple -- to minimize the impact of batch application failures. Without complicated restart logic or cumbersome operational procedures. Smart/RESTART *guarantees* that updates to DB2, MQ, IMS and other RRS compliant resources stay in synch with your program's sequential file and cursor position, working storage and random VSAM updates. This enables your applications to restart from near the point of failure after abends, recompiles, even system IPLs -- with *all* resources in a consistent state.

This manual describes how to design, develop, prepare and run restartable applications which access sequential files, VSAM datasets as well as any RRS compliant resources such as DB2 tables, IMS databases and WebSphere MQ queues. This material is applicable to new development as well as to retrofitting legacy applications for restartable operation. This manual is organized as follows:

Part I presents Smart/RESTART fundamentals and describes its use in applications which access sequential files, DB2 and other RRS compliant resources. Chapter 1 introduces Smart/RESTART Concepts and Facilities. Chapter 2 discusses in detail how to design and code restartable applications. It describes how Smart/RESTART handles the mechanics of checkpoint/restart and discusses the developer's responsibilities in enabling successful restart. Chapter 3 addresses program preparation for restartable applications (optionally using Smart/PRECOMPILER) while Chapter 4 describes the execution and restart of restartable jobs.

Since Smart/PRECOMPILER requires an additional jobstep in your program preparation JCL, Chapter 5 describes and illustrates how restart enablement can also be achieved through the use of COBOL copybooks or PL/1 INCLUDE text. Some organizations may prefer this technique to the 'Precompiler Transparency' approach illustrated in Chapter 2. RAI recommends both Smart/PRECOMPILER and COPY/INCLUDE techniques with equal weight.

Part II describes additional modes of Smart/RESTART usage including the EXEC SRS interface (a high level interface analogous to EXEC SQL), its Call Level Interface, and a mode of operation termed Object Transparency.

Part III documents Smart/RESTART support for applications which access a variety of resource managers – with and without the use of RRS as a synchpoint coordinator. Separate chapters describe how restartable applications can access WebSphere MQ or IMS resources in 'standalone mode' as well as the use of DB2 attachment facilities *other than* Smart/RRSAF (a separate, companion product of Smart/RESTART).

Part IV presents Smart/MONITOR, an ISPF dialog with which to monitor and control your restartable applications.

Part V describes a set of advanced Smart/RESTART features -- such as support for Assembler Language applications and Automatic Restart.

Lastly, **Part VI** describes Smart/LOGGER for VSAM, which provides automated restart capabilities for applications that access VSAM data and optionally RRS compliant resources as well.

Refer to the Table of Contents for additional topics that may be relevant at your site.

Notational Conventions

The following notational conventions are used in this User Guide:

- Uppercase commands and their operands should be entered as shown but need not be entered in uppercase.
- Operands shown in lower case are variables; a value should be substituted for them.
- Operands shown in brackets [] are optional, with a choice indicated by vertical bars |. One or none may be chosen; the defaults are underscored.
- Operands shown in braces { } are alternatives; one must be chosen.
- An ellipsis (...) indicates that the parameter shown may be repeated to specify additional items of the same category.

Table of Contents

Preface.....	iii
Notational Conventions.....	iv
 Section A: What's New in Smart/RESTART	A-1
What's New in Smart/RESTART V12.1.0	A-1
Smart/RESTART Recovery Services Updates.....	A-4
Smart/RESTART Utilities Updates.....	A-5
Smart/RESTART API Services Updates.....	A-5
New Smart/RESTART Exit Routines	A-6
Smart/RESTART IMS Features	A-7
What's New in Smart/MONITOR V11.1	A-7
Smart/RESTART Interoperability With Other IBM and ISV Products	A-8
 Chapter 1: Smart/RESTART Concepts and Facilities.....	1-1
1.1 Background and context	1-1
1.1.1 Application Restart.....	1-2
1.1.2 Checkpoints and Units-of-work	1-2
1.2 What does Smart/RESTART do?	1-3
1.2.1 Benefits of Smart/RESTART	1-4
1.3 Smart/RESTART Architecture and Components	1-5
1.4 Method of Operation.....	1-6
1.4.1 What happens during Restart?.....	1-7
1.4.2 Checkpoint Pacing and Performance	1-7
1.5 When is Restart Enabled?	1-8
1.6 Smart/RESTART Features.....	1-9
1.7 Customization	1-10
1.8 The Smart/RESTART Sample Library	1-10
 Chapter 2: Developing Restartable Applications.....	2-1
2.1 Summary of Application Responsibilities for Restart.....	2-1
2.2 Pseudo Code Example	2-2

2.3	Sample DB2/COBOL Application.....	2-4
2.3.1	Mainline Logic	2-7
2.3.2	Initialization Routines	2-7
2.3.2.1	Open / Reopen files	2-7
2.3.2.2	Do not initialize areas restored by Smart/RESTART	2-8
2.3.3	The Main Loop.....	2-11
2.3.3.1	Checkpoint Pacing and Performance	2-12
2.3.4	An Individual transaction	2-12
2.4	SQL Cursor Processing	2-13
2.5	Repositioning Input Files.....	2-14
2.5.1	Control Break Processing Model -- Reposition AT	2-14
2.5.2	Record Processed Model - Reposition AFTER.....	2-16
2.6	Applications Comprised of Multiple Modules.....	2-17
2.7	General restrictions on Smart/RESTART usage	2-19
2.8	Restrictions on Smart/RESTART's Repositionable Sequential Access Method	2-19

Chapter 3: Restartable Program Preparation 3-1

3.1	Illustrative Program Preparation Jobstream	3-3
3.2	DB2 Precompile [DB2 dependent section]	3-5
3.3	Smart/PRECOMPILER	3-5
3.3.1	JCL for Smart/PRECOMPILER	3-6
3.3.2	Input to Smart/PRECOMPILER	3-7
3.3.3	Output from Smart/PRECOMPILER.....	3-7
3.3.4	Smart/PRECOMPILER Run-time Options	3-8
3.3.5	PRECOMPILER Return Codes	3-14
3.4	Compiling your program.....	3-15
3.5	Link Editing your Program	3-17
3.5.1	Using Explicit Linkage Editor INCLUDE Statements.....	3-18
3.5.2	Using Automatic Library Call.....	3-20
3.6	BIND your Application Plan [DB2-dependent Section]	3-20
3.6.1	BIND DBRM for SRSDBRM [DB2-dependent Section]	3-20
3.6.2	BIND Package for SRSDBRM [DB2-dependent Section].....	3-20
3.6.3	Package BIND for Applications that issue SQL SET CURRENT PACKAGESET statements [DB2-dependent Section]	3-21

Chapter 4:	Program Execution and Restart	4-1
4.1	JCL for Initial and Restart Runs	4-1
4.1.1	Load Module Libraries.....	4-3
4.1.2	Defining the Smart/RESTART Checkpoint Dataset	4-4
4.1.2.1	Calculating the Checkpoint Size.....	4-5
4.1.2.2	Dynamic Allocation of Checkpoint Files	4-8
4.1.3	Dynamic Allocation of the SRSPRINT File	4-8
4.1.4	Pre-allocating Application Files.....	4-9
4.1.5	SYSIN, SYSOUT and DUMMY files	4-9
4.1.6	Smart/RESTART Run-time Parameters.....	4-10
4.1.6.1	Using a Smart/RESTART Run-time Parameter Library	4-11
4.1.6.2	Sharing Smart/RESTART Run-time Parameters.....	4-12
4.2	Application Restart	4-13
4.3	Development and Testing Procedures.....	4-13
4.4	Communicating with Smart/RESTART during Execution	4-14
4.4.1	Production Operation	4-14
4.4.2	Cancelling Restartable Jobs	4-15
4.5	Out of Space Failures.....	4-15
4.6	The SRSUTIL Utility.....	4-17
4.6.1	SRSUTIL invocation parameters	4-18
4.6.1.1	ABEND	4-18
4.6.1.2	CLEANUP.....	4-18
4.6.1.3	IGNORE	4-18
4.6.1.4	RC	4-18
4.6.1.5	RESET.....	4-19
4.6.1.6	ENVCHECK	4-19
4.6.1.7	DUMP	4-19
4.6.1.8	LOCATE_SERVER=<rai_server_name>	4-19
4.6.1.9	COPY	4-20
4.6.1.10	PRINT_MESSAGE	4-20
4.7	The SRSULS Utility	4-21
4.8	Releasing Unused Space	4-21
4.9	Deleting Checkpoint Data with the SRSUDEL Utility.....	4-22
4.9.1	SRSUDEL invocation parameters.....	4-23

Chapter 5:	Restart Enablement via COPY / INCLUDE Code.....	5-1
5.1	COBOL COPY member Example	5-1
5.2	Identifying Critical Working Storage to Smart/RESTART	5-7
5.3	Distinguishing between Initial and Restart Runs	5-7
5.4	COBOL COPY Members	5-8
5.5	PL/I Support.....	5-8
5.6	PL/I INCLUDEs supplied with Smart/RESTART.....	5-10
Chapter 10:	The Smart/RESTART EXEC SRS Interface.....	10-1
10.1	EXEC SRS Syntax	10-1
10.2	Sample DB2/COBOL Application.....	10-2
10.3	What Storage Areas Should be Checkpointed?.....	10-5
10.3.1	Examples.....	10-5
10.3.2	Default Storage Areas	10-6
10.4	INITIALIZE Service and Dynamically Acquired Storage.....	10-7
10.5	Distinguishing between Initial and Restart Runs	10-7
10.6	TERMINATE Service	10-8
10.7	Checkpointing with Smart/RESTART.....	10-8
10.8	Rollback with Smart/RESTART.....	10-9
Chapter 11:	Coding Restartable Applications with the Smart/RESTART Call Level Interface.....	11-1
11.1	The Smart/RESTART API	11-1
11.2	Call Level Interface Example	11-2
11.3	The INITIALIZE service	11-6
11.3.1	The INITIALIZE service -- Format I.....	11-6
11.3.2	The INITIALIZE service - Format II.....	11-8
11.4	Distinguishing between Initial and Restart Runs	11-8
11.5	The TERMINATE Service	11-9
11.6	The CHECKPOINT Service.....	11-9
11.7	The ROLLBACK Service.....	11-11
11.8	Designating an Explicit Restart Point	11-12

Chapter 12:	Object Transparency	12-1
12.1	Requirements and Restrictions on Object Transparency	12-2
12.2	Object Transparency Example	12-2
12.3	JCL to run the Object Transparent Application	12-5
12.4	Implementing Object Transparency	12-6
Chapter 20:	Resource Recovery Services and Applications that Access Multiple Resource Managers.....	20-1
Chapter 21:	Additional DB2 Topics	21-1
21.1	Set Current Packageset	21-1
21.2	DB2 Savepoints	21-2
21.3	DB2 Stored Procedures.....	21-2
Chapter 22:	DB2 Applications without Smart/RRSAF.....	22-1
22.1	Requesting a Call Attach facility other than Smart/RRSAF	22-1
22.1.1	The RMI Parameter.....	22-2
22.2	Application Development Considerations	22-3
22.2.1	SQL COMMIT and ROLLBACK statements.....	22-3
22.2.2	DB2 BIND	22-3
22.3	Smart/RESTART Installation Considerations	22-4
22.3.1	Using Other DB2 Attachment Facilities	22-4
22.4	Smart/RESTART Options Which Require Smart/RRSAF	22-7
Chapter 23:	Smart/RESTART for IMS and Mixed IMS / DB2 Applications.....	23-1
23.1	Checkpoint Restart in the IMS DB2 Environment.....	23-2
23.2	Application Design and Coding	23-2
23.2.1	Checkpoint, Rollback and Restart.....	23-3
23.2.2	PCB Addressability.....	23-3
23.2.3	Sequential File I/O	23-4
23.3	Program Preparation for IMS / DB2 Applications.....	23-4
23.3.1	Run Smart/PRECOMPILER.....	23-4
23.3.2	Link Edit your Application	23-5
23.3.3	Bind the Smart/RESTART DBRM into your DB2 application plan.....	23-6
23.3.4	Converting IMS Extended Restart applications to Smart/RESTART	23-6

23.4	Execution JCL for IMS and Mixed IMS / DB2 applications	23-8
23.4.1	STEPLIB Concatenation	23-8
23.4.2	DLIBATCH Considerations.....	23-9
23.5	Smart/RESTART run-time parameters for IMS	23-9
23.5.1	DB2 Attachment Facility Specification	23-9
23.5.2	Specify the Smart/RESTART Resource Manager Interface Module	23-9
23.6	Sample Applications and Jobstreams.....	23-10
23.7	Sample DB2 / IMS COBOL Application.....	23-10
23.8	Sample DLIBATCH JCL.....	23-16
23.9	Sample JCL for an IMS BMP	23-18

Chapter 24: Smart/RESTART Support for WebSphere MQ 24-1

24.1	Application Restrictions and Requirements.....	24-1
24.2	Linking the WebSphere MQ interface routine.....	24-4
24.2.1	Linking the MQ interface routine for the RRS Environment.....	24-5
24.2.2	Linking the MQ interface for the Standalone MQ Environment.....	24-6
24.3	Run-time Requirements	24-7
24.4	Application Managed CONNECT and DISCONNECT	24-7

Chapter 30: Smart/MONITOR 30-1

30.1	Overview.....	30-1
30.1	How to establish a Smart/MONITOR on-line session	30-2
30.2	Operating Levels.....	30-2
	Basic Mode	30-2
	Extended Mode	30-3
30.3	Smart/MONITOR Scrollable Job Summary	30-3
30.4	Job specific Row Commands.....	30-4

Chapter 31: Smart/MONITOR Setup 31-1

31.1	The CONFIG Command.....	31-1
31.1.1	Initial Display View	31-2
31.1.2	Miscellaneous Section.....	31-3
31.1.3	Specify DB2 - Related Parameters.....	31-3
31.1.4	Identify Checkpoint Synchronization Structure	31-4
31.1.5	Smart/MONITOR DB2 environment setup parameters	31-4

31.2	CONFIG Command to configure Smart/MONITOR.....	31-5
31.2.1	AUTOMODE Setup.....	31-5
31.2.2	Support for Monitoring Multiple DB2 Subsystems	31-6
31.2.3	Debugging and Message Control	31-6
31.3	Smart/MONITOR Security	31-7
Chapter 32: Smart/MONITOR Displays.....		32-1
32.1	Job Summary Display	32-1
32.2	Jobstep Detail.....	32-2
32.3	Repositionable File I/O	32-4
32.4	SQL Summary	32-5
32.5	SQL Statistics Detail.....	32-6
32.6	Smart/RESTART Statistics.....	32-7
Chapter 33: Smart/MONITOR Primary Commands		33-1
33.1	AUTOMODE Command	33-2
33.2	CONFIG Command.....	33-2
33.3	DEBUG Command	33-3
33.4	DFORMAT Command	33-3
33.5	DUMP Command	33-3
33.6	FULL / HALF Commands.....	33-4
33.7	SORT Command.....	33-4
33.8	TRACE Command.....	33-4
33.9	TFORMAT Command.....	33-4
33.10	Controlling Multiple Jobs with a Single Command.....	33-5
Chapter 34: Smart/MONITOR Browse and Edit.....		34-1
34.1	View and Change the Contents of File Buffers.....	34-1
34.2	Application program Data Storage	34-2
34.3	Browse and Edit commands.....	34-4
Chapter 35: Smart/PREDICTOR		35-1
35.1	Overview.....	35-1
35.2	How to Obtain Completion Time Predictions.....	35-1
35.3	Smart/PREDICTOR Displays.....	35-2

Chapter 40:	Smart/RESTART Support for Assembler Language Applications	40-1
40.1	An Assembler Language Main Module	40-2
40.2	An Assembler Language Sub-Module	40-7
40.3	Assembly / Link Edit JCL.....	40-10
40.4	Execution JCL.....	40-11
Chapter 41:	Checkpointing Media Alternatives	41-1
Overview		41-2
41.1	Checkpointing to a VSAM LDS	41-3
41.1.1	JCL Considerations	41-4
41.2	Checkpointing to a BDAM File	41-5
41.2.1	JCL Considerations	41-5
41.2.2	Checkpointing to a BDAM File -- Initial Run.....	41-6
41.2.3	Checkpointing to a BDAM File -- Restart Run.....	41-6
41.3	Checkpointing to DB2	41-7
41.3.1	RAINPUT Parameters for DB2 Checkpointing	41-7
41.3.2	Checkpointing to a DB2 Table -- Initial Run	41-8
41.3.3	Checkpointing to a DB2 Table -- Restart Run	41-8
41.4	Checkpointing to TAPE	41-9
41.4.1	RAINPUT Parameters for Tape Checkpointing.....	41-9
41.4.2	Checkpointing to Tape - Initial Run.....	41-10
41.4.3	Checkpointing to Tape -- Restart Run.....	41-11
41.5	Tying It All Together	41-12
Chapter 42:	Automatic Restart Facilities	42-1
42.1	Automatic Restart Overview.....	42-1
42.2	Automatic Restart Operation	42-2
42.2.1	Restrictions.....	42-3
42.3	Interactive Restart	42-3
42.4	Requesting Automatic Restart	42-3
42.4.1	ABEND_TRAP parameter.....	42-4
42.4.2	SQL_TRAP parameter	42-4
42.4.3	AUTO_RECOVER command	42-4

Chapter 43:	Abend Recovery Facilities	43-1
43.1	Abend Recovery Overview	43-1
43.2	Defining Eligible Abend Codes	43-2
43.3	Selecting Abend Recovery Options	43-2
43.4	Saving Images of Skipped or Replaced Records	43-2
43.5	Using Smart/MONITOR During Abend Recovery	43-3
43.6	Conducting an Interactive Abend Recovery Session via Smart/MONITOR	43-3
Chapter 44:	Externally Defined Checkpoints	44-1
44.1	External Checkpoint Overview	44-1
44.2	The Input Driver File processing model	44-3
44.2.1	Processing Model attributes	44-3
44.2.2	Application Processing Scenario.....	44-3
44.3	Unit-of-Work Definition.....	44-4
44.3.1	Units-of-Work defined by record count	44-4
44.3.2	Units-of-Work defined by record content	44-4
44.4	Application Feedback	44-4
44.5	External Checkpointing Example	44-5
44.5.1	Driver file contents.....	44-7
44.5.2	JCL for the Initial and Restart Runs	44-8
44.6	The Output Driver File processing model.....	44-9
Chapter 50:	Advanced Smart/RESTART Topics.....	50-1
50.1	Identifying Multiple Data Areas to Smart/RESTART	50-1
50.1.1	Explicit CALLs to the INITIALIZE Service	50-1
50.1.2	Explicit CALLs to the INITIALIZE Service – Format II	50-4
50.1.3	EXEC SRS SAVE statements	50-5
50.1.4	Smart/PRECOMPILER STORAGE and STORAGE-RANGE parameters....	50-6
50.2	Defining EXTERNAL Storage to Smart/RESTART.....	50-7
50.3	Point-in-Time Recovery with Smart/RESTART	50-8
50.4	SRSOFF Dual Environment Facility	50-9
50.5	Risk of Inconsistent Data with Language Environment (LE)	50-10
50.6	Spanned Record Support.....	50-11
50.7	The RMI parameter.....	50-11

50.7.1	Sync Data in DB2.....	50-12
50.7.2	DB2, WebSphere MQ and RRS.....	50-13
50.7.3	Synchronization Data Maintained in WebSphere MQ.....	50-14
50.7.4	RMI Summary.....	50-15
Chapter 51: Restart Points.....		51-1
51.1	Introduction.....	51-1
51.2	Restart Points	51-3
51.3	RESTART_POINT Syntax	51-5
51.4	Restart Point START	51-6
51.5	Restarting at the Main Processing Loop	51-7
51.6	Restart Point EXPLICIT	51-11
51.7	Restart Point COMMIT	51-12
51.8	Restart Point READ.....	51-14
51.9	Advantages of RESTART_POINT(EXPLICIT) vs. COMMIT and READ	51-14
51.10	Object Transparency and Restart Points	51-15
51.11	Mismatched Specifications	51-16
51.12	Distinguishing Between an Initial Run and a Restart Run.....	51-17
Chapter 52: Debugging Procedures		52-1
52.1.1	Job Message LOG	52-3
52.1.2	The Smart/RESTART message LOG.....	52-3
52.1.2.1	Error Messages	52-3
52.1.2.2	Warning Messages.....	52-4
52.1.2.3	Informational Messages.....	52-4
52.1.3	Smart/RRSAF Message LOG	52-5
52.1.3.1	Connect to DB2	52-6
52.1.3.2	Disconnect from DB2.....	52-6
52.1.4	Smart/RESTART formatted DUMP	52-6
52.2	Smart/RESTART Inputs and Setup	52-8
52.2.1	RAINPUT control file.....	52-8
52.2.2	Run-time options block.....	52-8
	Shared profile	52-9
52.2.3	DB2 Plan name and Target subsystem.....	52-10

52.2.4	Include the Smart/RESTART Package	52-10
52.2.5	Check job setup.....	52-10
52.2.6	Run on a Licensed z/OS System	52-10
52.2.7	Error Prone Practices and Procedures	52-10
52.2.8	Check for Correct DB2 BIND ENABLE Options	52-11
52.3	Gathering additional Diagnostics.....	52-11
52.3.1	MSG_DISPLAY(DEBUG).....	52-11
52.3.2	Trace Entry and Exit from 'registered' Modules	52-11
52.3.3	Function Trace	52-12
52.3.4	Check Operating shift Definitions.....	52-12
52.3.5	Check SQL trap definitions.....	52-13
52.3.6	SQL and RRSF tracing.....	52-13
52.3.7	Debugging with Smart/MONITOR.....	52-14
52.3.8	SRSUTIL and debugging	52-14
52.4	Operations with repositionable files.....	52-14
 Chapter 60: Smart / LOGGER for VSAM.....		60-1
60.1	Introduction.....	60-1
60.1.1	Key features of Smart/LOGGER	60-1
60.1.2	Key benefits	60-2
60.1.3	Product synergy.....	60-2
60.2	Smart/LOGGER processing overview.....	60-2
60.2.1	Initialization process	60-2
60.2.2	The Logging process	60-3
60.2.3	Checkpoint process	60-4
60.2.4	Backout process	60-4
60.3	Batch Backout Utility (SRLJBU)	60-5
60.4	Smart/LOGGER Installation Verification Procedure (IVP)	60-7
60.4.1	SRS8CV overview	60-7
60.4.2	Detailed Description of SRS8CV Processing.....	60-8
60.4.3	SRS8CV verification of a successful run.	60-9
60.5	Smart/LOGGER Customization	60-10
60.5.1	Table of System Defaults (SRL\$TSD)......	60-11
60.5.2	Logger Linear Dataset (LDS)......	60-13

60.5.2.1	LDS DASD space calculation.	60-13
60.5.2.2	Manual definition of a VSAM LDS	60-13
60.5.2.3	Dynamic creation of the VSAM LDS.....	60-14
60.6	Messages and codes	60-15

**Appendix A: Smart/RESTART Interoperation with
Other Products..... A-1**

Appendix B: Smart/RESTART Data Structures B-1

B.1	Assembler Language DSECTs.....	B-2
B.2	COBOL Copy Members	B -6
B.3	C Language Header Files.....	B -9
B.4	PL/1 Include Members.....	B -13

**Appendix C: Smart/RESTART's DB2 Checkpoint
Synchronization Table C-1**

C.1	Deleting Rows from the Checkpoint Synchronization Table.....	C-4
-----	--	-----

Section A

What's New in

Smart/RESTART

What's New in Smart/RESTART V12.10

- Smart/RESTART fully supports the following IBM(R) products:
 - o Enterprise COBOL 6.1.0, IBM Program Product 5655-EC6.
 - o Enterprise PL/I 5.1.0, IBM Program Product 5655-PL5.
 - o DB2 Version 12, IBM Program Product 5650-DB2.
 - o IMS Version 14, IBM Program Product 5751-CS3.
- Smart/RESTART can issue external checkpoints based on output file activity. See the Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description of the COMMIT_FILE directive (P116245A).
- Smart/PRECOMPILER now optionally generates PL/I code which references the PL/I builtin functions PLIRETC and PLIRETV rather than the PL/I RETCODE variable. See the Smart/RESTART User Guide (Publication SJS-004) for a detailed description of the Smart/PRECOMPILER PLI_RETCODE parameter (P116308A).
- Release 12.10.A1 incorporates corrective maintenance for the following PMRs:

PMR ID	Program Object	Description
P116143A	SRSRDB2	Smart/RESTART may suffer an S0C4 ABEND during auto-RESTART while running under z/OS RRS when the user application is erroneously linked-edited with the non-RRS WebMQ stub named

		CSQBSTUB. The ABEND takes place in program DCRARECO (+A62) of module SRS\$R31.
P116047A P116054A	SRSMTTERM	Smart/RESTART no longer references the IBM LE Condition Information Block during normal termination.
P11533A	DCRACKP3	Smart/RESTART may suffer an S0C4 ABEND in program DCRACKP3 (+1B6) of module SRS\$R31 while trying to commit a repositionable VSAM file's attributes to a checkpoint dataset.
P115233A P115317A	SRSACGL SRSCRCLS SRSCRFET SRSCRGCN	Smart/RESTART may fail to obtain a DB2 cursor name upon a SQL FETCH or SQL CLOSE request. The error is indicated by the Smart/RESTART error message SRSQ003E.
P115225A	DCRAIRWV	Smart/RESTART may fail to indicate the end-of-data condition for VSAM repositionable files.

What's New in Smart/RESTART V11.11

- Smart/RESTART fully supports Enterprise COBOL 5.2.0, IBM Program Product 5655-W32.
- The new PACEPERCENT command modifies the currently active checkpoint pacing criteria as specified by the CKPT_FREQ, CKPT_PACE and/or COMMIT_TIME parameters. See the Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description of the PACEPERCENT parameter.
- The REPOSITION_PDS_INPUT parameter specifies whether Smart/RESTART should reposition input files allocated to a member of a partitioned data set. Should such a file be included in the job step's repositioning list (as described by the REPOSITION parameter in the Smart/RESTART Reference Manual - Publication SJS-005) either explicitly or by default, then the REPOSITION_PDS_INPUT option specified (ON or OFF) governs whether or not Smart/RESTART will actually reposition the PDS input member at execution time.

What's New in Smart/RESTART V11.1

Smart/RESTART fully supports the following IBM(R) products:

- o Enterprise COBOL 5.1.0 Program Product 5655-W32. See Note below.

- o Enterprise PL/I for z/OS V4.R2.M0 Program Product 5655-W67.
- o DB2 Version 11 Program Product 5615-DB2.
- o IMS Version 11 Program Product 5635-A02.

NOTE: Smart/RESTART Object Transparency mode of operations does not support programs compiled with Enterprise COBOL 5.1.0 compiler.

- The new Smart/RESTART directive GLOBAL_OPEN_LIST specifies whether Smart/RESTART should track the OPEN and CLOSE activities of the non-repositionable files. See the Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description of this parameter.
- The new JOB_STEP_AND_PGMNAME setting of the AUTO_JOBID parameter directs Smart/RESTART to use the concatenation of job, jobstep and program names as the Smart/RESTART JOBID string.

E.g., AUTO_JOBID(JOB_STEP_AND_PGMNAME).

See the Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description of this parameter.

- The new JOB_STEP_AND_TASKPGM setting of the AUTO_JOBID parameter directs Smart/RESTART to use the concatenation of job, jobstep and task program names as the Smart/RESTART JOBID string.

E.g., AUTO_JOBID(JOB_STEP_AND_TASKPGM).

See the Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description of this parameter.

- The new Smart/RESTART parameter PACE_STATUS_CODE identifies a two character feedback literal with which the SRS7ARC program indicates a logical checkpoint request. See the Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description of this parameter.
- The new REPOSITION_REFRESH_AUTO parameter specifies whether Smart/RESTART should refresh repositioning matrix of any file listed within the REPOSITION_REFRESH parameter only if any of the following attributes of that file are changed since the last successful checkpoint: the last known volume serial number, the volume sequence number of the last known volume serial number, the number of allocated volumes, the dataset name, or the dataset device type (i.e., DASD or tape). See the Smart/RESTART Reference Manual (Publication SJS-005) for the detailed descriptions of this parameter.
- The new REPOSITION_VSAM_UNFIT parameter defines how Smart/RESTART should respond if an application attempts to open a VSAM file that is identified by a REPOSITION specification, but the DDname refers to a VSAM file that is ineligible for repositioning. This applies to ineligible VSAM files whose DD names appear in a REPOSITION specification with the ONLY keyword, as well as to ineligible VSAM files whose DDnames are omitted from a REPOSITION specification with

the EXCEPT keyword. See the Smart/RESTART Reference Manual (Publication SJS-005) for the detailed descriptions of this parameter.

- The new Smart/RESTART directive VERIFY_BLOCK specifies whether Smart/RESTART should signal a severe error when it detects a change in the size of the last committed block of a repositionable file. Alternatively, Smart/RESTART can issue a warning when it detects a change in size and continue restart processing. See the Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description of this parameter.
- Users of Smart/RESTART object transparency can now optionally direct Smart/RESTART to save and restore only part of their COBOL program's WORKING-STORAGE section. Users can identify "vital" WORKING-STORAGE (to be saved at checkpoint time and restored at restart time) via eyecatchers. The new VITAL_STORAGE_BEGIN(eyecatcher1) and VITAL_STORAGE_END(eyecatcher2) parameters are introduced to identify these eyecatchers to Smart/RESTART. See the Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description of these parameters.
- The RECOVER, REPO_STYLE, REPOSITION and REPOSITION_REFRESH parameters support the use of standard SQL wildcard characters in the DDname list. The underscore character "_" is a placeholder which matches any one character while the percent sign character "%" matches any number of characters. See the Smart/RESTART Reference Manual (Publication SJS-005) for the detailed descriptions of these parameters.
- The JOBID parameter accepts blanks within the Smart/RESTART JOBID specification, e.g. JOBID("This is my Job ID").
- Smart/RESTART can now reposition files whose concatenation references a mix of DASD and TAPE data sets.
- The maximum number of physical and logical checkpoints that can be displayed in the SRST052I message is increased from 9,999,999 to 999,999,999.
- Smart/RESTART can pace external checkpoints based on file activity. The COMMIT_FILE_PACING directive is introduced to support this functionality. See the Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description of this directive.
- Smart/RESTART can terminate restartable applications earlier based on time of day (TOD) or elapsed time. The EARLY_TERM_TOD and EARLY_TERM_ELAPSED directives are introduced to support this functionality. See the Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description of this directive.
- Smart/RESTART performance can now be improved for DB2 applications that use dynamic SQL, but do not issue dynamic SQL COMMIT and ROLLBACK requests. The new parameter SCAN_SQL_EXECUTE governs whether or not Smart/RRSAF, the DB2 attachment facility from RAI, should scan all dynamic SQL EXECUTE constructs for COMMIT and ROLLBACK requests. See the Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description.

- The default value for the SHORT_BLOCKS parameter is changed from NO to YES. See the Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description.

Smart/RESTART Recovery Services Updates

- The Smart/RESTART recovery service can now optionally close all repositionable files that are open when an ABEND occurs. This makes sure that any outstanding I/O buffer(s) are written out. The new ABEND_CLOSE_FILES directive specifies whether or not the Smart/RESTART recovery service should proceed with these CLOSEs. See the Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description of this parameter.
- The Smart/RESTART recovery service now allows IBM LE to handle any program interrupt and exception in which LE is interested. The new ESTAE_NOTIFY_IBMLE parameter is introduced to notify LE of any exception. See the Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description of the ESTAE_NOTIFY_IBMLE parameter.
- Smart/RESTART can now optionally adjust a jobstep completion code for the ABENDs triggered due to SQL_TRAP specifications. The unique jobstep completion code, for deliberate ABENDs triggered by various SQL failures, will provide simplified manual and automated operational responses. The new ABEND_DELTA_SQL_TRAP parameter is introduced to adjust the value of the default Smart/RESTART user ABEND code of 2005, should abnormal termination be caused by the 'ERROR' strategy of the SQL_TRAP parameter. See the Smart/RESTART Reference Manual (Publication SJS-005) for detailed descriptions of the SQL_TRAP and ABEND_DELTA_SQL_TRAP parameters.
- Smart/RESTART automatic restart performance can now be significantly improved for those applications that can tolerate residual system enqueues from the previous run. The new parameter AUTO_RECOVER_DEQ is introduced to govern whether or not the Automatic RESTART manager should scan for and release outstanding system enqueues. See the Smart/RESTART automatic restart performance can now be significantly Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description.
- The Smart/RESTART formatted dump processor now generates a summary report that includes a count of the following Smart/RESTART services executed: INITIALIZE, CHECKPOINT, TERMINATE, ALLOCATE, OPEN, CLOSE, READ and WRITE. Also, the report includes the elapsed, TCB and SRB times spent while executing these services. This report externalizes data collected by the Smart/RESTART MONITOR service. The parameter SRS_MONITOR governs whether or not the MONITOR service is running. The RAI-SNAP service, the DUMPDD and SRS_DUMP parameters govern the Smart/RESTART formatted dump processor. See the Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description.

Smart/RESTART Utilities Updates

The new Smart/RESTART Delete Utility (program SRSUDEL) provides a means to delete selected Smart/RESTART checkpoint datasets and their corresponding DB2 rows in a single jobstep. The SRSJUDEL member of the DCACNTL library contains JCL with which to execute the utility. See comments within the body of the SRSJUDEL member for detailed descriptions of utility parameters. See the Smart/RESTART User Guide (Publication SJS-004) for the detailed description of the utility itself.

The COPY function of SRSUTIL is now obsolete and superceded by the 2-step IEBGENER job described in Section 4.5 of the Smart/RESTART User Guide (Publication SJS-004). The member SRSJUCPY of the DCACNTL library contains this 2-step JCL.

Smart/RESTART API Services Updates

A restartable application can now force an initial run whether or not RESTART information exists for the current jobstep. The application can force an initial run by setting the status field of the Function Communications Area (FCA) to the value 'ERAS' – prior to calling the INITIALIZE service. The Smart/RESTART sample library, the dataset whose low level qualifier is DCASAMP, contains members which map the FCA in four (4) host languages as follows:

DCASAMP member	Host	FCA status
SRS@FCA	Assembler	FCASTAT
SRSHFCA	C or C++	fca_status
SRSCFCA	COBOL	FCA-STATUS
SRSPFCA	PL/I	FCA_STATUS

The Smart/RESTART checkpoint service now updates the new FCA checkpoint ID field with the ID of the most current Smart/RESTART checkpoint as follows:

DCASAMP member	Host	FCA checkpoint
SRS@FCA	Assembler	FCACHKID
SRSHFCA	C or C++	fca_ckpt_id
SRSCFCA	COBOL	FCA-CKPT-ID

SRSPFCA	PL/I	FCA_CKPT_ID
---------	------	-------------

The Smart/RESTART RAI-ENVCHECK service now presets FCA-CKPT-ID to the value of the current cross-run, physical checkpoint ID.

Smart/RESTART's API I/O services can now obtain a target file's record length from JCL or VTOC information associated with the file. This optional processing occurs during implicit OPEN when the first file I/O request is RAI-READ or RAI-WRITE. The new FCSIORL field is introduced to support this new functionality. In addition, the Smart/RESTART OPEN service now sets the new FCSORECL field with the file record length thus obtained. The Smart/RESTART sample library, the dataset whose low level qualifier is DCASAMP, contains members which map the FCS in four (4) host languages as follows:

DCASAMP member	Host	FCS field names	Output
SRS@FCS	Assembler	FCSIORL	FCSORECL
SRSHFCS	C or C++	fcs_implicit_open_recl	fcs_open_recl
SRSCFCS	COBOL	FCS-IMPLICIT-OPEN-RECL	FCS-OPEN-RECL
SRSPFCS	PL/I	FCS_IMPLICIT_OPEN_RECL	FCS_OPEN_RECL

The Smart/RESTART call level interface I/O services now support "DD DUMMY" files.

New Smart/RESTART Exit Routines

The new Smart/RESTART Job ID exit enables users to control the Smart/RESTART Job ID associated with an application run. The new parameter JOBID_EXIT is introduced to identify the name of the job ID user exit load module to Smart/RESTART. See the Smart/RESTART Reference Manual (publication SJS-005) for a detailed description of this parameter. The SRSXJBID member of the DCASAMP library contains a sample job ID exit, as an assembler CSECT. The SRS@JIX member of the DCAMACS library contains a macro with which to map the job ID exit communications area. Finally, the SRSEJBID member of the DCASAMP library contains JCL with which to assemble and link edit the SRSXJBID load module.

The new Smart/RESTART message exit enables users to override standard Smart/RESTART message routing. For each message, the exit can optionally (1) direct Smart/RESTART to terminate a restartable jobstep; (2) suppress message output; (3) write the message to the operator; (4) translate the message text to uppercase; (5) or change the Smart/RESTART MSG_DISPLAY setting for the current message.

The new parameter MESSAGE_EXIT is introduced to identify the name of the message exit load module to Smart/RESTART. See the Smart/RESTART Reference Manual (publication SJS-005) for a detailed description of this parameter. The SRSXMSG member of the DCASAMP library contains a sample message exit, as an assembler CSECT. The SRS@MXA member of the DCAMACS library contains a macro with

which to map the message exit communications area. Finally, the SRSBXMSG member of the DCASAMP library contains JCL with which to assemble and link edit the SRSXMSG load module.

Smart/RESTART IMS Features

Smart/RESTART can now update DB2 synchronization row with no explicit RAI-TERMINATE call under IMS environment.

Smart/RESTART can now automatically exclude all internal IMS VSAM clusters from repositioning processing.

Smart/RESTART can trigger an external checkpoint based on IMS PC activity. The COMMIT_PCB directive is introduced to support this functionality. See the Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description of this directive.

Smart/RESTART interoperates with IMS extended restart and symbolic checkpoint functionality.

The new Smart/RESTART directive IMS_BKO_NO specifies whether Smart/RESTART should signal a severe error when it detects the IMS DLI BKO parameter is set to NO. Alternatively, Smart/RESTART can issue a warning when it detects DLI BKO=NO, and continue application processing. See the Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description of this parameter.

What's New in Smart/MONITOR V11.1

The following Smart/MONITOR commands do not require for a target Smart/RESTART job to establish cross memory environment: PRO (profile), SUS (suspend) and RES (resume).

The new Resource Access Definitions Facility (RADF) is provided by Smart/RESTART to permit more granular access to resources controlled by Smart/MONITOR. See the Smart Jobstream Series Installation and Customization Guide (Publication SJS-001) for a detailed description of RADF.

The new ARC command is introduced to archive Smart/RESTART job entries within the Smart/MONITOR ISPF dialog. See Chapters 30, 31 and 33 of the Smart/RESTART User Guide (Publication SJS-004) for details about this new Smart/MONITOR command.

Smart/MONITOR now routes the CANCEL, FORCE and QUIESCE commands to the z/OS system where the target Smart/RESTART job is running. There is no longer any need to establish cross memory communications between Smart/MONITOR and the target Smart/RESTART job in order to issue any of these three (3) commands. These commands can be issued by authorized users only. The Smart/MONITOR user must have access to the RADF (Resource Access Definitions Facility) profile associated with the

corresponding command. See Chapter 9 of the Smart Jobstream Series Installation and Customization Guide (publication SJS-001) for a detailed description of the Smart/MONITOR Resource Access Definitions Facility. Also, see chapter 30 of the Smart/RESTART User Guide (Publication SJS-004) for detailed descriptions of the Smart/MONITOR CANCEL, FORCE and QUIESCE commands.

Smart/RESTART Interoperability With Other IBM and ISV Products

LE enabled Smart/RESTART applications written in COBOL, PL/I and Assembler can now be debugged with the IBM Debug Tool without restriction. An enhancement developed collaboratively with IBM enables task level SVC screening to be shared between Smart/RESTART and the Debug Tool. Appendix I of the Smart Jobstream Series Installation Guide describes the Debug Tool configuration options and runtime parameters required for the two products to interoperate when Debug Tool activates SVC screening. Note that the normal operation of the Debug Tool makes no use of SVC screening and requires no special setup to interoperate with Smart/RESTART. The ISPF EDIT macro SJSMEDIT is enhanced to identify the dataset name of the IBM Debug Tool load library. The Smart/RESTART sample library (the dataset whose low level qualifier is DCASAMP,) contains a sample COBOL program along with build and execute jobs to verify interoperability between Smart/RESTART and the IBM Debug Tool when the latter activates SVC screening. The interoperability members of the DCASAMP library are as follows:

DCASAMP member	Description
SRS8CIDT	Sample COBOL program
SRSBCIDT	Build job contains JCL with which to prepare SRS8CIDT
SRSRCIDT	Run job contains JCL with which to execute SRS8CIDT

Smart/RESTART can now optionally connect to and disconnect from WebSphereMQ manager even when Smart/RESTART synchronization data is not kept in MQ. The new FORCE_CONNECT_TO_QMGR parameter is introduced to direct Smart/RESTART to establish such a WebSphereMQ connection. See the Smart/RESTART Reference Manual (Publication SJS-005) for the detailed description of this parameter.

Smart/RESTART fully interoperates with DINO VELOCI-Raptor¹.

Smart/RESTART can now optionally invoke Macro 4 DumpMaster² for z/OS SNAP requests to collect ABEND diagnostics and write them to a DumpMaster Dump data set. Specify the SNAP Aid(ON) directive to enable this option. See the Smart/RESTART Reference Manual (Publication SJS-005) for a detailed description of this directive.

1. VELOCI-Raptor is trademark of DINO software.

2. DumpMaster is trademark of Macro 4.

Chapter 1

Smart/RESTART

Concepts and Facilities

This chapter introduces **Smart/RESTART** and provides an overview of its capabilities. The main objective in restart is to resume execution after a failure, rather than start the application over from the beginning. In this way, any completed work performed during a previous run need not be discarded. Smart/RESTART lets your application resume from a point of consistency (near the point of failure) after abends, recompiles, even system IPLs. Since Smart/RESTART handles the *mechanics* of checkpoint/restart, developers can concentrate on defining the *logical* unit-of-work that successful restart requires.

1.1 Background and context

There's less and less time to tolerate failures in batch production as organizations progress towards 'twenty-four by seven' operation and sharing data among multiple, concurrently active processes becomes routine. Meanwhile, the on-line portion of the day steadily encroaches upon a shrinking batch window.

Many organizations run precautionary backups, *just in case* shared files, tables, databases and message queues have to be restored. Should a job fail or be interrupted, (due to hardware errors, software errors, invalid input data, etc.) the typical recovery procedure is to **restore** these shared resources to their initial state and **rerun** the application from the beginning. The problem is compounded by the time consuming backout of changes which often take longer than the original processing.

Moreover, in a shared data environment, restoring these resources to an initial state risks losing the updates made by concurrent processes and effectively precludes such a practice. In such cases, checkpoint / restart capability becomes a *prerequisite* for concurrency.

1.1.1 Application Restart

In order to resolve these problems and conserve the precious batch window, a reliable means is needed to **restart** a failing application rather than **rerun** it from the beginning. The basic objective in restart is to resume execution from a point of consistency near the point of failure, rather than start the application over from the beginning. In this way, any completed work performed during a previous run need not be discarded.

To illustrate, consider an application that abends after reading 720 records from an input file containing 1000 records. Without restart logic, the program must be rerun from record number one. All the work performed during the initial run is lost and must be reprocessed. Suppose, however, the same application takes a checkpoint after each 100 records it processes. Should this 'restart enabled' version of the application abend on record 720, it can resume execution from record 701 (near the point of failure on record 720). In this case, nearly all the work performed during the initial run is retained. *Only the processing associated with records 701 through 720, (the work performed after the last successful checkpoint), is lost and must be reprocessed.*

Unfortunately, implementing such a restart capability considerably complicates application development. By some estimates, restart logic can *double* an application's size and significantly impair one's ability to comprehend and maintain it. Moreover, in the absence of a generic solution, each application must make its *own* provisions for restart.

1.1.2 Checkpoints and Units-of-work

In order for an application to restart successfully, (i.e. retain the useful work it completed in a previous run), it must take checkpoints at unit-of-work boundaries. A unit-of-work is the processing performed by an application from one point of consistency to another. A point of consistency reflects a state in which logically related resources are in accordance with one another. For example, a DB2 application establishes a point of logical consistency by issuing an EXEC SQL COMMIT request. IMS and WebSphere MQ provide their own checkpoint and rollback services to maintain data integrity by signaling the success or failure of a unit-of-work. In addition, RRS (the Resource Recovery Services component of the operating system) provides commit and backout services that any RRS compliant resource manager can utilize.

NOTE: This publication uses the term **resource manager** to refer to an authorized subsystem like DB2, IMS or WebSphere MQ. The term **protected resources** refers to the resources (such as DB2 tables, IMS databases and/or MQ queues) whose access is controlled by a resource manager.

However, to successfully restart an application, a more comprehensive unit-of-work is required -- one whose scope keeps **all** processing synchronized. Not just changes to DB2, IMS, MQ and RRS compliant resources -- but changes to working storage contents, VSAM updates and sequential file and cursor position as well. Such a **restartable-unit-of-work** represents a point of consistency from which processing can successfully resume upon restart. This restartable unit-of-work must operate in an atomic fashion in which all processing succeeds or fails as a unit. Consider what would happen if they did not succeed or fail *together*. A banking application, for example, might transfer funds between accounts A and B. Logically, the scope of this unit-of-work spans two actions. First, funds are *deducted* from account A. Next, funds are *added* to account B. Only after both updates complete successfully is a new point of consistency established.

Suppose, however, the application immediately made permanent every one of its changes -- instead of waiting until all changes within the unit-of-work were complete. In other words, the application takes a checkpoint immediately after subtracting funds from the first account. Suppose further that an abend occurs before the funds are added to the second account. This would leave the bank with inconsistent data -- and irate customers.

Clearly, checkpoints should only be issued after all logically related application processing and resource manager updates complete successfully. Conversely, if there is a failure in *any* of the steps comprising the unit-of-work, then application storage, sequential file position and 'protected' resources must be restored to the consistent state that prevailed at the beginning of the unit-of-work. In other words, all changes should be backed out as if they never took place.

1.2 What does Smart/RESTART do?

Smart/RESTART delivers such a restartable unit-of-work, in a way that's simple to use, and with minimal impact on performance. Smart/RESTART takes an execution-time 'snapshot' of your application so there's no need for complicated restart logic or cumbersome operational procedures. There's often no need to make changes to application source code. Smart/RESTART automatically

- saves application storage at checkpoint time and restores it at restart time;
- repositions sequentially accessed files during restart;
- guarantees that working storage contents as well as sequential file and DB2 cursor position remain 'in synch' with the unit-of-work managed by DB2, IMS and/or WebSphere MQ. (Smart/RESTART can be regarded as the resource manager for application storage, sequential files and VSAM datasets.)
- informs the application whether an initial run or restart run is in progress.

Since Smart/RESTART handles the **mechanics** of checkpoint/restart, developers can concentrate on defining the **logical** unit-of-work that successful restart requires. Chapter 2 discusses **logical** unit-of-work definition in detail.

1.2.1 Benefits of Smart/RESTART

Smart/RESTART users realize significant operational benefits along with simplified development and maintenance. Smart/RESTART minimizes the impact of batch job failures to keep you on schedule. Your nightly batch cycle becomes more predictable, reliable and secure -- and takes less time. The enterprise as a whole benefits from shortened recovery times and improved service levels.

By ensuring that processing always proceeds in a forward direction, Smart/RESTART eliminates the need to rerun abended jobs from scratch. In domino fashion, Smart/RESTART further reduces the need to back up shared files, DB2 tables, IMS databases and MQ queues -- to the extent precautionary copies are made to allow job restart. Further, Smart/RESTART

- eliminates redundant processing;
- predicts job completion time and issues 'behind schedule' warnings so operators can take remedial action;
- reduces deadlocks, timeouts and 'resource unavailable' conditions as a cause of job failure. Smart/RESTART detects such conditions and can automatically retry the failed unit-of-work. (The Advanced Features chapters in Part V of this manual describes Smart/RESTART's automatic recovery and retry facilities in detail).
- enhances job scheduling products that provide step restart capability (such as CA-11 from Computer Associates). Smart/RESTART enables jobs to resume from a checkpoint *within* a jobstep -- something a job scheduler alone cannot do.

Developers using Smart/RESTART benefit from not having to implement restart logic separately in each application. Smart/RESTART supports applications written in COBOL, PL/I, C/C++ and Assembler languages and supports all RRS compliant resource managers (such as DB2, IMS and WebSphere MQ). This makes it easy to adopt an enterprise-wide restart methodology.

1.3 Smart/RESTART Architecture and Components

Smart/RESTART is comprised of the following main components:

Smart/PRECOMPILER

Smart/PRECOMPILER can completely automate the restart enablement process. At program preparation time, Smart/PRECOMPILER adds the data structures and procedural code that enable restartable operation. Smart/PRECOMPILER also *translates* any EXEC SRS statements embedded in your application into compilable source statements. Lastly, Smart/PRECOMPILER conducts additional syntax, validity and consistency checks that help reduce errors and speed development.

Repositionable Sequential Access Method -- RSAM

Smart/RESTART provides the Repositionable Sequential Access Method (RSAM) as one of its standard facilities. I/O to files whose position must be reestablished at restart time can be performed with the I/O verbs native to the language in which your application is coded. For example, COBOL developers can code standard COBOL SELECT and FD statements and use native OPEN, CLOSE, READ and WRITE verbs to handle sequential file I/O. RSAM utilizes optimized, parallel I/O techniques that outperform native COBOL sequential file access.

DB2 Attachment Facilities

Smart/RRSAF (Recoverable Resource Services Attachment Facility) lets you run your DB2 batch applications using regular JCL, so they *look* and *behave* like regular OS jobs. For example, jobstep completion codes accurately reflect the outcome of processing. Moreover, SMF accounting is recorded under the application program name rather than being lumped together under the name of the TSO Terminal Monitor Program.

Smart/RRSAF supports both RRSAF and CAF as DB2 attachments -- without the need for user code to manage an explicit DB2 connection. Support for RRSAF enables your applications to make changes to DB2, IMS, MQ and other RRS compliant resources within a single unit of recovery.

Smart/RRSAF can be used with DB2 stored procedures as well as with DB2 batch applications. Smart/RRSAF implicitly determines whether a DB2 stored procedure is executing in a DB2 or WLM established stored procedure address space and automatically uses the correct DB2 attachment.

Together, Smart/RRSAF and Smart/RESTART provide valuable monitoring, diagnostic and exception handling facilities that are not available with TSO/DSN (or other DB2 attachment facilities). Smart/RRSAF provides the data for Smart/RESTART's real-time displays of SQL activity and for the detail and summary reports you can analyze off-line. Smart/RRSAF is described in its own User Guide (Publication DCA-002).

Smart/LOGGER

Smart/LOGGER for VSAM automates application restart to reduce computer down time, shorten application restart times, and ensure data integrity for VSAM users. Smart/LOGGER for VSAM works with Smart/RESTART and Smart/RRSAF to provide automated restart capabilities for applications that access VSAM data as well as other RRS compliant resources. In addition, Smart/LOGGER for VSAM, by utilizing VSAM Record Level Sharing, further improves data availability by allowing a batch job to share VSAM data with other batch jobs and CICS regions.

Execution Component

The execution component provides Smart/RESTART run-time facilities such as its coordinated checkpoint and rollback services whose scope spans application processing and resource manager activity.

Smart Job Management

Smart/RESTART provides facilities that let you monitor and control your workload of restartable applications. Smart/MONITOR (described in Part IV) provides summaries of SQL requests, file I/O and CPU activity in real-time. It also lets you request a variety of print reports for off-line analysis. Smart/PREDICTOR (also described in Part IV) estimates job completion times. Its early warnings allow you to take remedial action to help meet production schedules.

1.4 Method of Operation

Smart/RESTART receives control whenever your application signals the completion of a unit-of-work. A checkpoint request indicates a unit-of-work was processed successfully while the issuance of a rollback request signals failure. Smart/RESTART receives control *explicitly* when you call its CHECKPOINT or ROLLBACK services. Smart/RESTART receives control *implicitly* when your application issues either EXEC SQL COMMIT or ROLLBACK requests or calls the SRRCMIT or SRRBACK subroutines that provide RRS commit and rollback services respectively.

Smart/RESTART records a 'snapshot' of your application at checkpoint-time that represents a logical point of consistency in processing. Checkpoint processing operates in an atomic fashion, wherein application processing and resource manager activity succeed or fail *as a unit*. This ensures that application storage and sequential file position remain synchronized with DB2, IMS and/or MQ resources.

If your application requests a rollback or an expected failure occurs, Smart/RESTART directs the coordinated "back out" of changes applied since the beginning of the unit-of-work. The Smart/RESTART ROLLBACK service restores working storage, sequential file position and VSAM updates along with changes to DB2, IMS, MQ and other RRS compliant resources to their most recent point of consistency.

1.4.1 What happens during Restart?

KEY POINT: *Restart occurs from a checkpoint, **not** the point of failure.*

At restart-time, Smart/RESTART restores working storage contents and sequential file and cursor position to their state at checkpoint time -- a state consistent with the 'protected' resources managed by RRS compliant resource managers such as DB2, WebSphere MQ, and IMS. This enables your application to resume processing with the unit-of-work that *follows* the units-of-work processed successfully on previous run(s).

First, Smart/RESTART restores the contents of working storage. Next, Smart/RESTART repositions each sequentially accessed file so that input and output resume from the checkpointed record position -- rather than from the beginning of the file. Lastly, Smart/RESTART returns control to your application with an indication of the restart environment. This allows your application to distinguish between an initial and restart run and perform different initialization processing if appropriate. For example, the initialization routine can display a message that identifies either an initial or restart run.

1.4.2 Checkpoint Pacing and Performance

You can vary the **checkpoint frequency** to change the number of *logical* CHECKPOINT (or SQL COMMIT or SRRCMIT) requests Smart/RESTART receives before triggering a *physical* checkpoint of application processing and resource manager activity.

The physical checkpoint frequency you choose represents a tradeoff between speed (infrequent checkpoints) and concurrency (frequent checkpoints). As the application designer, you must establish the boundaries of each unit-of-work (as described in Chapter 2) and determine a checkpointing frequency appropriate for your application.

NOTE: *For purposes of application restart, processing always resumes from the point of consistency associated with a **physical** checkpoint.*

1.5 When is Restart Enabled?

This section introduces the four modes of restart enablement supported by Smart/RESTART.

With **Smart/PRECOMPILER**, restart enablement occurs at *program preparation time*, without the need to reference Smart/RESTART within the application source code. RAI recommends use of Smart/PRECOMPILER whenever source code is available and program preparation is possible. Application development using Smart/PRECOMPILER is the focus of Chapter 2 which addresses the fundamentals of Smart/RESTART.

However, use of Smart/PRECOMPILER requires an additional jobstep in your program preparation JCL. Thus, Chapter 5 describes and illustrates how restart enablement can also be achieved through the use of COBOL copybooks or PL/I INCLUDE text. Some organizations may prefer this technique to the Precompiler Transparency approach illustrated in Chapter 2. RAI recommends both techniques with equal weight.

Next, Smart/RESTART provides a **call level interface** to its services, as an alternative to the automatic enablement provided by Smart/PRECOMPILER (along with its support for high level EXEC SRS requests as described in Chapter 10). Chapter 11 describes the Smart/RESTART Call Level Interface and those Smart/RESTART features (such as dynamic modification of checkpoint pacing) which are available only through the API.

Lastly, **Object Transparency** refers to the Smart/RESTART facility that endows existing DB2/COBOL applications with restart support -- without source code changes of any kind and without the need to recompile. This makes Object Transparency appropriate for software supplied in object code only format or when program preparation, for whatever reason, is not feasible. Object Transparency is an attractive alternative to the MVS CKPT macro. It has extremely modest overhead in comparison with the MVS CKPT macro, yet it lets you restart after recompiles or system IPLs. Chapter 12 describes Object Transparency in detail.

NOTE: *Applications composed of separately compiled or assembled source modules can employ more than one mode of Smart/RESTART support. For example, an application might be a composite of an precompiled main module and lower level routines that use the Smart/RESTART call level interface. In general, however, RAI recommends that only the main module be 'restart enabled', using any of the restart enablement techniques Smart/RESTART supports.*

1.6 Smart/RESTART Features

- Smart/RESTART applications can be resubmitted for restart using the same JCL used for the initial run. (Chapter 4)
- Authorized users can vary options like checkpoint pacing *during* a run -- based on factors such as elapsed time, number of records processed, time of day, heuristic code and/or operator request. This lets you adjust execution speed and resource contention in accordance with changing conditions -- without having to cancel a job. (Chapter 4)
- Smart/RESTART applications can be comprised of multiple, independently compiled modules. Moreover, there can be multiple load modules within a single restartable run unit. (Chapter 2)
- The Smart/RESTART QUIESCE command lets you gracefully cancel an application on a unit-of-work boundary. For example, you can quiesce a job running beyond its batch window and resume it at a more convenient time -- without wasted processing or time consuming backout. (Chapter 4)
- Checkpoints can be forced in applications without commit logic. The logical unit-of-work can even be defined **external** to the application. (Part V)
- Smart/RESTART supports all COBOL compilers and runtime libraries including OS/VS COBOL, VS COBOL II, COBOL for MVS and VM, COBOL for OS/390 and VM and Enterprise COBOL for z/OS and OS/390.
- Smart/RESTART fully supports multi-volume tape datasets. On restart, only the relevant tape volume need be mounted.
- Applications can be modified and recompiled between job failure and job restart.
- Smart/RESTART provides an automatic retry facility designed to reduce or eliminate timeouts, deadlocks and 'resource unavailable' conditions as a cause of background job failure. When Smart/RESTART detects a 'retry' condition, it restores the contents of application storage and sequential file position to their state as of the last successful checkpoint. Your application can then retry the failing unit-of-work -- without the need to manually resubmit the job. (Part V).

1.7 Customization

The Smart/RESTART architecture provides interfaces to installation written exit routines that let you customize and extend Smart/RESTART processing as needed. Exit interfaces are supplied for the following Smart/RESTART events:

- Initialization
- Checkpoint requests
- Termination
- Upon reading a record from a repositionable file
- When the Smart/ RESTART runtime profile is altered dynamically

For example, statistics and performance-monitoring packages can be turned on and off at program initialization and termination time. Alternatively, a checkpoint exit can decide whether to request a physical checkpoint. Lastly, a read exit can inspect and modify data and even request checkpoints based on record content. Appendix A of the Installation and Customization Guide for the Smart Jobstream Series describes how to develop exit routines for Smart/RESTART.

1.8 The Smart/RESTART Sample Library

Smart/RESTART is supplied with a sample library whose low level dataset qualifier is DCASAMP. The names of the sample members within the DCASAMP library adhere to the following standards to make it easier to discern their content:

```
SRS@ - Assembly language mapping macros
SRSC - COBOL copy members
SRSH - C header files
SRSP - PL/I members
SRSX - source code of sample exit routines
SRSB - JCL to Build (prepare) sample applications
SRSR - JCL to Run sample applications
SRSD - Document members
SRSG - GEN members (e.g. IMS DBDGEN and PSBGEN)
SRSJ - Jobstream JCL
SRS8 - Source for sample members (follows the DB2 convention)
SRS8A- Assembler source members
SRS8C- COBOL sample source members
SRS8H- C sample source members
SRS8P- PL/I sample source members
```

Developing Restartable Applications

This chapter presents **principles** of restartable application design. The examples in this chapter are COBOL and DB2, but the same principles apply to all the programming languages and resource managers that Smart/RESTART supports. Moreover, these principles apply both to new development as well as to retrofitting existing applications for restartable operation.

All applications must adhere to these design principles in order to restart successfully. Smart/RESTART automates the mechanics of checkpoint/restart by keeping application storage and sequential file position in synch with updates to DB2 tables, IMS databases MQ queues and/or other RRS compliant resources. What Smart/RESTART does **is necessary but not sufficient for successful restart**. Application (and developer) responsibilities for successful restart are summarized in Section 2.1. The remainder of Chapter 2 amplifies and illustrates this topic.

2.1 Summary of Application Responsibilities for Restart

The design and implementation of **every** restartable application must address the issues listed below. Once these issues are addressed, the remaining responsibility for checkpoint/restart can be delegated to Smart/RESTART.

1. You must divide application processing into units-of-work, the processing performed from one logical point of consistency to another. A unit-of-work is a discrete sequence of steps that modify one or more items of data. If the entire sequence of steps completes successfully, the changes can be made permanent. Otherwise, all changes must be backed out to the *previous* point of consistency. **The logical unit-of-work is the crucial concept in restartable application design.**

The essence of restartable design is to identify your application's unit-of-work and then request a checkpoint when it completes successfully. Alternatively, you should back out processing if it fails. Each checkpoint represents a 'point of consistency' from which processing can resume successfully upon restart.

NOTE: You should request checkpoints at unit-of-work boundaries even in the absence of Smart/RESTART. For example, the SQL COMMIT statement not only makes changes permanent, it permits DB2 to release locks as well as the log records it needs to back out changes performed during the unit-of-work. In the absence of COMMIT requests, all locks and log records are held until the application terminates (normally or abnormally).

2. You must design main line logic (as described in Section 2.3.1) such that your application **commences** or **resumes** execution at a unit-of-work boundary.
2. You must pre-allocate the files that need to be repositioned on restart. This lets you submit the *same JCL* for either an initial or restart run. (Chapter 4 discusses JCL for restartable applications in detail.)

2.2 Pseudo Code Example

Figure 2.1 presents a pseudo-coded example which performs sequential I/O and issues SQL requests within a unit-of-work loop. There is no reference to Smart/RESTART within the pseudo code. Smart/RESTART **intercepts** all I/O requests and SQL statements as well as calls to the RRS commit and rollback services SRRCMIT and SRRBACK. Thus, from the standpoint of the example, Smart/RESTART operates **transparently**.

NOTE: This transparent restart enablement is accomplished via a component called Smart/PRECOMPILER which requires an additional jobstep in your program preparation JCL. Chapter 5 describes and illustrates how restart enablement can also be achieved through the use of COBOL copybooks or PL/1 INCLUDE text. Some organizations may prefer this technique to the Precompiler Transparency approach discussed in this Chapter. RAI recommends both techniques with equal weight.

```

Do until End-of-File
  Do until unit-of-work complete
    READ input file                (1)
    update DB2 resources
    WRITE output record            (1)
  End
  If unit-of-work completes successfully
    EXEC SQL COMMIT or Call SRRCMIT (2)
  else
    EXEC SQL ROLLBACK or Call SRRBACK (3)
  Endif
End

```

where

- (1) Smart/RESTART's Repositionable Sequential Access Method (RSAM) **transparently** manages I/O to files whose position must be re-established at restart time. Thus, you can perform I/O using verbs native to the host language in which your program is coded. For example, COBOL programmers use OPEN, CLOSE, READ and WRITE verbs whereas Assembler Language developers can use GET and PUT macros.
- (2) The Smart/RRSAF component of Smart/RESTART gets control on every SQL request. When Smart/RRSAF receives an EXEC SQL COMMIT request, it calls the Smart/RESTART checkpoint service to save an execution-time 'snapshot' of application storage and sequential file position that is guaranteed to be in synch with the DB2 changes made permanent by SQL COMMIT.

NOTE: *This entire chapter assumes the use of Smart/RRSAF as the DB2 attachment facility. Chapter 22 discusses how to use attachment facilities other than Smart/RRSAF to access DB2.*

Alternatively, your application can call the RRS commit routine SRRCMIT to make permanent changes to *multiple* RRS compliant resources. Smart/RESTART gets control on the SRRCMIT call and invokes the Smart/RESTART checkpoint service to save an execution-time 'snapshot' of application storage and sequential file position that is guaranteed to be in synch with changes to RRS compliant resources made permanent by the SRRCMIT service.

- (3) When Smart/RRSAF receives an EXEC SQL ROLLBACK request, it calls the Smart/RESTART ROLLBACK service to back out changes to application storage, sequential file position and DB2 in a coordinated fashion -- to a prior point of consistency.

Alternatively, your application can call the RRS rollback service SRRBACK to backout changes to *multiple* RRS compliant resources. Smart/RESTART gets control on the SRRBACK call and invokes the Smart/RESTART ROLLBACK service to back out changes to application storage, sequential file position and RRS compliant resources in a coordinated fashion -- to a prior point of consistency.

Figure 2.1 *Pseudo Coded Example*

2.3 Sample DB2/COBOL Application

This section presents a complete COBOL/DB2 program that illustrates how restartable applications can be developed *without any reference to Smart/RESTART whatsoever*. Instead, Smart/PRECOMPILER adds data structures and procedural code at program preparation time that restart enable your application.

NOTE: *Smart/RESTART provides several sample applications in source and load module format. The source modules reside in the Smart/RESTART library whose low level qualifier is **DCASAMP**. The JCL to prepare and run some of these applications also reside in the DCASAMP library. Chapter 1.8 describes the naming conventions for members of the DCASAMP library. In contrast, the JCL to run the IVP applications are tailored as part of installation in the Smart/RESTART library whose low level qualifier is **DCAFILE**.*

The bold, italic numbers within parentheses correspond to the numbered annotations which follow the program. Many of these points are discussed in greater detail in subsequent sections of this chapter.

```

ID DIVISION.
PROGRAM-ID.      EXAMPLE.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INPUT-FILE ASSIGN TO INFILE
        FILE STATUS IS INPUT-FILE-STATUS.

DATA DIVISION.
FILE SECTION.

FD  INPUT-FILE
01  INPUT-RECORD                      PIC X(80).

WORKING-STORAGE SECTION.

01  VITAL-STORAGE.
    05  UOW-COUNT                      PIC S9(4) COMP VALUE 1.
    05  TRANSACTION-COUNT              PIC S9(4) COMP VALUE 0.

01  HOST-VARIABLES.
    05  HOST-VARIABLE-1                PIC S9(5) COMP.
    05  HOST-VARIABLE-2                PIC S9(5) COMP.
    05  HOST-VARIABLE-3                PIC S9(5) COMP.

EXEC SQL INCLUDE SQLCA END-EXEC.

01  NON-CRITICAL-FIELDS.
    05  PROCESSING-FLAG                PIC X(1) VALUE SPACE.
    88  SUCCESSFUL-PROCESSING          VALUES ARE ' ' 'E'
    88  END-OF-FILE                    VALUE 'E'.
    88  IO-ERROR                       VALUES ARE 'I'.
    88  PROCESSING-ERROR               VALUES ARE 'I' 'P'.
    05  INPUT-FILE-STATUS              PIC 9(02) VALUE ZEROS.

PROCEDURE DIVISION.
    PERFORM A100-INITIALIZE THRU A100-EXIT.
    PERFORM B100-UNIT-OF-WORK THRU B100-EXIT          <=== (1)
        UNTIL END-OF-FILE
        OR    PROCESSING-ERROR.
    DISPLAY 'EXAMPLE - PROCESSING COMPLETED'.
    GOBACK.

A100-INITIALIZE.
    IF RETURN-CODE = +2001                      <=== (2)
        DISPLAY 'THIS IS A RESTART RUN OF THE EXAMPLE PROGRAM'
    ELSE
        DISPLAY 'THIS IS THE INITIAL RUN OF THE EXAMPLE PROGRAM'.
    OPEN INPUT INPUT-FILE.                      <=== (3)
A100-EXIT.
    EXIT.

B100-UNIT-OF-WORK.                                <=== (4)
    MOVE 0 TO READ-COUNT.
    PERFORM B200-PROCESS-TRANSACTION THRU B200-EXIT
        UNTIL TRANSACTION-COUNT > 10
        OR    END-OF-FILE OR PROCESSING-ERROR.

    IF SUCCESSFUL-PROCESSING
        EXEC SQL COMMIT WORK END-EXEC          <=== (5)
        ADD 1 TO UOW-COUNT
    ELSE
        EXEC SQL ROLLBACK WORK END-EXEC.        <=== (6)
B100-EXIT.
    EXIT.

```

```

B200-PROCESS-TRANSACTION.
  READ INPUT-FILE INTO HOST-VARIABLES
  AT END MOVE 'E' TO PROCESSING-FLAG
  GO TO B200-EXIT.
  IF INPUT-FILE-STATUS NOT = '00' THEN
    PERFORM IO-ERROR-ROUTINE
    GO TO B200-EXIT.

  ADD 1 TO TRANSACTION-COUNT.
  EXEC SQL INSERT INTO SOME.TABLE
    VALUES (:HOST-VARIABLE-1, :HOST-VARIABLE-2, :HOST-VARIABLE-3)
  END-EXEC.
  IF SQLCODE < 0 THEN ...
B200-EXIT.
EXIT.

```

where

- (1) The flow of control enters the main unit-of-work loop at the same point in either an initial run or a restart run. Thus, execution begins at a unit-of-work boundary whether processing *commences* on an initial run or *resumes* during a restart run. (Discussed in Section 2.3.1.)
- (2) During a restart run, Smart/RESTART sets the COBOL special register RETURN-CODE to the value +2001 so your application can distinguish between an initial and restart run.
- (3) OPEN works normally during an initial run. However, during a restart run, Smart/RESTART sets sequential files to their position at checkpoint time as part of OPEN processing. (Discussed in Section 2.3.2.1.)
- (4) The routine which performs a unit-of-work is discussed in Section 2.3.4.
- (5) The application issues an EXEC SQL COMMIT request when the unit-of-work completes successfully. The Smart/RRSAF component of Smart/RESTART intercepts this SQL COMMIT request and calls the Smart/RESTART CHECKPOINT service. This Smart/RESTART facility saves an execution-time 'snapshot' of application storage and sequential file position that is guaranteed to be in synch with the DB2 changes made permanent by EXEC SQL COMMIT. Because the scope of this restartable unit-of-work spans both application processing and resource manager activity, it represents a point of consistency from which processing can resume successfully upon restart.
- (6) The application requests a ROLLBACK if the unit-of-work fails to complete successfully. The Smart/RRSAF component of Smart/RESTART intercepts the SQL ROLLBACK request and calls Smart/RESTART's more comprehensive ROLLBACK service. The ROLLBACK facility implemented by Smart/RESTART backs out changes to application storage, sequential file position and DB2 to their last checkpointed state. This represents a point of consistency from which processing can successfully resume.

Figure 2.2 **Sample DB2/COBOL Application Program**

2.3.1 Mainline Logic

Figure 2.3 illustrates the mainline logic of a restartable application. The flow of control is the same for both initial and restart runs. Control flows naturally into unit-of-work processing during an initial run. **Re-entry** into the unit-of-work loop also occurs naturally in the event of a restart. RAI strongly recommends your main modules implement such a unit-of-work loop. It should be entered directly, once initialization processing is performed.

```
PROCEDURE DIVISION.  
  PERFORM A100-INITIALIZE THRU A100-EXIT.  
  PERFORM B100-UNIT-OF-WORK THRU B100-EXIT      <=== unit-of-work Loop  
    UNTIL END-OF-FILE  
    OR   PROCESSING-ERROR.  
  DISPLAY 'EXAMPLE - PROCESSING COMPLETED'.  
  GOBACK.
```

Figure 2.3 Same Mainline Logic for both Initial and Restart Runs

2.3.2 Initialization Routines

This section discusses the requirements of initialization routines for restartable applications.

2.3.2.1 Open / Reopen files

Smart/RESTART **does not** automatically reopen repositionable files during a restart run. Therefore, your application must explicitly OPEN files for both initial and restart runs as illustrated in Figure 2.4.

RSAM (Smart/RESTART's Repositionable Sequential Access Method) intercepts OPEN (as well as all other I/O requests) for each sequentially accessed file. OPEN works normally during an initial run. During restart, however, OPEN repositions sequentially accessed files to their position at checkpoint time to allow processing to resume from the proper record.

```
A100-INITIALIZE.  
  . . .  
  OPEN INPUT INPUT-FILE.      <=== (1)  
A100-EXIT.  
  EXIT.
```

where

(1) OPEN sequential files for both initial and restart runs

Figure 2.4 OPEN sequential files

2.3.2.2 Do not initialize areas restored by Smart/RESTART

This sections deals with *what not to do*. The main restriction in making changes to the source code of a main module that is restart enabled with Smart/RESTART involves the initialization of storage that will be saved at checkpoint time and restored at restart time. We provide several pseudo code fragments below (followed by commentary) to describe and illustrate what works and what does not work. The first fragment illustrates a storage initialization technique that is not conducive to successful restart while the next two examples illustrate techniques consistent with restart.

To illustrate what is at issue, suppose a restartable batch application maintains a counter like the data item below to reflect how many units-of-work it has successfully processed. The value of this counter should be saved at checkpoint time and restored at restart time.

```
DCL
    Commit_Count  BINARY FIXED (31)
```

Suppose further the application processes 30 units-of-work (issues 30 commits) and then abends *while processing* unit-of-work number 31. Smart/RESTART saved a value of 30 for **Commit_Count** as of the last successful checkpoint. When the failed application is restarted, Smart/RESTART restores the **Commit_Count** data item with its saved value of 30. Conceptually, Smart/RESTART performs storage restoration before the first executable statement of the restarted program (although this default can be overridden via the RESTART-POINT directive).

```
Restart Example 1 does not work -
Not consistent with successful restart
```

```
DCL
    Commit_Count  BINARY FIXED (31)
```

```
Do Initialization      /* at this point, Commit_Count = 30 */
    (re)open sequential files, VSAM datasets and DB2 cursors
    Commit_count = 0    /* overlays the restored value of 30 */
End

Do while (more input)
    Process 1 - n transactions within a Unit_of_Work
    If Unit_of_Work processed successfully then
        Commit_Count = Commit_Count + 1
        COMMIT
    Endif
End
```

In this example of *what not to do* above, the **Commit_Count** variable (whose value was restored on restart to 30) is re-initialized on restart to 0 -- via an unconditional procedural statement. This *overlays* the restored value of 30 that reflects how many units-of-work were successfully processed before the initial run failed and *interferes* with the restarted application's correct and accurate processing.

The next two fragments illustrate storage initialization techniques that resolve this problem so as to be conducive to successful restart. In example 2, the initialization of the **Commit_Count** variable via a procedural statement occurs only during the initial run and is *bypassed* at restart time. Note that one of Smart/RESTART's fundamental capabilities is to enable the application to distinguish between an initial and restart run.

Restart Example 2 works - Consistent with successful restart

DCL

```
Commit_Count  BINARY FIXED (31)
```

```
Do Initialization  /* On restart, Commit_Count = 30 */
    (re)open sequential files, VSAM datasets and DB2 cursors
    If initial-run then /* Now sensitive to restart vrs initial run */
        Commit_Count = 0 /* restored value of 30 undisturbed */
    Endif
End
```

In example 3, the **Commit_Count** variable is initialized to 0 non procedurally through a DECLARE statement (or an analogous COBOL VALUE clause). As such, there is no procedural statement to overlay the 'restart time' value of 30 restored by Smart/RESTART.

Restart Example 3 works - Consistent with successful restart

DCL

```
Commit_Count  BINARY FIXED (31) INIT(0)
```

```
Do Initialization  /* On restart, Commit_Count = 30 */
    (re)open sequential files, VSAM datasets and DB2 cursors
End
```

In Figure 2.5 below, for example, the fields grouped under the structure VITAL-STORAGE are among the storage areas Smart/RESTART saves at checkpoint-time and restores at restart-time. The minimal initialization routine labeled A100-INITIALIZE does not alter this storage. Thus, the unit-of-work count restored into the field labeled UOW-COUNT is valid when the application's flow of control re-enters its main unit-of-work loop (as described in Section 2.3.3).

```

01  VITAL-STORAGE.
    05  UOW-COUNT          PIC S9(4) COMP VALUE 0.
    05  READ-COUNT        PIC S9(4) COMP VALUE 0.

. . .

A100-INITIALIZE.
    IF RETURN-CODE = +2001
        DISPLAY 'THIS IS A RESTART RUN OF THE EXAMPLE PROGRAM'
    ELSE
        DISPLAY 'THIS IS THE INITIAL RUN OF THE EXAMPLE PROGRAM'
    OPEN INPUT INPUT-FILE.
A100-EXIT.
    EXIT.

```

Figure 2.5 *Initialization Logic*

As an example of what not to do, suppose this application completes 10 units-of-work before failing. Smart/RESTART restores UOW-COUNT to the value 10 so the restart run can resume execution with a counter that indicates that 10 units-of-work are already complete. Your initialization routine must not include a statement like:

```
MOVE 0 to UOW-COUNT.
```

In summary, you must not use procedural statements to initialize the same fields that Smart/RESTART will restore on a restart run.

2.3.3 The Main Loop

Figure 2.6 illustrates **iteration** as the fundamental aspect of any restartable process. The loop defined in the main-line logic below drives the processing of discrete **units-of-work**. This loop is entered during an initial run and **re-entered** on any restart run(s).

```
PROCEDURE DIVISION.  
  PERFORM A100-INITIALIZE THRU A100-EXIT.  
  PERFORM B100-UNIT-OF-WORK THRU B100-EXIT      <=== (1)  
    UNTIL END-OF-FILE  
    OR    PROCESSING-ERROR.  
  DISPLAY 'EXAMPLE - PROCESSING COMPLETED'.  
  GOBACK.
```

where

(1) defines the **Logical unit-of-work loop**

Figure 2.6 **Main-line Logic**

Figure 2.7 illustrates the EXAMPLE program's unit-of-work routine. This routine encompasses the processing which occurs from one logical point of consistency to another. Your design responsibility is to identify where processing reaches a point of logical consistency within your own application and thereby establish the unit-of-work boundary. If all processing within the unit-of-work completes successfully, then changes are made permanent. Otherwise, all changes must be backed out to the previous point of consistency.

```
B100-UNIT-OF-WORK.  
  MOVE 0 TO READ-COUNT.  
  PERFORM B200-PROCESS-TRANSACTION THRU B200-EXIT  
    UNTIL TRANSACTION-COUNT > 10  
    OR    END-OF-FILE OR PROCESSING-ERROR.  
  
  IF SUCCESSFUL-PROCESSING  
    EXEC SQL COMMIT WORK END-EXEC      <=== (1)  
    ADD 1 TO UOW-COUNT  
  ELSE  
    EXEC SQL ROLLBACK WORK END-EXEC.    <=== (2)  
B100-EXIT.  
EXIT.
```

where

(1) Make changes permanent if the unit-of-work completes successfully

(2) Back out changes if the unit-of-work fails to complete successfully

Figure 2.7 **Unit-of-work routine**

2.3.3.1 Checkpoint Pacing and Performance

The *checkpoint frequency* can be adjusted during execution by varying the number of EXEC SQL COMMIT requests Smart/RESTART must receive before it physically issues a checkpoint of application processing and resource manager activity. The decision to issue a physical checkpoint can be based upon such factors as elapsed time, number of records processed, time of day, heuristic code, or explicit operator request.

The physical checkpoint frequency you choose represents a tradeoff between speed (infrequent checkpoints) and concurrency (frequent checkpoints). The CKPT_PACE command is described in more detail in Chapter 4 (which discusses program execution and restart) and within Chapter 11 of the Smart/RESTART Reference.

2.3.4 An Individual transaction

Figure 2.8 illustrates the processing that occurs during an individual transaction. The transaction processing routine reads a record from a sequential input file and then inserts it as a row into a DB2 table.

```
B200-PROCESS-TRANSACTION.  
  READ INPUT-FILE INTO HOST-VARIABLES  
    AT END MOVE 'E' TO PROCESSING-FLAG  
    GO TO B200-EXIT.  
  IF INPUT-FILE-STATUS NOT = '00' THEN  
    PERFORM IO-ERROR-ROUTINE  
    GO TO B200-EXIT.  
  
  ADD 1 TO TRANSACTION-COUNT.  
  EXEC SQL INSERT INTO SOME.TABLE  
    VALUES (:HOST-VARIABLE-1, :HOST-VARIABLE-2, :HOST-VARIABLE-3)  
  END-EXEC.  
  IF SQLCODE < 0 THEN ...                               <=== (1)  
B200-EXIT.  
  EXIT.
```

where

- (1) Smart/RRSAF sets SQLCODE and SQLSTATE values exactly like the DB2 attachment facilities supplied by IBM.

Figure 2.8 *An individual transaction within a unit-of-work*

2.4 SQL Cursor Processing

Although Smart/RESTART can automatically reposition DB2 cursors (via the REPOSITION_CSR directives described in Chapter 12 of the Smart/RESTART Reference), RAI recommends that DB2 applications **re-establish** cursor position during restart using the coding techniques described in this section.

This discussion applies to SQL cursors declared with and without the HOLD option. DB2 provides the WITH HOLD option to maintain cursor position across COMMIT requests. However, the WITH HOLD option cannot preserve cursor position across a restart because the DB2 thread is terminated when the application fails.

Restartable DB2 applications can declare a single, **self-repositioning** cursor that applies to both the initial run of an application as well as any restart run(s) that may be necessary. Such cursor declarations should include a WHERE clause that is parameterized by host variables. The values of these host variables at SQL OPEN time govern what answer set is produced. The host variables that drive the SELECT (i.e. parameterize the WHERE clause) should be among the fields saved at checkpoint time and restored at restart time. In the pseudo coded example in Figure 2.9, the WHERE clause is parameterized by the host variable named `primary-key`.

```
Ø1 Vital-storage.  
  Ø5 primary-key pic x(Ø5) value low-values.  
  
Declare c1 cursor for  
  select * from someones.keyed_table  
  where primary_key > :primary-key  
  
Open c1  
Do while SQLCODE = Ø  
  fetch c1 into :primary-key, ....  
End
```

Figure 2.9 Self Repositioning Cursor Declaration

In this example, the host variable `primary-key` is initialized to low-values during the initial run of the application so it produces the *entire* answer set. The application saves the value of `primary-key` at checkpoint time and restores it during restart. Since the cursor declaration references the value of `primary-key` in its WHERE clause, the answer set materialized at restart time will contain rows whose key values are greater than the restored `primary-key` value. Thus, the restart answer set represents the *unprocessed* rows of the original query result.

2.5 Repositioning Input Files

This section describes how Smart/RESTART repositions a sequential input file on restart. To illustrate, let's assume an application reads records **1** through **100** on the initial run and then issues a checkpoint. Suppose processing continues and the application **abends** after reading record **107**.

On restart, how *should* Smart/RESTART reposition the input file? Should the last record read at checkpoint time be the first record the application reads on restart (i.e. re-read record 100)? Or should the first record read on restart *follow* the last record read at checkpoint time (i.e. read the next record -- number 101)? The answer depends on how your application processes input.

Section 2.5.1 describes and illustrates a processing model in which input file position on restart is set at the last record read at checkpoint time. Section 2.5.2 discusses a different model in which input file position is set after the last record read at checkpoint time.

2.5.1 Control Break Processing Model -- Reposition AT

Figure 2.10 presents a pseudo coded example in which a 'control break' in the input record signals the completion of a unit-of-work. The application exits its loop when it detects a 'control break', *before* it fully processes the record which triggers the break. On restart, the application must re-read this record and fully process it as part of the next unit-of-work. To support this requirement, Smart/RESTART resets input file position at the last record read at checkpoint time.

The control break processing model is characterized by a single input file whose record contents or record counts govern when a unit-of-work is complete.

NOTE: *Your application can access other repositionable input and output files. However, they do not define the boundaries of the unit-of-work.*

Smart/RESTART provides the REPO_STYLE parameter as a means of governing input file repositioning during restart. The repositioning style appropriate for the 'control break' model described in this section is REPO_STYLE(AT).

Chapter 4 of this manual describes Program Execution and Restart in detail. It describes how to specify run-time parameters (such as REPO_STYLE) to Smart/RESTART. All the Smart/RESTART parameters, commands and callable services are discussed in the Smart/RESTART Reference.

```
Open Input-file                (1)
READ input file                (2)
Do while more input records
  Do until Control-Break      (3)
    process input record
    READ input file          (4)
  End
  If successful-processing    (5)
    Commit unit-of-work
  else
    Rollback unit-of-work
  Endif
End
```

where

- (1) On restart, OPEN repositions the input file **at** the last record read at checkpoint time. This is the 'control break' record which signaled the completion of a unit-of-work during the previous run.
- (2) During restart, the first READ returned by the priming READ **is the same as** the last record read at checkpoint time. The application can re-read the control break record and fully process it within a new unit-of-work.
- (3) Test for exit from the loop **after** a record is read but **before** the record is processed.
- (4) Read the next record at the bottom of the loop.
- (5) The application commits the unit-of-work before it fully processes the record which triggered the control break.

Figure 2.10 Control Break Processing Model -- Reposition AT

2.5.2 Record Processed Model - Reposition AFTER

The 'Record Processed' model described here assumes every record read within a unit-of-work is fully processed within that unit-of-work. The cursor driven processing illustrated in Figure 2.11 is an example of this processing model. The application exits its read/process loop **after** it fully processes each record it reads. To support this requirement, Smart/RESTART resets input file position **after** the last record read at checkpoint time.

The repositioning style appropriate for the 'Record Processed' model described in this section is REPO_STYLE(AFTER).

```
Open Input-file                (1)
Do until End-of-File
  Do until unit-of-work complete
    READ input file            (2)
    process input record
  End
  If successful-processing      (3)
    Commit unit-of-work
  else
    Rollback unit-of-work
End
```

where

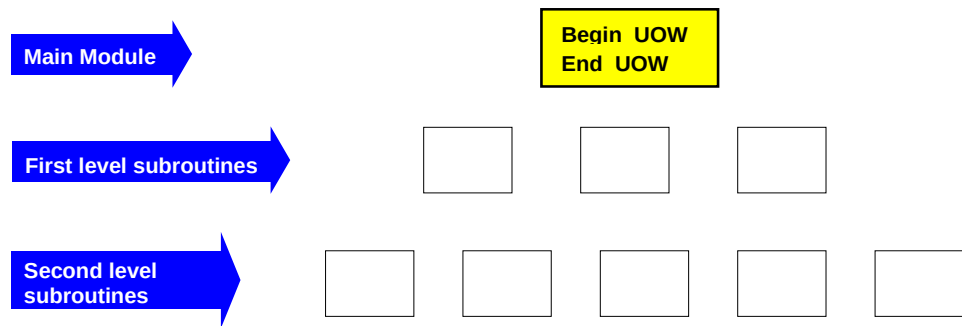
- (1) On restart, OPEN repositions the input file such that the first record returned by read is the record **after** the last record read at checkpoint time.
- (2) The first record read in the new unit-of-work **follows** the last record read in the successfully completed unit-of-work in the previous run.
- (3) The application commits the unit-of-work **after** it fully processes the current input record.

Figure 2.11 Record Processed Model -- Reposition AFTER

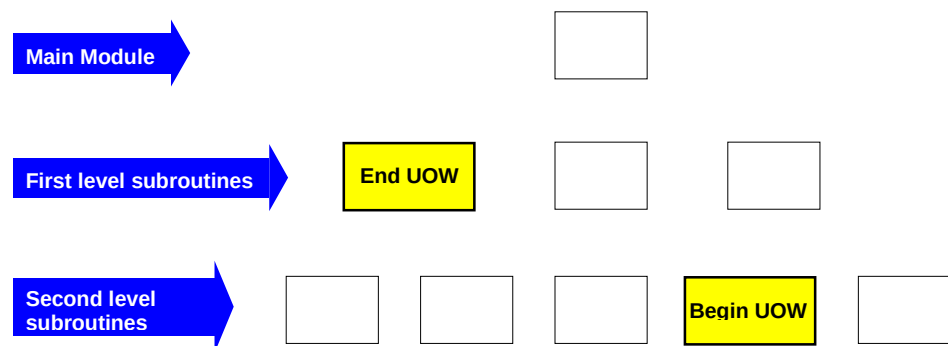
2.6 Applications Comprised of Multiple Modules

This section discusses restartable applications comprised of multiple modules. These can be written in any mix of COBOL, PL/1, C/C++ and Assembler Languages. You can link-edit several independently compiled and/or assembled modules into a single load module. Alternatively, your application can invoke its subroutines dynamically. COBOL and PL/1 modules can be preprocessed by Smart/PRECOMPILER whereas C/C++ and Assembler Language modules must use the Call Level Interface (described in Chapter 11) to request Smart/RESTART services.

RIGHT WAY to define the unit-of-work



WRONG WAY to define the unit-of-work



Multi-module applications are composed of separately compiled or assembled source modules. To restart enable a multi-module batch application, the *only* module that *should* be 'restart enabled' (via Smart/PRECOMPILER or another method) is the main one. Even an application comprised of hundreds of modules only needs its main module to be processed by Smart/PRECOMPILER. ***In general, sub programs need not and should not be precompiled or Smart/RESTART aware.*** This allows subroutines to be shared among applications running in CICS, IMS, TSO and/or batch environments. Only sub programs that define storage to be saved at checkpoint time and restored in the event of restart must be precompiled. Moreover, Relational Architects *does not recommend* that storage to be saved and restored be defined in sub programs.

In contrast, Relational Architects *strongly recommends* that you define the logical unit-of-work in the application's main module as illustrated in the 'RIGHT WAY' diagram below. This promotes clarity as to where the unit-of-work begins and ends, and allows developers to state with confidence that an application is logically as well as physically restartable (i.e. will restart successfully and produce correct results).

NOTE: *Smart/RESTART provides its checkpoint/restart services at the module level. An example of a module is a single COBOL program identified by its COBOL Program-ID. A module may also describe a routine written in C/C++, PL/1 or Assembler Language. In the following discussion, the term module refers only to application routines within a run unit. It does not refer to any compiler run-time library routines or operating system routines invoked as part of the run unit.*

Although Smart/RESTART allows substantial flexibility in coding, RAI recommends that your multi-module applications follow these guidelines:

- Use Smart/PRECOMPILER to preprocess just the main module in a multi-module application. Alternatively, use the COPY/INCLUDE technique described in Chapter 5 to restart enable the main module or call the Smart/RESTART INITIALIZE and TERMINATE services through the Call Level Interface described in Chapter 11. The main module ***must*** be restart enabled, even if it has no storage areas to save.
- Define the unit-of-work in the main module (the highest level routine) and recognize the success or failure of the unit-of-work in the main module as well. The actual checkpoint requests (for example EXEC SQL COMMIT in DB2 programs) however, can be issued from sub routines that are called from the main module. Such a structure also promotes clarity and maintainability.
- Preferably, define the storage areas to be saved and restored in the main module. The storage defined to Smart/RESTART in the main module can be passed to called subprograms.

Although Smart/RESTART lets you restart enable a subroutine one or more levels down from the main module, RAI does not recommend doing so. If you implement such a complicated structure, Smart/RESTART saves the storage areas defined to it in each module *active* at checkpoint time. During a restart run, Smart/RESTART restores a subroutine's storage the first time the module is called. In order to restart successfully, your application must re-invoke these subroutines in the same order they were called in the previous run.

2.7 General restrictions on Smart/RESTART usage

- COBOL applications can invoke an internal SORT so long as no checkpoint and rollback requests are issued within any sort input or output procedure.
- DB2 does not support the use of multiple DB2 attachment facilities (such as Smart/RRSAF and TSO attach) within the same run unit. This DB2 restriction also means Smart/RRSAF cannot toggle between RRSAF and CAF attachments within a single run.
- When multiple modules refer to EXTERNAL data, they actually refer to the same storage. Therefore, data structures defined with the EXTERNAL attribute should be saved and restored within a single module only. Otherwise, Smart/RESTART will save and restore these areas in each module in which they are defined. Chapter 50 discusses how to save and restore EXTERNAL data.

2.8 Restrictions on Smart/RESTART's Repositionable Sequential Access Method (RSAM)

- RSAM does not support files allocated to SYSIN, SYSOUT or DUMMY. Smart/RESTART returns either a warning or error message at OPEN time for such files, depending on the site default. In warning mode, standard access methods handle I/O to SYSIN, SYSOUT and DUMMY files. No repositioning is possible in the event of application failure and restart. Chapter 4 describes how to run an additional jobstep to circumvent this restriction.
- Smart/RESTART's Repositionable Sequential Access Method only supports files which specify QSAM locate and move mode. This includes all COBOL and PL/1 releases as well as Assembler Language modules which specify QSAM locate mode as a DCB MACRF operand.
- Repositionable I/O is not supported within the INPUT or OUTPUT procedures of a COBOL internal SORT.
- RSAM does not currently interoperate with CICS VSAM file sharing.
- RSAM does not support OPEN with either the EXTEND or I-O option.

Restartable Program Preparation

Program preparation refers to the various steps required to ready your applications for execution. Beyond compiling and linking your application, DB2 applications go through an optional precompile step and mandatory BIND step. Finally, Smart/RESTART has program preparation requirements of its own which are described in detail in this chapter.

The following program preparation steps in Figure 3.1 should be performed in sequence. Several steps are optional as denoted by the [dependent section] notation within brackets.

- Step 1 [DB2 dependent step]
Modules which contain embedded SQL statements can be preprocessed by the DB2 precompiler to translate these SQL requests into compilable source code. Alternatively, a compiler's SQL Coprocessor can translate embeded SQL statements.
[End DB2 dependent step]
- Step 2 Optionally, execute Smart/PRECOMPILER to add the data structures and procedural code that enables restart. Smart/PRECOMPILER also *translates* any EXEC SRS statements embedded in your application into compilable source statements. The original source module can contain EXEC SQL statements, EXEC SRS statements and/or Call Level Interface requests for Smart/RESTART services -- in any combination.

Alternatively, restart enablement can also be achieved through the use of COBOL COPY members or PL/I INCLUDE text as described and illustrated in Chapter 5. Some organizations may prefer this technique in lieu of using Smart/PRECOMPILER. RAI recommends either technique with equal weight.

- Step 3: Compile or assemble the source program into an object module using either the COBOL, PL/I or C/C++ compilers or the High Level Assembler.
- Step 4: Link Edit the application (with the z/OS BINDER) to produce an executable load module or program object.
- Step 5: [DB2 dependent step]
BIND applications which contain embedded SQL statements to produce a DB2 application plan.
- [End DB2 dependent step]

NOTE: *Restartable applications that access just sequential files and/or VSAM datasets do not require a DB2 application plan or the SRSDBRM package. In contrast, restartable applications which access RRS compliant resources (such as those maintained in DB2 tables or WebSphere MQ queues) require a synchronizing entity like the Smart/RESTART DB2 table. As such, they require either a DB2 application plan that includes the SRSDBRM package or a recoverable WebSphere MQ queue dedicated to Smart/RESTART use.*

3.1 Illustrative Program Preparation Jobstream

Figure 3.1 illustrates program preparation JCL for a COBOL/DB2 application that is restart enabled with Smart/PRECOMPILER. The numbers in the right hand margin (within parentheses) correspond to the numbered annotations which follow the example.

```

//JOBNAME JOB ....
//SRSPCOB PROC MEMBER=,HOST=IBMC0B,SOUT=H,UNIT=SYSDA,
//          RAIHLQ='?raihlq?',
//-----
//PC        EXEC PGM=DSNHPC,PARM='HOST(&HOST),SOURCE,APOST'          <=== (1)
//SYSTEM    DD  SYSOUT=&SOUT
//SYSPRINT  DD  SYSOUT=&SOUT
//SYSUT1    DD  SPACE=(800,(500,500),,,ROUND),UNIT=&UNIT
//SYSUT2    DD  SPACE=(800,(500,500),,,ROUND),UNIT=&UNIT
//SYSCIN    DD  DSN=&DSNHOUT,DISP=(MOD,PASS),UNIT=&UNIT,              <=== (2)
//          SPACE=(800,(500,500)),DCB=(LRECL=80,BLKSIZE=3120)
//DBRMLIB   DD  DSN=MY.DBRM.LIBRARY,DISP=SHR
//-----
//SRSPC     EXEC PGM=SRSPC,PARM='HOST(&HOST)', REGION=4096K          <=== (3)
//STEPLIB   DD  DISP=SHR,DSN=&RAIHLQ..DCALOAD
//SRSLIB    DD  DISP=SHR,DSN=&RAIHLQ..DCASAMP
//SRSIN     DD  DSN=&DSNHOUT,DISP=(MOD,PASS),UNIT=&UNIT,              <=== (4)
//SRSOUT    DD  DSN=&SRSOUT,DISP=(MOD,PASS),UNIT=&UNIT,              <=== (5)
//          SPACE=(800,(500,500)),DCB=(LRECL=80,BLKSIZE=3120)
//SRSPRINT  DD  SYSOUT=&SOUT
//RAINPUT   DD  DUMMY
//-----
//COB       EXEC PGM=IGYCRCTL,COND=(4,LT,PC),                        <=== (6)
//SYSIN     DD  DSN=&SRSOUT,DISP=(OLD,DELETE)                          <=== (7)
//SYSLIB    DD  DISP=SHR,DSN=&RAIHLQ..DCASAMP
//SYSLIN    DD  DSN=&OBJSET,DISP=(MOD,PASS),UNIT=&UNIT,
//          SPACE=(800,(500,500))
//SYSPRINT  DD  SYSOUT=*
//SYSUT1    DD  SPACE=(800,(500,500),,,ROUND),UNIT=&UNIT
//SYSUT2    DD  SPACE=(800,(500,500),,,ROUND),UNIT=&UNIT
//SYSUT3    DD  SPACE=(800,(500,500),,,ROUND),UNIT=&UNIT
//SYSUT4    DD  SPACE=(800,(500,500),,,ROUND),UNIT=&UNIT
//SYSUT5    DD  SPACE=(800,(500,500),,,ROUND),UNIT=&UNIT
//-----
//LKED      EXEC PGM=IEWL,COND=(4,LT),PARM=(LIST,XREF,REUS,REFR)     <=== (8)
//SYSLIB    DD  DISP=SHR,DSN=&RAIHLQ..DCANCAL
//SYSLMOD   DD  DISP=SHR,DSN=program.load.library
//SYSUT1    DD  DSN=&SYSUT1,UNIT=VIO,SPACE=(1024,(50,20))
//SYSPRINT  DD  SYSOUT=&SOUT,DCB=(RECFM=FB,LRECL=121,BLKSIZE=1210)
//SYSLIN    DD  DSN=&OBJSET,DISP=(OLD,DELETE)
//-----
//BIND      EXEC PGM=IKJEFT01                                         <=== (9)
//DBRMLIB   DD  DSN=MY.DBRM.LIBRARY,DISP=SHR
//SYSTSPRT  DD  SYSOUT=*
//SYSTSIN   DD  DUMMY
//          PEND

```

where

- (1) The DB2 Precompile step is described in Section 3.2.
- (2) Temporary dataset &DSNHOUT contains the modified source module produced by the DB2 precompiler.
- (3) The Smart/PRECOMPILER step is described in Section 3.3.
- (4) Temporary dataset &DSNHOUT becomes input to Smart/PRECOMPILER.
- (5) The modified source module produced by Smart/PRECOMPILER becomes input to the COBOL compiler in the next jobstep.
- (6) The COBOL compile step is described in Section 3.4.
- (7) Temporary dataset &SRSOUT contains output from Smart/PRECOMPILER which in turn becomes input to the COBOL compiler.
- (8) The Link Edit step is described in Section 3.5.
- (9) The DB2 BIND step is described in Section 3.6.

Figure 3.1 Restart Program Preparation Jobstream

Smart/PRECOMPILER runs *after* the DB2 precompiler but *before* the COBOL compile step. Installations that utilize Smart/RESTART can revise their catalogued procedures to include a Smart/PRECOMPILER step like the one illustrated in Section 3.3.1, or adapt the SRSPC procedure distributed in Smart/RESTART's DCASAMP dataset. The SRSPC procedure illustrates the use of Smart/PRECOMPILER within the context of a DB2 program preparation jobstream.

3.2 DB2 Precompile [DB2 dependent section]

The DB2 precompiler translates any EXEC SQL requests embedded in your program into compilable host language statements. In addition, the DB2 precompiler adds the data structures and procedural code necessary for your application to execute SQL statements.

NOTE: *If your compiler invokes the DB2 SQL Coprocessor, then the DB2 Precompiler jobstep which runs programs DSNHPC is not needed.*

3.3 Smart/PRECOMPILER

In a similar fashion, Smart/PRECOMPILER further modifies your COBOL and/or PL/I modules to enable them for restart. These modifications entail adding the data structures and procedural code with which Smart/RESTART keeps your application's working storage contents and sequential file position synchronized with changes to RRS compliant resources manager like DB2, IMS and/or MQ. Smart/PRECOMPILER also translates any embedded EXEC SRS statements it encounters into compilable source statements.

This time honored tradition of taking *simple input* and producing more *complicated input* (to a subsequent compile step) allows you to maintain your application at a source level. Should you change it, you can simply re-prepare the original source code. It is seldom (if ever) necessary to capture the translated source program and maintain the version modified by either the DB2 precompiler or Smart/PRECOMPILER.

NOTE: *Smart/PRECOMPILER can run before or without the DB2 precompiler and fully supports the DB2 SQL Coprocessors associated with various compilers. However, Smart/PRECOMPILER does not support C/C++ or Assembler languages. Source modules written in C/C++ or Assembler language can only request Smart/RESTART services through the Call Level Interface described in Chapter 11.*

Source modules which make no reference whatsoever to Smart/RESTART are termed **source transparent**. This allows the same source code to run with or without Smart/RESTART, depending on how it is prepared. For example, a source transparent module can run restartably by including Smart/PRECOMPILER as a step in its program preparation job. The same source module can run without Smart/RESTART simply by re-preparing it without Smart/PRECOMPILER.

3.3.1 JCL for Smart/PRECOMPILER

You can revise your program preparation procedures to include a Smart/PRECOMPILER step like the one illustrated in Figure 3.2.

NOTE: *The numbers in the right hand margin (within parentheses) correspond to the numbered annotations which follow the JCL. Several of these annotations in turn are discussed in further detail in the subsections which follow.*

```
//SRSPC      EXEC  PGM=SRSPC,PARM='HOST(IBMCOB) '           (1)
//STEPLIB    DD    DSN=DCAvrms.DCALOAD,DISP=SHR             (2)
//SRSIN      DD    DSN=&DSNHOUT,DISP=(OLD,DELETE)            (3)
//SRSLIB     DD    DSN=DCAvrms.DCASAMP,DISP=SHR              (4)
//SRSOUT     DD    DSN=&SRSOUT,DISP=(NEW,PASS)                (5)
//SRSPRINT   DD    SYSOUT=*                                  (6)
//RAINPUT    DD                                           (7)
RM(DB2)
/*
```

where

- (1) Run Smart/PRECOMPILER and pass it run-time parameters.
- (2) STEPLIB defines the Smart/RESTART system load library which contains the SRSPC load module.
- (3) The SRSIN file defines input to Smart/PRECOMPILER (described in Section 3.3.2).
- (4) The SRSLIB DD statement identifies the Smart/PRECOMPILER copy library, the dataset whose low level qualifier is DCASAMP.
- (5) The SRSOUT DD statement defines the precompiled source program to which Smart/PRECOMPILER has added restart enabling code. Temporary dataset &SRSOUT is passed to a subsequent compile step (described in Section 3.3.3).
- (6) The SRSPRINT file defines the Smart/PRECOMPILER list dataset (described in Section 3.3.3).
- (7) The RAINPUT DD statement defines additional run-time options for Smart/PRECOMPILER to be merged with the parameter string specified on the EXEC statement (described in Section 3.3.4).

Figure 3.2 *Smart/PRECOMPILER Jobstep*

3.3.2 Input to Smart/PRECOMPILER

The SRSIN DD statement defines the input to Smart/PRECOMPILER. It should be defined with DCB attributes RECFM F or FB with an LRECL of 80. Two cases of input to Smart/PRECOMPILER are illustrated below:

Case 1: The original source module is read directly by Smart/PRECOMPILER

```
//SRSIN DD DSN=your.source.program,DISP=SHR
```

Case 2: Smart/PRECOMPILER reads the preprocessed source module produced by the DB2 precompiler

```
//SRSIN DD DSN=&DSNHOUT,DISP=(OLD,PASS)
```

NOTE: All source statements should be coded within the margins specified by the Smart/PRECOMPILER MARGINS option and should correspond to any MARGINS option specified to the DB2 precompiler.

3.3.3 Output from Smart/PRECOMPILER

Smart/PRECOMPILER writes a modified source program to the file SRSOUT. This restart enabled source module becomes input to the compile step. The dataset referenced by the SRSOUT DD statement should be defined with DCB attributes RECFM F or FB, with an LRECL of 80.

In addition, Smart/PRECOMPILER writes several listings to file SRSRINT. You can allocate SRSRINT to SYSOUT or specify a dataset with the following DCB attributes: LRECL=133 and RECFM=F or RECFM=FB. The listings produced by Smart/PRECOMPILER include the following:

- the default and explicit options in effect for this execution of Smart/PRECOMPILER (if OPTIONS is itself in effect)
- a listing of the precompiled source program (if the SOURCE option is in effect.)
- diagnostic messages (if any) which indicate the precompiler assigned line numbers of any statements in error.

3.3.4 Smart/PRECOMPILER Run-time Options

Smart/PRECOMPILER is controlled by options specified (or defaulted) when the precompiler is invoked. These options specify how input should be processed and how output is to be presented. Smart/PRECOMPILER options can be specified via the PARM field of the JCL EXEC statement, through an RAINPUT DD statement, or both as illustrated in the following examples:

```
//SRSPC EXEC PGM=SRSPC,PARM='HOST(IBMCOB)'  
//RAINPUT DD DSN=smart.precompiler.options,DISP=SHR
```

Smart/PRECOMPILER merges options from both these sources together (if present) to develop the set of run-time options. If duplicate or conflicting options are specified, only the last specified value takes effect. The RAINPUT dataset should be defined with DCB attributes RECFM F or FB with a LRECL of 80. Alternatively, you can code precompiler options *instream* as in the following example:

```
//RAINPUT DD *  
HOST(IBMCOB)  
* this is a comment  
//*
```

Any options supplied through the RAINPUT file must be coded between columns 1 and 72. Smart/PRECOMPILER treats any line beginning with an asterisk (*) as a comment. To continue a precompiler option onto another line, place a hyphen '-' (the precompiler's continuation character) at the end of the line and continue the option on the next line.

A set of default values for Smart/PRECOMPILER options are defined when Smart/RESTART is installed. Thereafter, you need not specify the defaulted values except to override them. For example, if most of your restartable applications are written in Enterprise COBOL for z/OS and OS/390, the precompiler's HOST option can be defaulted to COB2 (a value valid for any COBOL compiler subsequent to OS/VS COBOL). The Smart Jobstream Series Installation Guide describes the administrative dialog with which to define default options for Smart/PRECOMPILER. Check with your Smart/RESTART product administrator for the list of precompiler defaults in effect at your site.

Figure 3.3 enumerates Smart/PRECOMPILER options along with acceptable abbreviations for these options. The vertical bar | separates mutually exclusive option values.

Of particular note are Smart/PRECOMPILER's STORAGE and STORAGE_RANGE options. Both enable you to identify the names of one or more data structures. If a source module defines these data structure(s), then Smart/PRECOMPILER generates code so that Smart/RESTART saves the contents of these structure(s) at checkpoint time and restores them at restart time.

For example, your site can adopt a convention that all application storage to be saved at checkpoint time and restored at restart time should be subsumed within the COBOL group item named VITAL-STORAGE. Your site can then define VITAL-STORAGE as the default value for Smart/PRECOMPILER's STORAGE option. Thereafter, Smart/PRECOMPILER will generate code to save and restore the contents of VITAL-STORAGE in any preprocessed source module in which VITAL-STORAGE is defined.

CAF(SMARTRRS|TSO|IMS|OTHER)

Identifies the DB2 attachment package used to connect to DB2 and execute SQL statements. The default is SMARTRRS.

FLAG(I|W|E|S)

Directs Smart/PRECOMPILER to suppress diagnostic messages below the specified severity level. The flag codes are I-Informational, W-Warning, E-Error and S-Severe error. They correspond to severity codes 0, 4, 8, and 12, respectively.

HOST(PLI|IBMCOB)

Specifies the host language in which the source module is coded. The HOST option also sets the default margins for the specified source language (see the MARGINS option below).

IO_CONVERT (ALL|NONE|ONLY|EXCEPT(ddname_list))

Directs Smart/PRECOMPILER to *convert* native file I/O statements (such as COBOL's OPEN, CLOSE, READ and WRITE verbs) to calls to Smart/RESTART I/O services. ***RAI does not recommend the use of this feature except as directed by RAI technical support.*** Instead, RAI recommends you use the *REPOSITION* run-time parameter (described in the Smart/RESTART Reference). The REPOSITION parameter supports repositionable I/O without changes to the application's original I/O statements.

When the IO_CONVERT feature of Smart/PRECOMPILER is specified, it identifies the file(s) whose native I/O statements should be converted. The ALL keyword signals that native I/O statements for all application files should be converted to calls to the Smart/RESTART I/O services. Conversely, NONE means no native I/O statements for any file should be converted to calls to the Smart/RESTART I/O services. The ONLY keyword directs Smart/PRECOMPILER to convert native I/O statements for only those files whose names appear in the ddname list. Lastly, the EXCEPT keyword means all native I/O statements for all application files, except those which appear in the ddname list, should be converted to calls to the Smart/RESTART I/O services.

MARGINS(m,n)

MAR

Specifies what part of each source record contains host language statements. The first value (m) denotes the beginning column while the second value (n) represents the last column in which host language statements will appear. Default values for beginning and ending columns depend on the HOST language option specified, as follows:

PLI:	2,72	
IBMCOB:	8,72	(no other values are permitted)

NOOPTIONS

NOOPTN

Directs Smart/PRECOMPILER to suppress the precompiler options listing.

NOSOURCE

NOS

Directs Smart/PRECOMPILER to suppress the precompiler source listing.

OPTIONS

OPTN

Directs Smart/PRECOMPILER to print the list of explicit and default precompiler options.

PLI_RETCODE(YES|NO)

PLI_RETCODE(YES)

Directs Smart/PRECOMPILER to insert references to PL/I variable named RETCODE to indicate a RESTART run, an INITIAL run, and Smart/RESTART initialization and termination failures:

```
CALL RAIPI(RAI_INITIALIZE,FCA,. . .);
IF FCA_STATUS = 'REST' & FCA_RETURN_CODE = 0 THEN
    RETCODE = 2001;
ELSE
    RETCODE = FCA_RETURN_CODE;
. . .
FCA_PGM_RETURN_CODE = RETCODE;
CALL RAIPI(RAI_TERMINATE,FCA);
IF FCA_RETURN_CODE = 0 THEN
    RETCODE = FCA_PGM_RETURN_CODE;
```

Starting with IBM Enterprise PL/I Version 5.1.0, references to RETCODE variable may lead to the following compiler error:

```
IBM1373I E      nnn.n      Variable RETCODE is implicitly declared.
```

PLI_RETCODE(NO)

Directs Smart/PRECOMPILER to insert references to PL/I PLIRETC() and PLIRETV() built-in functions to indicate a RESTART run, an INITIAL run, and Smart/RESTART initialization and termination failures:

```
CALL RAIAPI(RAI_INITIALIZ,FCA,. . .);
IF FCA_STATUS = 'REST' & FCA_RETURN_CODE = 0 THEN
    CALL PLIRETC(2001);
ELSE
    CALL PLIRETC(FCA_RETURN_CODE);
. . .
FCA_PGM_RETURN_CODE = PLIRETV();
CALL RAIAPI(RAI_TERMINATE,FCA);
IF FCA_RETURN_CODE = 0 THEN
    CALL PLIRETC(FCA_PGM_RETURN_CODE);
```

PRECOMPILE(YES|NO)

PRECOMPILE(YES)

The source program should be modified and enabled for restart.

PRECOMPILE(NO)

The source program should be written out unchanged to the precompiler's SRSOUT file

RESIDENT(YES/NO)

The RESIDENT parameter is designed for use in conjunction with COBOL only. This parameter controls the format of the COBOL call statement used to invoke Smart/RESTART services. RESIDENT(YES) causes Smart/PRECOMPILER to use an identifier in a call statement as in:

```
CALL RAI-RAIAPI USING RAI-INITIALIZE...
```

RESIDENT(NO) causes Smart/PRECOMPILER to use a literal in the call statement as in:

```
CALL 'RAIAPI' USING RAI-INITIALIZE...
```

Relational Architects recommends leaving the default setting of RESIDENT(YES) except when required.

RESTART_RC(YES/NO)

This parameter directs Smart/PRECOMPILER whether or not to generate code which sets the application return code to 2001 upon a RESTART run:

Example 1: RESTART_RC(YES) HOST(IBMCOB)

```
CALL RAI-RAIAPI USING RAI-INITIALIZE, FCA, . . .
IF FCA-RESTART-RUN AND FCA-RETURN-CODE = 0 THEN
    MOVE 2001 TO RETURN-CODE
```

Example 2: RESTART_RC(NO) HOST(IBMCOB)

```
CALL RAI-RAIAPI USING RAI-INITIALIZE, FCA, . . .
IF FCA-RESTART-RUN AND FCA-RETURN-CODE = 0 THEN
    MOVE 0 TO RETURN-CODE
```

Example 3: RESTART_RC(YES) HOST(PLI)

```
CALL RAIAPI(RAI_INITIALIZE,FCA, . . .
IF FCA_STATUS = 'REST' & FCA_RETURN_CODE = 0 THEN
    RETCODE = 0;
```

Example 4: RESTART_RC(NO) HOST(PLI)

```
CALL RAIAPI(RAI_INITIALIZE,FCA, . . .
IF FCA_STATUS = 'REST' & FCA_RETURN_CODE = 0 THEN
    RETCODE = 2001;
```

RM(AUTO|NONE|DB2|IMS|MQS)

Identifies the resource manager (if any) that your application accesses (for validity checking purposes only). RM is synonymous with the DBMS parameter. Specifying AUTO effectively disables this feature and permits the application to access resource managers without restriction.

SAVE_RESTORE(U|C)

Smart/PRECOMPILER can now be directed to conditionally or unconditionally generate code to save and restore the work areas defined to it via the STORAGE and STORAGE_RANGE commands. When SAVE_RESTORE(U) is specified, Smart/PRECOMPILER generates code to save and restore selected data items whether or not these areas are found within the program source code.

SERVER_NAME(name)

The SERVER_NAME command identifies the name of the RAI Jobstream Series (RJS) Server and subsystem. A Smart/PRECOMPILER jobstep attempts to connect to the RJS Server address space with this name.

The name of RJS Server address space is specified via the SERVER parameter in the startup JCL for the RJS server. Make sure to keep the following two settings in synch.

SERVER_NAME	specified by a Smart/PRECOMPILER jobstep
SERVER	specified in the startup JCL for the RJS server

SOURCE

S Directs Smart/PRECOMPILER to write the original, unmodified source program to the SRSPRINT file.

STORAGE(dataname1,dataname2,...)

Identifies the names of data items or structures in your program's storage that should be saved at checkpoint time and restored at restart time. A list of data names should be coded within parentheses and separated by commas. If STORAGE(ALL) is specified, then all storage areas defined with the source module will be saved at checkpoint time and restored at restart time.

NOTE: OS/VS COBOL does not support this feature. Applications coded in OS/VS COBOL should use the alternative STORAGE_RANGE option described next.

STORAGE_RANGE(dataname1,dataname2,|NONE)

Identifies the data items or structures that should be saved at checkpoint time and restored at restart time. A list of data names should be coded within parentheses and separated by commas. These STORAGE_RANGE values must be coded as pairs. Each pair consists of a starting address and an ending address. If STORAGE_RANGE(ALL) is specified, then all storage areas defined within the source module will be saved at checkpoint time and restored at restart time.

The specification STORAGE_RANGE(NONE) identifies the special case in which **no** storage need be saved or restored. STORAGE_RANGE(NONE) overrides the Smart/PRECOMPILER default action to save all working storage when the source module contains no EXEC SRS BEGIN SAVE and EXEC END SAVE statements.

TERMINATE(END|FINISH)

TERMINATE(END) - This option pertains only to source modules written in PL/I. The END option directs Smart/PRECOMPILER to insert a call to the Smart/RESTART TERMINATE service at the physical end of the PL/I main procedure.

TERMINATE(FINISH) - directs Smart/PRECOMPILER to insert a call to the TERMINATE service within any explicitly coded ON FINISH routine. If no explicit ON FINISH routine is coded, Smart/PRECOMPILER generates one for the sole purpose of embedding a call to the TERMINATE service within it. The default is TERMINATE(FINISH).

TRACE(NO|YES)

TRACE(YES)

The trace option is used to diagnose problems or errors within Smart/PRE-COMPILER itself. TRACE(YES) requests that all program clauses be traced before execution.

TRACE(NO)

Nothing is traced.

Figure 3.3 *Smart/PRECOMPILER Options*

3.3.5 PRECOMPILER Return Codes

Return codes and messages for Smart/PRECOMPILER are described in their own section of the Smart/RESTART Messages and Codes manual. Associated with each message is a severity code. The codes and their meanings are as follows:

0	Successful execution
4	Warning
8	Error
12	Severe error

3.4 Compiling your program

A compiler converts your source module into object code (a set of machine instructions that can be executed by the hardware). The language statements processed by the compiler consist of those you coded originally, as well as any new or translated statements inserted by the DB2 precompiler, SQL coprocessor and/or Smart/PRECOMPILER. Figure 3.4.1 illustrates how output from the DB2 precompiler becomes input to Smart/PRECOMPILER. Figure 3.4.2 shows the original, unmodified source program as input to Smart/PRECOMPILER.

```
//PRECOMP EXEC PGM=SRSPC
//SRSIN DD DSN=&DSNHOUT,DISP=(OLD,DELETE) (1)
//SRSOUT DD DSN=&SRSOUT,DISP=(MOD,PASS),UNIT=&UNIT, (2)
// SPACE=(800,(500,500)),DCB=(LRECL=80,BLKSIZE=3120)
//COMPILE EXEC PGM=compiler
//SYSIN DD DSN=&SRSOUT,DISP=(OLD,DELETE) (3)
//SYSLIB DD DSN=DCAvrm.DCASAMP,DISP=SHR (4)
```

where

- (1) Temporary dataset &DSNHOUT (containing output from the DB2 Precompiler) provides input to Smart/PRECOMPILER.
- (2) The SRSOUT DD statement defines the modified source program produced by Smart/PRECOMPILER with restart enabling code added. Temporary dataset &SRSOUT is passed to a subsequent compile step.
- (3) Temporary dataset &SRSOUT (containing output from Smart/PRECOMPILER) in turn becomes input to the COBOL compiler.
- (4) All COBOL copy books and PL/I INCLUDEs as well as some assembly language macros, sample programs and sample JCL are supplied in the Smart/RESTART library whose low level qualifier is *DCASAMP*

Figure 3.4.1 Output from DB2 Precompiler is input to Smart/PRECOMPILER

```

//PRECOMP EXEC PGM=SRSPC
//SRSIN DD DSN=original.source.program,DISP=SHR (1)
//SRSOUT DD DSN=&SRSOUT,DISP=(MOD,PASS),UNIT=&UNIT, (2)
// SPACE=(800,(500,500)),DCB=(LRECL=80,BLKSIZE=3120)
//COMPILE EXEC PGM=compiler
//SYSIN DD DSN=&SRSOUT,DISP=(OLD,DELETE) (3)
//SYSLIB DD DSN=DCAvrm.DCASAMP,DISP=SHR (4)

```

where

- (1) The original, unmodified source program is input to Smart/PRECOMPILER.
- (2) The SRSOUT DD statement defines the modified source program produced by Smart/PRECOMPILER with restart enabling code added. Temporary dataset &SRSOUT is passed to a subsequent compile step.
- (3) Temporary dataset &SRSOUT (containing output from Smart/PRECOMPILER) in turn becomes input to the COBOL compiler.
- (4) All COBOL copy books and PL/I INCLUDEs as well as some assembly language macros, sample programs and sample JCL are supplied in the Smart/RESTART library whose low level qualifier is DCASAMP.

Figure 3.4.2 Source program input to Smart/PRECOMPILER

3.5 Link Editing your Program

NOTE: *If your program calls all its subprograms dynamically (e.g. a COBOL module compiled with the DYNAM option), you can skip this Section (and its sub-sections) entirely. These considerations are deferred until Chapter 4 which discusses execution JCL.*

Your application must be link-edited with the interface routines for Smart/RESTART and optionally the interface routines for DB2, IMS, MQ and/or RRS as well. The material presented here considers applications which use Smart/RRSAF to access DB2 as well as applications which rely on RRS to maintain consistency when multiple resources are accessed within a single unit-of-work.

Part II of this manual considers the link edit requirements of restartable applications which access IMS or MQ resources in 'standalone mode' as well as applications which access DB2 through attachment packages other than Smart/RRSAF.

Smart/RESTART provides an Applications Programming Interface routine named RAIAPI through which Smart/RESTART services are executed. By link editing RAIAPI, you resolve any explicit calls you may have coded to the Smart/RESTART API as well as any calls inserted by Smart/PRECOMPILER.

[DB2-dependent Section]

In addition, applications which contain embedded SQL statements must link-edit a DB2 attachment facility to provide access to DB2. ***RAI strongly recommends use of Smart/RRSAF as a DB2 attachment facility.***

The DB2 attachment package supplied with Smart/RRSAF is named DCAHLI. It provides the linkage for both SQL calls (through its alias DSNHLI) as well as IFI calls (through its alias DSHWLI). See the Smart/RRSAF User Guide for additional information. The use of other DB2 attachment packages is discussed in Chapter 22.

[End DB2-dependent Section]

Sections 3.5.1 and 3.5.2 illustrate two procedures through which applications can resolve external references to the interface routines for both Smart/RESTART and Smart/RRSAF at link-edit time.

3.5.1 Using Explicit Linkage Editor INCLUDE Statements

Figure 3.5 illustrates a link edit jobstep for the restartable DB2 application program named EXAMPLE in which the linkage editor is directed to explicitly include the interface routines required by Smart/ RESTART and Smart/RRSAF.

```
//LINK      EXEC PGM=IEWL,PARM=' '  
//SYSLIB    DD DSN=automatic.call.library,DISP=SHR  
//RAILIB     DD DSN=DCAvrM.DCANAL,DISP=SHR          (1)  
//SYSLIN     DD DSN=8OBJSET,DISP=(OLD,DELETE)       (2)  
//          DD *  
            INCLUDE RAILIB(RAIAPI)                  (3)  
            INCLUDE RAILIB(DCAHLI)                  (4)  
            INCLUDE RAILIB(SRS4SRR)                  (5)  
            NAME      EXAMPLE(R)  
//
```

where

- (1) The RAILIB DD statement defines the Smart/RESTART system library which contains the interface routines for Smart/RESTART and Smart/RRSAF. Ask your Smart/RESTART product administrator for the name of this dataset in your environment.
- (2) The SYSLIN file identifies the object module produced by the compiler or assembler.
- (3) The first INCLUDE control statement directs the linkage editor to fetch the load module named RAIAPI from the dataset defined by the RAILIB DD statement and incorporate it in the executable load module.
- (4) The second INCLUDE control statement directs the linkage editor to fetch the load module named DCAHLI from the dataset defined by the RAILIB DD statement and incorporate it in the executable load module. DCAHLI is the name of the DB2 attachment package supplied with Smart/RRSAF.
- (5) Applications which update resources managed by multiple RRS compliant products (such as DB2, WebSphere MQ and IMS) within a single unit-of-work should specify a third INCLUDE control statement to direct the linkage editor to fetch the load module named SRS4SRR from the dataset defined by the RAILIB DD statement and incorporate it in the executable load module. Smart/RESTART provides an application stub named SRS4SRR that defines SRRCMIT, SRRBACK and ATRSCSS as *alternate entry points*. SRS4SRR provides the link between your batch program and RRS.

Figure 3.5 *Smart/RESTART Link-Edit Jobstep*

3.5.2 Using Automatic Library Call

The linkage editor's automatic library call mechanism can include the Smart/RESTART and Smart/RRSAF interfaces into the link edited load module -- **without** the need for explicit INCLUDE statements. To enable automatic library call, simply add the system library which contains the Smart/RESTART and Smart/RRSAF interface routines to the SYSLIB concatenation defined for the linkage editor -- as illustrated in Figure 3.6:

```
//LKED      EXEC PGM=IEWL,....  
//SYSLIB DD  DSN=DCAvrn.DCANCAL,DISP=SHR          (1)  
//          DD  DSN=DSNxxx.SDSNLOAD,DISP=SHR  
//          DD  DSN=user.nca1.library,DISP=SHR
```

where

- (1) The SYSLIB DD statement defines the Smart/RESTART system library which contains the interface routines for Smart/RESTART and Smart/RRSAF. The Smart/RESTART Applications Programming Interface is called RAIAPI, the Smart/RESTART application stub that defines SRRCMIT, SRRBACK and ATRSCSS as *alternate entry points* is named SRS4SRR and the DB2 attachment package supplied with Smart/RRSAF is named DCAHLI.

DCAHLI provides the linkage for both SQL calls (through its alias DSNHLI) as well as IFI calls (through its alias DSHWLI). Applications which update resources managed by multiple RRS compliant products (such as DB2, WebSphere MQ and IMS) within a single unit-of-work also link the module named SRS4SRR and incorporate it in the executable load module. Smart/RESTART provides an application stub named SRS4SRR that defines SRRCMIT, SRRBACK and ATRSCSS as *alternate entry points*. SRS4SRR provides the link between your batch program and RRS. Ask your Smart/RESTART product administrator for the name of this dataset at your site.

Figure 3.6 Smart/RESTART Link Edit Jobstep

NOTE: The linkage editor's automatic library call mechanism fetches the **first** copies of DSNHLI, SRRCMIT and SRRBACK it finds to resolve external references. Therefore, if you want to use Smart/RRSAF you must ensure the dataset containing the Smart/RRSAF versions of DSNHLI and DSNWLI appear **ahead** of any DB2 system load library in the link edit step's SYSLIB concatenation.

In addition, applications which utilize Smart/RESTART support for RRS must ensure the dataset containing the Smart/RESTART versions of SRRCMIT and SRRBACK appear first in the link edit step's SYSLIB concatenation, **ahead** of those provided by RRS in data set SYS1.CSSLIB or any RRS stubs provided with resource managers like DB2 and WebSphere MQ.

3.6 BIND your Application Plan [DB2-dependent Section]

In the DB2 environment, Smart/RESTART stores synchronizing information in its own DB2 table that corresponds to data recorded in the Smart/RESTART checkpoint file. This synchronizing information is included in the same Unit-of-Recovery for which your application requests a COMMIT. This two phase commit protocol ensures that your application's working storage and sequential file position remain synchronized with DB2 updates -- regardless of whether the unit-of-work succeeds or fails.

These DB2 table updates are performed by a Smart/RESTART module whose package or DBRM must be bound into your application's DB2 plan. The name of the Smart/RESTART database request module for DB2 is SRSDBRM - a member of the Smart/RESTART DBRM library. Ask your Smart/RESTART product administrator for the name of this dataset at your site or the name of the collection which contains the SRSDBRM package.

3.6.1 BIND DBRM for SRSDBRM [DB2-dependent Section]

Smart/RESTART has one DB2 program, SRSDBRM, and all restartable DB2 applications need to include it in their plans. Program SRSDBRM issues SQL statements to the DB2 table used internally by Smart/RESTART.

Plans that consist entirely of DBRMs need to have the DBRM for program SRSDBRM included in the BIND. Plans using packages can access the package for program SRSDBRM in a number of ways.

3.6.2 BIND Package for SRSDBRM [DB2-dependent Section]

If there is a shared collection bound as part of every plan, then simply add the package for SRSDBRM to this shared collection. Without a shared collection, the package for SRSDBRM needs to be included in at least one collection in the package list (PKLIST in the BIND command) for each plan used by a restartable DB2 application. The DB2 BIND PACKAGE COPY command can be used to copy a package into multiple collections.

3.6.3 Package BIND for Applications that issue

SQL SET CURRENT PACKAGESET statements

[DB2-dependent Section]

DB2 applications that set the CURRENT PACKAGESET special register change the normal search order for finding plan elements (DBRM's and packages) in a plan. When the CURRENT PACKAGESET special register is set to a non-blank value, the search for packages is restricted to the collection specified in the SET CURRENT PACKAGESET statement. Any other collections in the package list of the plan are ignored.

To assist applications that issue SQL SET CURRENT PACKAGESET statements, Smart/RESTART provides two run-time parameters, SET_PACKAGE and COLLECTION. These parameters direct the SRSDBRM program to save the value of the current package collection, issue a SET CURRENT PACKAGESET statement to specify a collection ID *known* to contain the SRSDBRM package, issue its SQL statements and finally, return the CURRENT PACKAGESET to its initial value with another SET CURRENT PACKAGESET statement.

This processing is intended for development and test environments *only* since routine PACKAGESET switching has very high overhead. For production applications that set the CURRENT PACKAGESET special register, RAI strongly recommends the SRSDBRM package be bound into every package collection that may be included within the DB2 plans of such restartable applications. Chapter 21 discusses this subject in greater detail.

NOTE: *Smart/RESTART run-time parameters are introduced in Chapter 4 while the SET_PACKAGE and COLLECTION run time parameters are described and illustrated in Chapter 14 of the Smart/RESTART Reference.*

Program Execution and Restart

This chapter discusses how to run applications with Smart/RESTART. Section 4.1 describes how the same JCL can be used for both initial and restart runs. Section 4.2 describes the facilities Smart/RESTART provides to test application failure, recovery and restart -- without changing source code. Section 4.4 describes how to communicate with Smart/RESTART during a run from an operator console or Smart/MONITOR. Chapter 4 concludes with a discussion of various operational topics such as how to recovery from out of space failures.

NOTE: *Advanced operational topics such as point-in-time recovery appear in Part V - Advanced Features.*

The material presented in this chapter is specific to applications which access sequential files and/or DB2 data using Smart/RRSAF as a DB2 attachment facility. Chapter 20 describes Smart/RESTART support for applications that make changes to DB2, IMS, MQ and other RRS compliant resources within a single unit of recovery. Applications which access IMS or MQ resources in 'standalone mode' as well as application which utilize DB2 attachment facilities other than Smart/RRSAF are described in separate chapters of their own within Part III - Support for Resource Managers.

4.1 JCL for Initial and Restart Runs

Figure 4.1 illustrates the JCL for an application that accesses both sequential files and DB2. The same JCL can be used for both initial and restart runs because all files reference pre-allocated datasets. The numbers in the right hand margin (within parentheses) correspond to the numbered annotations which follow the JCL.

```

//STEP1 EXEC PGM=EXAMPLE,PARM='EXAMPLE PARMs'      (1)
//STEPLIB DD DSN=user.program.library,DISP=SHR      (2)
//          DD DSN=DCAvrn.DCALOAD,DISP=SHR
//          DD DSN=DSNxxx.SDSNLOAD,DISP=SHR
//SRSCHECK DD DSN=check.point.dataset,DISP=SHR      (3)
//SRSPRINT DD SYSOUT=*                               (4)
//INFILE DD DSN=program.input.dataset,DISP=OLD      (5)
//OUTFILE DD DSN=program.output.dataset,DISP=OLD    (6)
//RAINPUT DD *                                       (7)
CKPT_PACE(50)                                       (8)
//

```

where

- (1) The DB2/COBOL application named EXAMPLE **looks** and **behaves** like a regular OS job because it uses Smart/RRSAF as a DB2 attachment facility. Thus, the EXAMPLE program is invoked directly via the JCL EXEC statement and receives its parameters in the conventional way.

***NOTE:** Smart/RRSAF is a standard component of Smart/RESTART which is described in its own manual (Publication DCA-002).*

- (2) The STEPLIB DD statement defines a concatenated set of load module libraries. (Discussed in Section 4.1.1.)
- (3) The SRSCHECK DD statement defines a pre-allocated Smart/RESTART checkpoint dataset defined via JCL as discussed in Section 4.1.2. Alternatively, Section 4.1.2.1 discusses how the SRSCHECK file can be dynamically allocated.
- (4) The SRSPRINT DD statement defines the Smart/RESTART message log. RAI recommends you allocate an SRSPRINT file rather than issue Smart/RESTART messages via WTO because the latter option can flood the console and system log with non essential messages. Smart/RESTART can also dynamically allocate the SRSPRINT file as described in Section 4.1.3.
- (5) The INFILE DD statement defines a pre-allocated, sequential input file that must be repositioned should the application fail and be restarted. (Discussed in Section 4.1.4.)
- (6) The OUTFILE DD statement defines a pre-allocated, sequential output file that must be repositioned in the event of restart. (Discussed in Section 4.1.4.)
- (7) The RAINPUT DD statement supplies run-time parameters to Smart/RESTART as described in Section 4.1.6
- (8) You can control Smart/RESTART's run-time behavior through parameters passed in the RAINPUT stream. For example, the CKPT_PACE parameter governs checkpoint pacing.

Figure 4.1 JCL to run an application with Smart/RESTART

4.1.1 Load Module Libraries

The STEPLIB DD statement in Figure 4.2 defines a concatenated set of load module libraries which includes the user library in which the application load module or program object resides as well as the system libraries for Smart/RESTART, Smart/RRSAF and DB2. The numbers in the right hand margin (within parentheses) correspond to the numbered annotations which follow the JCL.

```
//STEPLIB DD DSN=user.program.library,DISP=SHR (1)
//          DD DSN=DCAvrn.DCALOAD,DISP=SHR (2)
//          DD DSN=DSNxxx.SDSNLOAD,DISP=SHR (3)
```

where

- (1) The application named EXAMPLE is fetched from the user program library named user.program.library.
- (2) The DCAvrn.DCALOAD dataset defines the Smart/RESTART system library which contains the executable code for both Smart/RESTART and Smart/RRSAF. This dataset should be defined **ahead** of the DB2 system library.

NOTE: The component code for Smart/RESTART and Smart/RRSAF is DCA (which originally stood for DB2 Call Attach).

- (3) The DB2 system library containing DB2 load modules should be defined last.

Figure 4.2 STEPLIB Concatenation

Alternatively, these user and system load modules can be fetched from libraries defined in the MVS LINKLIST concatenation. In this case, you must ensure the DB2 attachment facility supplied with Smart/RRSAF (module DCAHLI) as well as the Smart/RESTART versions of SRRCMIT and SRRBACK are utilized at execution time. See Chapter 4 of the Smart/RRSAF User Guide for details.

4.1.2 Defining the Smart/RESTART Checkpoint Dataset

RAI recommends the checkpoint file be pre-allocated using JCL like that illustrated in Figure 4.3. This allows you to resubmit the application JCL without change, in the event the application fails and needs to be restarted. Alternatively, Section 4.1.2.1 describes how Smart/RESTART can dynamically allocate a checkpoint file during initial and restart runs.

```
//ALLOCATE EXEC PGM=IEFBR14
//SRSCHECK DD DSN=checkpoint.dataset.name,
//          DISP=(NEW,CATLG,CATLG),
//          RECOG=LS,                      (1)
//          SPACE=(TRK,60)                 (2)
```

where

- (1) RAI recommends the checkpoint file be defined as a VSAM LDS (linear data set). You need not specify an LRECL or BLKSIZE.
- (2) A space allocation of 60 tracks for the checkpoint dataset should be ample for nearly all applications. See the CKPT_SIZE and CKPT_DEPTH parameters described in Chapter 14 of the Smart/RESTART Reference for further details.

Figure 4.3 *Pre-allocating a VSAM LDS Checkpoint File*

RAI recommends that each job have its own checkpoint file. If the same application runs concurrently in separate jobs, then each concurrently active jobstep must define a unique checkpoint dataset. For example, suppose JOB1 runs PROGRAM1 and uses checkpoint dataset A. JOB2 cannot run PROGRAM1 and also use checkpoint dataset A. If it does, then JOB2 will behave as a restart of JOB1.

Smart/RESTART can maintain checkpoint data in a disk or tape dataset as well as in a DB2 table. Checkpoint overhead depends on checkpoint frequency, the amount of data saved and the checkpoint media. Checkpointing overhead can be reduced by limiting the amount of data saved. This User Guide describes a number of techniques to selectively identify the storage areas to save at checkpoint time. To achieve the best performance, RAI recommends the use of VSAM linear datasets as checkpoint files. Chapter 41 presents a detailed discussion of the checkpoint media options supported by Smart/RESTART.

Checkpoint files are sometimes used to ensure that a failed job is restarted from the failed jobstep rather than the first jobstep. This can be accomplished by creating a new checkpoint file in the beginning of a job, statically allocating it in your application via an SRSCHECK DD statement, and then deleting it at the end of the job. Should a failed job be incorrectly restarted from the first step, the allocation of a new checkpoint file will fail and prevent the rest of the job from running. Alternatively, Section 4.6 describes the

SRSUTIL utility which serves the same purpose and works with dynamically allocated checkpoint files as well as those statically allocated via JCL.

4.1.2.1 Calculating the Checkpoint Size

- calculate a value for the Smart/RESTART CKPT_SIZE operand - in 4K blocks
- allocate an appropriate amount of space for the checkpoint dataset

The CKPT_SIZE (in 4 K blocks) can be calculated as follows:

CKPT_SIZE =

```
(base requirement of 85 KB
+ (the number of repositionable datasets * 1 KB)
+ (the number of repositionable DB2 cursors * 1 KB)
+ (the number of registered modules * 1 KB)
+ the amount of checkpointed storage in KB)

/ 4
```

where:

- Smart/RESTART requires 85 KB for its basic control information
- Smart/RESTART requires 1KB for each repositionable file and concatenated dataset. A repositionable file is a DDname identified (either explicitly or by default) through the REPOSITION operand or refers to a file accessed via the Smart/RESTART API. Each additional dataset within a repositionable file concatenation requires another 1 KB (as described and illustrated in Figure 1 below)
- each repositionable DB2 cursor identified via the Smart/RESTART operands REPOSITION_CSR_AT and/or REPOSITION_CSR_AFTER requires an additional 1 KB
- each application module that *registers* with Smart/RESTART via the INITIALIZE service requires an additional 1 KB
- the amount of storage in KB to be saved at checkpoint time and restored at restart time

A Restartable Jobstep

Consider the restartable application whose JCL appears in Figure 1. The numbers in the right hand margin (within parentheses) correspond to the numbered annotations which follow the figure.

```

//      JOB
//S1     EXEC PGM=EXAMPLE                      (1)
//INDD1  DD DSN=input.dataset.one,DISP=SHR      (2)
//INDD2  DD DSN=concatenated.dataset.one,DISP=SHR (3a)
//      DD DSN=concatenated.dataset.two,DISP=SHR (3b)
//      DD DSN=concatenated.dataset.three,DISP=SHR (3c)
//OUTDD1 DD DSN=output.dataset.one,DISP=SHR     (4)
//RAINPUT DD *                                  (5)
REPOSITION(ALL),REPOSITION_CSR_AT(ONLY(SUBRTN1,CURS0R1))

```

where

- (1) EXAMPLE is a multi module application whose structure is described below in Figure 2
- (2) File INDD1 defines a single input dataset
- (3) INDD2 defines three input datasets concatenated as a single file
- (4) File OUTDD1 defines a single output dataset
- (5) The RAINPUT file specifies explicit runtime directives for Smart/RESTART. In this example, REPOSITION(ALL) specifies that *all* files are to be considered repositionable. In contrast, *only* the explicitly designated cursor is to be considered repositionable.

Figure 1 JCL for a Restartable Jobstep

The Restartable Application Program

The EXAMPLE program invoked via the JCL EXEC statement is comprised of 10 independently written and compiled modules. The main module and four subroutines are restart enabled and thus *registered* with Smart/RESTART. The remaining 5 subroutines are not restart enabled.

The Main module is restart enabled via Smart/PRECOMPILER. It identifies 20 KB of storage to be saved at checkpoint time and restored at restart time.

- Subroutine 1 explicitly calls the INITIALIZE service and identifies 5 KB of storage to checkpoint
- Subroutine 2 is restart enabled via the Smart/RESTART PL/I %INCLUDEs. It identifies 1 KB of storage to checkpoint
- Subroutine 3 is restart enabled by embedding the Smart/RESTART COBOL copy members. It identifies 2 KB of storage to checkpoint
- Subroutine 4 issues an explicit EXEC SRS INITIALIZE request to become restart enabled. It identifies 3 KB of storage to checkpoint
- Subroutines 5 through 9 are not restart enabled

Figure 2 Module Structure of the EXAMPLE application

NOTE: RAI recommends that all storage to be saved at checkpoint time and restored at restart time be defined in the application's main module. As such, only the main module need register with Smart/RESTART (via one of the supported modes of restart enablement).

Several modules of the EXAMPLE application are restart enabled *solely* to illustrate the calculation of CKPT_SIZE in a more comprehensive scenario.

The Checkpoint Size Calculation

The CKPT_SIZE requirement for the EXAMPLE program is calculated as follows:

CKPT_SIZE =

(base requirement of 85 KB
+ (5 repositionable datasets * 1 KB)
+ (1 repositionable DB2 cursor * 1 KB)
+ (5 registered modules * 1 KB)
+ (5 + 1 + 2 + 3 KB of storage to checkpoint)

/ 4

Summing the terms yields $85 + 5 + 1 + 5 + 11 = 107$ KB

We round this sum up to the next multiple of 4 and then divide by 4 giving

$108 \text{ KB} / 4 = 27 \text{ 4KB blocks}$

As such, the appropriate CKPT_SIZE specification for the EXAMPLE application is CKPT_SIZE(27)

Allocating the Checkpoint Dataset

The space allocation for the checkpoint dataset (in 4 K blocks) can be calculated as follows:

$\text{Allocation (in 4K blocks)} = (\text{CKPT_SIZE} * \text{CKPT_DEPTH}) + 1$

where:

- CKPT_SIZE as calculated previously
- CKPT_DEPTH specifies how many checkpoints Smart/RESTART should maintain as points of consistency from which to possibly restart.

Assuming a default CKPT_DEPTH of two, the space allocation for the EXAMPLE application's checkpoint dataset is calculated as follows:

$\text{Space allocation} = (27 * 2) + 1 = 55 \text{ 4K blocks}$

4.1.2.2 Dynamic Allocation of Checkpoint Files

Smart/RESTART can dynamically allocate its checkpoint dataset in both initial runs and restart runs to eliminate the need to specify the SRS CHECK DD statement. Two requirements must be met in order for Smart/RESTART to dynamically allocate the correct checkpoint dataset in a restart run:

1. The Job ID must derive from either the JOBID, AUTO_JOBID(JOBNAME) or AUTO_JOBID(SYSTEMANDJOBNAME) parameters (described in Chapters 12 and 14 respectively of the Smart/RESTART Reference). There are no restrictions on the job names assigned to initial or restart run(s) with the JOBID parameter. Conversely, the job name for the initial run **must** be the same as that for any subsequent restart runs when the AUTO_JOBID parameter is used. RAI recommends use of an explicit JOBID or the AUTO_JOBID(JOBNAME) option to permit seamless restart elsewhere in a sysplex.
2. Smart/RESTART must maintain synchronizing data for the job step.

Synchronizing data is typically maintained for each restartable job step. A job step that accesses any IBM resource manager (such as DB2, IMS, MQ) causes Smart/RESTART to maintain synchronizing data. Only application job steps that are limited to updating sequential files and/or VSAM files will not have synchronizing data. Should you wish to dynamically allocate checkpoint files for a file-only application, include the RM(DB2) parameter in RAINPUT to force Smart/RESTART to maintain synchronizing data.

The absence of synchronizing data for the Job ID indicates an initial run. This causes Smart/RESTART to allocate a new checkpoint file. The Job ID is stored in the new checkpoint file and new synchronizing data is created for the current Job ID.

If Smart/RESTART finds synchronizing data, it dynamically allocates the checkpoint dataset whose name is in the synchronizing data. The checkpoint dataset is read and validity checks are performed to ensure it is the correct file. One of these checks is to verify the Job ID stored in the checkpoint dataset. If the checkpoint dataset is valid, the restart information it contains is used as the basis for the current job step. This restart information can indicate either an initial run or a restart run.

4.1.3 Dynamic Allocation of the SRSPRINT File

Smart/RESTART output messages go to either SRSPRINT (by default) or to the DD name specified via the MSGDD profile parameter. Smart/RESTART can dynamically allocate its output message dataset, eliminating the need to include a DD statement for it in the JCL for the job step. Every dynamically allocated message dataset uses the same SYSOUT class. This class is specified when Smart/RESTART is installed and defaults to "H".

4.1.4 Pre-allocating Application Files

RAI recommends you **preallocate** any disk resident, sequential files that must be repositioned on restart. If you pre-allocate the datasets your application requires, (i.e. specify a dataset disposition of OLD or SHR), you can use the same JCL for both initial and restart runs. There's no need to change the JCL before resubmitting the job for restart.

Conversely, if you first allocate a sequential file during the initial run (i.e. specify a disposition of NEW), you must change the dataset disposition to OLD or SHR before resubmitting the job for restart. Failure to do so will **prevent** successful restart. Note, however, that no JCL changes are required for tape files because Smart/RESTART can automatically override a dataset disposition of NEW to MOD.

Never specify DELETE as an abnormal dataset disposition for any repositionable file. Should the jobstep abend, the system removes any file whose abnormal disposition specifies DELETE.

NOTE: By default, Smart/RESTART provides repositioning support for **all** the files your application accesses sequentially -- **except** those allocated to SYSIN, SYSOUT or DUMMY (as described in Section 4.1.4). Smart/RESTART lets you identify repositionable files outside the scope of your application via the REPOSITION run-time parameter. You can identify repositionable files selectively on a DDname basis or refer to them globally as ALL or NONE. See Chapter 11 of the Smart/RESTART Reference for a complete discussion.

4.1.5 SYSIN, SYSOUT and DUMMY files

Smart/RESTART cannot reposition sequential files allocated to SYSIN, SYSOUT or DUMMY. Smart/RESTART performs standard (non-repositionable) I/O when it detects a reference to a SYSIN, SYSOUT or DUMMY file. In such cases, Smart/RESTART also returns a warning or error indication to your application (depending on the site default).

Consider the following circumvention if a print file (allocated to SYSOUT) must be repositioned on restart: Convert the SYSOUT file to a sequential dataset on disk. Then, follow the restartable step with an IEBGENER step that executes unconditionally (COND=EVEN). This IEBGENER step can write the disk dataset to SYSOUT.

4.1.6 Smart/RESTART Run-time Parameters

Smart/RESTART run-time operation is governed by runtime parameters. These options are typically defined as defaults when Smart/RESTART is installed. The RAINPUT DD statement (like the one illustrated in Figure 4.4), defines an optional dataset containing run-time parameters for both Smart/RESTART and Smart/RRSAF. Any parameters you specify will override the corresponding profile defaults.

```
//RAINPUT DD *           (1)
CKPT_PACE(50)           (2)
//
```

where

- (1) The RAINPUT DD statement supplies run-time parameters to both Smart/RESTART and Smart/RRSAF.
- (2) You can control Smart/RESTART's run-time behavior through parameters passed in the RAINPUT stream. For example, the CKPT_PACE parameter governs checkpoint pacing.

Figure 4.4 *Supplying run-time parameters to Smart/RESTART*

When you allocate the RAINPUT file in your jobstep, Smart/RESTART opens the file and reads your specifications. Most parameters are specified as keyword/value pairs in the form KEYWORD(value). Parameters may be specified in any order, on one or more statements. You can supply all Smart/RRSAF and Smart/RESTART parameters in a single instream file as illustrated in Figure 4.4 above. Alternatively, you can specify RAINPUT as a concatenation of unique and/or shared runtime parameters from several sources as described in Sections 4.1.6.1 and 4.1.6.2.

The complete set of Smart/RESTART profile parameters are described in the Smart/RESTART Reference. See the Smart/RRSAF User Guide for descriptions of the keyword parameters associated with that product.

4.1.6.1 Using a Smart/RESTART Run-time Parameter Library

Instead of specifying RAINPUT parameters *instream* with the JCL, your organization can create a library to contain Smart/RESTART runtime parameters in which each member corresponds to a particular restartable application. Figures 4.5 and 4.6 below illustrate how a restartable application named EXAMPLE can reference its runtime options maintained in members of a Smart/RESTART parameter library. The RAINPUT DD statement in Figure 4.5 utilizes the runtime parameters maintained in the EXAMPLE member of the DCAINPUT library (shown in Figure 4.6 below).

```
//MYSTEP EXEC PGM=EXAMPLE
//RAINPUT DD DISP=SHR,DSN=h1q.DCAINPUT(EXAMPLE)
```

Figure 4.5 *RAINPUT parameters reside in a PDS member*

```
BROWSE      h1q.DCAINPUT(EXAMPLE) - 01.00          Line 00000000 Col 001 080
Command ==>                                     Scroll ==> HALF
***** Top of Data *****
      CKPT_PACE(50)          /* Govern Checkpoint Pacing */
***** Bottom of Data *****
```

Figure 4.6 *Run-time parameters for the EXAMPLE application*

Specifying RAINPUT runtime options as members of a Smart/RESTART parameter library has several advantages over specifying runtime options *instream* with the JCL. First, such a job step can be incorporated in any catalogued JCL procedure whereas a job step with instream RAINPUT defined cannot. Moreover, there is no need to modify the JCL should Smart/RESTART runtime options need to change.

4.1.6.2 Sharing Smart/RESTART Run-time Parameters

Smart/RESTART lets you *share* sets of Smart/RESTART run-time parameters among applications with similar processing characteristics and requirements. As examples, shared sets of runtime options might be defined for long-running applications, for applications which access DB2 or for programs which run during peak processing hours.

Your organization can create one or more Smart/RESTART parameter libraries (as described in the previous section) whose members can be shared among multiple applications. You can even specify RAINPUT as a concatenation of unique and/or shared runtime parameters from several sources as illustrated in Figure 4.7 below.

```
//          EXEC PGM=PAYROLL1
//RAINPUT DD DISP=SHR,DSN=h1q.DCAINPUT(PAYROLL) /* Shared PAYROLL parms      (1)
//          DD *                                /* Unique instream parms      (2)
MSG_DISPLAY(DEBUG)                                (3)
//          EXEC PGM=PAYROLL2
//RAINPUT DD DISP=SHR,DSN=h1q.DCAINPUT(PAYROLL) /* Shared PAYROLL parms      (1)
```

where

- (1) The PAYROLL1 and PAYROLLB2 applications share the runtime parameters (illustrated below in Figure 4.8) defined in the PAYROLL member of the DCAINPUT library. Any number of Payroll programs might share these same defaults.
- (2) Suppose the PAYROLL1 application needs its Smart/RESTART processing to be traced for debugging purposes. Unique runtime options can be specified in a separate dataset or instream as illustrated in this example.
- (3) The MSG_DISPLAY(DEBUG) option directs Smart/RESTART to produce debugging messages which reflect its internal processing.

Figure 4.7 *Sharing run-time parameters between two PAYROLL applications*

```
BROWSE      h1q.DCAINPUT(PAYROLL) - 01.00                Line 00000000 Col 001 080
Command ==>                                           Scroll ==> HALF
***** Top of Data *****
      CKPT_PACE(25) /* Checkpoint Pacing criteria for all PAYROLL applications */
***** Bottom of Data *****
```

Figure 4.8 *Run-time options shared by all PAYROLL applications*

4.2 Application Restart

For purposes of application restart, processing always resumes from the point of consistency associated with a *physical* checkpoint. In other words, a logical checkpoint or SQL COMMIT request which does not trigger a physical checkpoint is a no-operation with respect to both restart and rollback.

NOTE: All initial and subsequent restart runs associated with a single restartable job instance *MUST* be executed under the SAME Smart/RESTART release.

4.3 Development and Testing Procedures

Smart/RESTART provides a pair of run-time parameters to facilitate application development and testing. For example, the **TEST_ABEND** parameter provides a convenient method to test an application's restartability during development. TEST_ABEND requests a user abend after NNNN physical checkpoints *or* NNNN logical checkpoints, whichever occurs first. TEST_ABEND enables you to test abend/recovery/restart processing without modifying the source program. TEST_ABEND can be specified in the RAINPUT stream or can be issued from the MVS console or Smart/MONITOR. Chapter 11 of the Smart/RESTART Reference describes TEST_ABEND in detail.

The CLEAR_CKPT parameter is also useful during development to force an initial run. CLEAR_CKPT governs when the contents of the checkpoint dataset is cleared. By default, Smart/RESTART clears the contents of the checkpoint dataset (and any synchronizing data maintained by a resource manager like DB2) when the application terminates normally. During testing, developers may wish to repeatedly submit an initial run using the same checkpoint dataset, even if the application fails. This can be achieved by specifying CLEAR_CKPT with a value of BEFORE. CLEAR_CKPT(BEFORE) directs Smart/RESTART to ignore the contents of the checkpoint dataset and ensures the application executes as an initial run rather than a restart run. See Chapter 11 of the Smart/RESTART Reference for further details.

NOTE: CLEAR_CKPT(BEFORE) should ***never*** be used for production operation since it prevents step restart.

4.4 Communicating with Smart/RESTART during Execution

You can alter the behavior of Smart/RESTART during execution of a restartable application by issuing Smart/RESTART commands from the MVS console or via Smart/MONITOR. The console interface accepts the same syntax as that specified through the RAINPUT DD statement.

For example, the console operator can issue the following MVS MODIFY command to direct Smart/RESTART to gracefully cancel the job named EXAMPLE when it next requests a checkpoint:

MODIFY EXAMPLE,QUIESCE

You can also issue these commands through Smart/MONITOR, an ISPF dialog described in Part IV of this publication.

In addition, the application can dynamically alter Smart/RESTART behavior by issuing PROFILE requests at run-time. Chapter 10 describes the EXEC SRS PROFILE command while Chapter 11 describes the Call Level Interface to the Smart/RESTART PROFILE service.

4.4.1 Production Operation

The following Smart/RESTART profile commands are typically entered from the MVS Console or Smart/MONITOR. Chapter 13 of the Smart/RESTART Reference describes these commands in detail.

CANCEL	gracefully halt an application at the next Unit-of-Work boundary - irrespective of checkpoint pacing
CKPT_PACE	Govern checkpoint pacing
CKPT_IMMED	Trigger an immediate physical checkpoint
CKPT_TIME	Govern checkpoint pacing on the basis of elapsed time
EST_RUN_TIME	Specify an estimated runtime for a jobstep
FORCE	Immediately halt an application
PREDICT	Activate or disable Smart/PREDICTOR
QUIESCE	gracefully halt an application at a Unit-of-Work boundary
RESUME	Resume execution of a suspended application
SUSPEND	Make an application non-dispatchable for a period of time
TEST_ABEND	Test restart processing without changing the source program

Figure 4.9 MVS Console commands for Smart/RESTART

4.4.2 Cancelling Restartable Jobs

RAI strongly recommends that MVS console operators issue Smart/RESTART QUIESCE, CANCEL or FORCE commands rather than MVS CANCEL and FORCE commands to terminate restartable jobs. Use of the Smart/RESTART QUIESCE, CANCEL or FORCE commands ensures application files are closed properly for a subsequent RESTART run. In contrast, the MVS CANCEL and FORCE commands may cause problems when you submit a restart run.

Smart/RESTART's QUIESCE, CANCEL and FORCE commands all halt a production application by issuing an abend. The commands differ in terms how gracefully and consistent with normal processing the abnormal termination is as well as how much application processing can occur before the command takes effect.

Smart/RESTART's QUIESCE and CANCEL commands allow an operator to gracefully halt an application at a unit-of-work. Both commands avoid the need to discard successfully processed work and eliminate the potentially costly and time consuming rollback of an inflight unit-of-work.

With QUIESCE, standard pacing criteria govern when Smart/RESTART honors a logical checkpoint request from the application as a physical checkpoint event. In contrast, the CANCEL command physically honors the application's next checkpoint request, without regard to checkpoint pacing. In both cases, Smart/RESTART abends the application once the physical checkpoint is complete. This enables the jobstep to resume execution at a more propitious time – with no wasted processing.

Use the FORCE command to immediately terminate a restartable jobstep and cause processing performed during the current unit-of-work to be backed out. Operators can issue the Smart/RESTART FORCE command in order to free locks and relieve resource contention. The application may be restarted from the last valid checkpoint or from a prior checkpoint (provided point in time recovery is performed).

Chapter 13 of the Smart/RESTART reference describes Smart/RESTART's QUIESCE, CANCEL and FORCE commands which may be issued from the MVS Console or through the Smart/MONITOR dialogs.

4.5 Out of Space Failures

The typical response to an x37 abend is to reallocate a repositionable output file with more space. Next, the original file must be copied into the newly allocated, larger dataset. During restart, Smart/RESTART repositions to the correct output record (which now resides within the newly allocated dataset).

Smart/RESTART supports such file relocation so long as dataset attributes and device type remain the same. However, special care must be taken while relocating repositionable (RSAM) datasets.

Figure 4.10 below illustrates a two-step IEBGENER job with which to copy and relocate a blocked RSAM dataset. Smart/RESTART supplies this JCL in the SRSJUCPY member of the Smart/RESTART control library (the dataset whose low level qualifier is DCACNTL).

```

// SET IN=?source_sequential_dataset_name?
// SET OUT=?target_sequential_dataset_name?
// SET LRECL=?lrecl?
// SET RECFM=?recfm?
/*
/*      Step 1: Copy one blocked physical sequential (PS) dataset
/*      into another and PRESERVE in the target dataset any short
/*      short blocks that may exist within the source dataset
/*
//COPY      EXEC PGM=IEBGENER                                (1)
//SYSPRINT DD  SYSOUT=*
//SYSUT1 DD  DISP=OLD,DCB=RECFM=U,DSN=&IN
//SYSUT2 DD  DISP=OLD,DCB=RECFM=U,DSN=&OUT
//SYSIN DD  DUMMY
/*
/*      Step 2: Reset the LRECL and RECFM DCB attributes to their
/*      original values
/*
//RECFM      EXEC PGM=IEBGENER                                (2)
//SYSPRINT DD  SYSOUT=*
//SYSUT1 DD  DUMMY,DCB=(LRECL=&LRECL,RECFM=&RECFM)
//SYSUT2 DD  DISP=MOD,DSN=&OUT,DCB=(LRECL=&LRECL,RECFM=&RECFM)
//SYSIN DD  DUMMY

```

Figure 4.10 SRSJUCPY – Copy RSAM dataset without reblocking

- (1) Step 1 copies the source file to a larger dataset using IEBGENER, while specifying RECFM=U as the record format for both input and output datasets. RECFM=U causes the physical records to be copied as they exist, without reblocking. The copy must be to a like device type (with the same track size). If the dataset resides on multiple volumes, only the last volume of data can be copied. RSAM maintains position within multiple volumes by way of a relative volume sequence number. Smart/RESTART records the current volume sequence number associated with a repositionable file during each checkpoint. Should a restart be necessary, Smart/RESTART repositions a file that spans multiple volumes directly to the volume sequence number corresponding to the last successful, physical checkpoint. Therefore, when extending a multi-volume dataset, the last committed record must reside on the **same** relative volume sequence number. In addition, the offset to this record from the beginning of the volume should remain the same.
- (2) Step 2 sets the LRECL and RECFM parameters of the copied file back to their original (and proper) values. As such, the SYSUT1 DD statement must be a dummy data set, and the SYSUT2 DD statement must reference the target dataset to which source data was copied in the previous step. The SYSUT2 DD statement must specify DISP=MOD. This second step causes IEBGENER to open the dataset for output but no records are copied. Instead, IEBGENER rewrites the DSCB with updated RECFM and LRECL values when the dataset is closed.

If the relative position of the last committed record is altered for any reason, it will interfere with (and prevent) a successful restart. This can occur for a variety of reasons that include the following:

- the two step IEBGENER procedure described above cannot be followed
- the above procedure was ignored (an operational error)
- an RSAM file was moved or copied to another device type with a *different* track size

In such cases, use Smart/RESTART's REPOSITION_REFRESH command to correctly reposition these files by reading records sequentially to the last commit point. Note that use of REPOSITION_REFRESH will slow application restart somewhat. The performance penalty depends on the number of committed records that need to be reread to reach the proper position. See the Smart/RESTART Reference manual for a detailed description of the REPOSITION_REFRESH command.

Smart/RESTART issues either an error or warning message whenever a dataset is renamed or is allocated on a different volume.

4.6 The SRSUTIL Utility

The Smart/RESTART Utility program SRSUTIL is introduced to invoke Smart/RESTART Utility services. These utility services can be invoked as an independent jobstep as well as via dynamic and static calls issued from an application program that is restart enabled with Smart/RESTART.

NOTE: *SRSUTIL replaces the utility program named SRSUCHK. SRSUCHK is no longer the preferred method to invoke the Smart/RESTART Utility services but is retained for backward compatibility.*

SRSUTIL is a standalone multi-purpose Smart/RESTART utility with which you can perform various RESTART related tasks. as well as supports both The checkpoint file that the SRSUTIL program requires may be allocated via an explicit SRSCHECK DD statement coded in the JCL. In addition, SRSUTIL supports dynamic allocation of the checkpoint file.

Members SRSJUCHK, SRSJULRS and SRSJUMSG of the DCACNTL library contain JCL that can be adapted to run SRSUTIL in the context of your restartable jobstreams. These members include detailed SRSUTIL usage notes along with descriptions of SRSUTIL invocation parameters.

The sample program named SRS8CUTL demonstrates how to invoke SRSUTIL services from within an application program. Members SRS8CUTL, SRSBCUTL and SRSRCUTL of the DCASAMP library respectively contain the source of the SRS8CUTL sample program source, as well as JCL with which to build and execute the sample program.

4.6.1 SRSUTIL invocation parameters

The mode of operation of the SRSUTIL program are governed via invocation parameters which may be specified via the JCL EXEC PGM PARM= statement as in:

```
//SRSUTIL EXEC PGM=SRSUTIL,REGION=OM,PARM='ABEND'
```

4.6.1.1 ABEND

The ABEND parameter directs SRSUTIL to trigger a user abend whenever it finds *residual* RESTART information for the current job ID in the checkpoint file.

A batch job which fails cannot simply be restarted from the first job step. Rather, such a job must be restarted from the job step which failed. SRSUTIL prevents a job from executing accidentally when the RESTART=stepname parameter of the JCL JOB statement is missing.

SRSUTIL can be invoked (via a JCL EXEC statement) as the first step in a job. SRSUTIL will abend to prevent the execution of a failed job that is incorrectly restarted from its first step. Conversely, SRSUTIL ends normally during an initial run to let the execution of the job proceed normally.

4.6.1.2 CLEANUP

The CLEANUP parameter directs SRSUTIL to remove any filler records left in an application's repositionable files when the restartable jobstep terminates abnormally. The SRSUTIL utility can optionally delete uncommitted records as well. Specify the Smart/RESTART parameter INFLIGHT_RECORD(REMOVE) to delete all uncommitted records in all repositionable files.

4.6.1.3 IGNORE

The IGNORE parameter directs SRSUTIL to set the jobstep return code to 0 -- whether or not any residual RESTART information for the current job ID is found in the checkpoint file.

4.6.1.4 RC

The RC parameter directs SRSUTIL to set the jobstep return code to 8 whenever it finds residual RESTART information for the current job ID in the checkpoint file. Conversely, SRSUTIL sets the jobstep return code to zero during an initial run of the job.

The RC parameter is used in a similar fashion as the ABEND parameter. However, any jobstep following the SRSUTIL step must be cognizant of a possible non-zero return code and respond properly in accordance with the jobstep return code set by SRSUTIL. See the description of the ABEND parameter for detailed information about when PARM='RC' and PARM='ABEND' are appropriate.

4.6.1.5 RESET

The RESET parameter directs SRSUTIL to *clear* any residual RESTART information for the current job ID from the checkpoint file. Using PARM='RESET' forces SRSUTIL to *erase* RESTART information from the target checkpoint file. No RESTART is possible for the current job ID once its checkpoint information is erased.

4.6.1.6 ENVCHECK

The ENVCHECK parameter directs SRSUTIL to set the jobstep return code to zero (0) whenever the Smart/RESTART environment can be initialized successfully with the current run-time parameters, default settings, checkpoint medium and status of the Smart/RESTART resource manager synchronization row.

4.6.1.7 DUMP

The DUMP parameter directs SRSUTIL to print the records of the Smart/RESTART checkpoint file. This parameter should only be used at the direction of RAI technical support.

4.6.1.8 LOCATE_SERVER=<rai_server_name>

The LOCATE_SERVER parameter directs SRSUTIL to check whether or not the target RAI Server is operational. The SRSUTIL LOCATE_SERVER service can be invoked either through JCL as an independent jobstep, or via a dynamic call from within a user program.

The locate server service of the SRSUTIL utility sets the return code to zero (0) whenever the target RAI Server is active. Conversely, the locate server utility service sets the return code to four whenever the specified RAI Server cannot be located.

Member SRSJULRS of the DCACNTL library contains JCL that can be adapted to invoke the LOCATE_SERVER utility service as an independent jobstep:

```
//Jobname JOB . . .
// SET RAIHLQ=?raihlq?,
//     SERVER=RAI0
//FINDRSV EXEC PGM=SRSUTIL,REGION=0M,
// PARM='LOCATE_SERVER=&SERVER'
//STEPLIB DD DISP=SHR,DSN=&RAIHLQ..DCALOAD
//SRSPRINT DD SYSOUT=*
```

Figure 4.11 SRSJULRS – Locate Server jobstep

The parameter LOCATE_SERVER=RAI0 directs SRSUTIL to check whether or not the RAI Server named RAI0 is up and running. SRSUTIL sets the jobstep return code to zero (0) whenever the RAI Server named RAI0 is active. Conversely, SRSUTIL sets the return code to four whenever the RAI Server named RAI0 cannot be located.

The SRSUTIL LOCATE_SERVER service can also be invoked from a user written program via a dynamic or static call. Member SRS8CUTL of the DCASAMP library demonstrates how to invoke SRSUTIL from a COBOL sample program to check whether or not the RAI Server named RAI0 is active. The members SRSBCUTL and SRSRCUTL contain JCL with which to respectively compile and run the SRS8CUTL sample program.

4.6.1.9 COPY

The COPY function of SRSUTIL is now obsolete and superceded by the 2 step IEBGENER job described and illustrated in Section 4.5. The COPY parameter of SRSUTIL is disabled and attempts to specify it will produce an error message. Use the SRSJUCPY procedure described in Section 4.5 instead.

4.6.1.10 PRINT_MESSAGE

The PRINT_MESSAGE parameter directs SRSUTIL to print the text of one or more Smart/RESTART messages that may be issued while executing a restartable application jobstep.

Member SRSJUMSG of the DCACNTL library contains JCL that can be adapted to print various Smart Jobstream Series message texts, as illustrated in Figure 4.13 below:

```
//Jobname JOB . . .
// SET RAIHLQ=?raihlq?,
//     LEVEL=A, {TIVIDE|WII|A}
//     PREFIX='DCA,DCRA,RAI,RRFB,SRR,SRSA,SRSC,SRSH,SRSI,SRSJ,SRSL,SRSO+
//             ,SRSQ,SRSR,SRST,SRSV'
//MESSAGES EXEC PGM=SRSUTIL,REGION=0M,
// PARM='PRINT_MESSAGE,LEVEL=&LEVEL,&PREFIX'
//STEPLIB DD DISP=SHR,DSN=&RAIHLQ..DCALOAD
//SRSPRINT DD SYSOUT=*
```

Figure 4.12 SRSJUMSG – Issue and Print Smart/RESTART messages

4.7 The SRSULS Utility

NOTE: The *LOCATE_SERVER* parameter of SRSUTIL (documented in Section 4.6.1.8) replaces the SRSULS program. SRSULS is no longer the preferred method to invoke the Smart/RESTART Utility service *LOCATE_SERVER=<rai_server_name>* but is retained for backward compatibility.

4.8 Releasing Unused Space

The SRSJRLSE member of the DCACNTL library can be used to release the unused space allocated to a repositionable, output dataset. SRSJRLSE makes use of the standard IBM IEBGENER utility.

```
// SET DSNAME=?sequential_dataset_name?
/*_____
//
/*      Release space allocated to an output data set, but not used
/*_____
//RLSE   EXEC PGM=IEBGENER
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  DUMMY,DCB=&DSNAME
//SYSUT2   DD  DISP=MOD,DSN=&DSNAME,SPACE=(, ,0),RLSE)
//SYSIN    DD  DUMMY
```

Figure 4.13 SRSJRLSE – Release unused space

Once a restartable job step completes successfully, a SRSJRLSE step can be executed to release the unused space in a repositionable output dataset created by the preceding restartable application.

NOTE: Do not specify the RLSE sub-parameter on the DD statement which allocates the repositionable output dataset of the restartable job step. The RLSE keyword will trigger the release of the dataset's primary allocation. Should the restartable jobstep terminate abnormally, a subsequent restart run is more likely to ABEND with an x37 completion code if the dataset's secondary allocation is insufficient.

4.9 Deleting Checkpoint Data with the SRSUDEL Utility

The new Smart/RESTART Delete Utility (program SRSUDEL) provides a means to delete selected Smart/RESTART checkpoint datasets and corresponding rows from the Checkpoint Synchronization table maintained in DB2 – all in a single execution of the utility.

The SRSJUDEL member of the DCACNTL library (the dataset whose low level qualifier is DCACNTL) contains prototype JCL with which to execute the utility. The JCL must be modified before it can be submitted for execution. This can be accomplished manually as described in steps (1) through (3) embedded in the JCL. Alternatively, make the changes by invoking the SJSMEDIT Edit macro from the EDIT command line. Documentation within the ISPF Edit macro SJSMEDIT describe how to customize the exec and invoke it as an Edit macro. SJSMEDIT is a member of the DCAEXEC library.

Figure 4.14 below shows the JCL with which to run the SRSUDEL utility.

```
//Jobname JOB
// SET RAIHLQ=?raihlq?,
//     DB2HLQ=?db2hlq?,
//     DSN=?dsn?,
//     DCAVRM=?dcavrm?
// *
// *          Execute the Smart/RESTART DELETE utility
// *
//SRSDELET EXEC PGM=SRSUDEL,DYNAMNBR=10,REGION=0M,
//          PARM='DB2=?dsn?'
//STEPLIB DD DISP=SHR,DSN=&DB2HLQ..SDSNEXIT
//          DD DISP=SHR,DSN=&DB2HLQ..SDSNLOAD
//          DD DISP=SHR,DSN=&RAIHLQ..DCALOAD
//ISPLIB DD DISP=SHR,DSN=&RAIHLQ..DCAMLIB
//SYSTSPRT DD SYSOUT=*
//SRSPRINT DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//SRSINPUT DD *
//          PLAN=DCA?dcavrm?C
// *
//SQLINPUT DD *
-- Delete Smart/RESTART checkpoint datasets and associated DB2 rows for
-- jobs that ran today and failed with system completion code 0C7

DELETE FROM DCA?dcavrm?.CKPT_SYNC
        WHERE DAYS(CURRENT TIMESTAMP) - DAYS(CKPT_START_DATE) = 0
        AND CKPT_TERM_CODE = 'S0C7'
/
```

Figure 4.14 SRSJUDEL – JCL with which to run the SRSUDEL utility

Parameters which govern the execution of the SRSUDEL utility can be specified via the PARM field of the JCL EXEC statement or be specified in an SRSINPUT file.

The SQL DELETE statement is specified in the SQLINPUT file. The utility does *not* execute the SQL DELETE statement as specified. Instead, SRSUDEL uses the WHERE clause of the DELETE statement to fetch rows from the DB2 Checkpoint Synchronization table. Each fetched row in turn identifies the name of the corresponding Smart/RESTART checkpoint dataset. Next, for each row, the utility deletes the dataset and the DB2 row.

NOTE 1: The SQLINPUT stream can contain comments which are denoted by a dash – in columns 1

NOTE 2: the WHERE clause of the SQL DELETE statement is required.

4.9.1 SRSUDEL invocation parameters

The parameters which govern the operation the SRSUDEL utility are documented below in alphabetical order:

COMMIT={SINGLE|GLOBAL|OFF}

SINGLE Issue one SQL COMMIT request after each DB2 row is deleted.

GLOBAL Issue one SQL COMMIT request after all DB2 rows are deleted.

OFF Do not issue any SQL COMMIT requests. The DB2 changes will be committed when the utility completes successfully

The default value is SINGLE.

DB2=?dsn?

The DB2 parameter identifies the name of the target DB2 subsystem. This parameter is required and has no default.

DELETE_DSN={ON|OFF}

ON Delete the Smart/RESTART checkpoint files whose dataset names were resolved by the WHERE clause of the SQL DELETE statement specified in the SQLINPUT file.

OFF Do not delete any checkpoint dataset.

The default value is ON.

DELETE_ROW={ONLY|EVEN|OFF}

- ONLY** Delete a DB2 row when any of the following three conditions are met:
- (a) Delete any DB2 row that corresponds to a checkpoint dataset the utility has deleted successfully
 - (b) the DELETE_DSN command is set to OFF
 - (c) No checkpoint dataset exists for the row because all checkpoint information is saved in DB2 (as occurs when the BULK_CKPT(DB2) setting is in force).
- EVEN** In addition to conditions (a), (b) and (c) described above, delete any DB2 row that corresponds to a checkpoint dataset that this utility failed to delete successfully. Successful deletion is indicated by a zero return code. However, a non-zero return code may or may not indicate that the dataset is deleted.
- OFF** Do not delete any DB2 row.

The default value is ONLY.

MSGDD=SRSPRINT

The MSGDD parameter specifies the name of the file to which informational, warning and error messages issued by the Smart/RESTART Delete Utility will be written.

The default value is SRSPRINT.

PLAN=DCA?dcavrm?C

The PLAN parameter specifies the name of the DB2 application plan that the Smart/RESTART Delete Utility will use. The utility makes use of the same DB2 plan name as the Smart/MONITOR ISPF dialog.

This parameter is required.

TRACE={OFF|C|E|F|I|L|N|O|R|S|}

The TRACE parameter controls the Smart/RESTART Delete Utility tracing level. See the IBM TSO REXX Reference for a detailed description of these trace level actions. This parameter should only be used for debugging purposes as instructed by Smart/RESTART support representative.

The default value is OFF.

Restart Enablement via COPY / INCLUDE Code

This chapter describes and illustrates how applications can be enabled for restartable operation via COBOL COPY members or PL/I INCLUDE text. Some organizations may prefer the COPY/INCLUDE technique to the Precompiler Transparency approach discussed in Chapter 2.

Smart/PRECOMPILER runs as a discrete step in a program preparation job to automatically add the data structures and procedural calls required to restart enable your applications. Manually coding COBOL COPY members or PL/I INCLUDE text in your applications accomplishes the same thing – without the need to change existing program preparation procedures.

NOTE: *RAI recommends you use this COPY/INCLUDE technique to restart enable just the main module in a multi-module application. The main module **must** be restart enabled, even if it has no storage areas to save.*

The choice of restart enablement method (Smart/PRECOMPILER or COPY / INCLUDE text) can be made on a module by module basis. It need not be a single, site-wide choice for all applications because RAI recommends both techniques with equal weight.

5.1 COBOL COPY member Example

Figure 5.1 shows a sample COBOL program that includes the COPY members necessary to run a program restartably with Smart/RESTART. The first two COPY members add data structures to Working Storage while the last two invoke calls to Smart/RESTART that indicate the initiation and termination of the program.

In contrast to the ‘source transparent’ program in Chapter 2, the code of the sample program makes **explicit** reference to Smart/RESTART data structures and services (by way of COBOL COPY statements). In Figure 5.1, the bold, italic numbers within parentheses correspond to the numbered annotations which follow the sample code.

The Smart/RESTART dataset whose low level qualifier is DCASAMP contains the sample program named SRS8CCB as well as the COPY / INCLUDE members whose names start with the Smart/RESTART component code 'SRS'. The fourth character of the member name denotes the *host language* in which your application is coded. For example, C denotes a COBOL copy member, P denotes PL/I, @ stands for Assembly language and H designates a C header file.

```

ID DIVISION.
PROGRAM-ID.    SRS8CCB.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-3090.
OBJECT-COMPUTER. IBM-3090.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT OUTPUT-FILE ASSIGN TO OUTFILE
    FILE STATUS IS OUTPUT-FILE-STATUS.

DATA DIVISION.
FILE SECTION.

FD  OUTPUT-FILE
    LABEL RECORDS STANDARD
    RECORDING MODE F
    BLOCK CONTAINS 0 RECORDS.
01  OUTPUT-RECORD                      PIC X(80).

WORKING-STORAGE SECTION.

    COPY SRSCDCL1.                                     <=== (1)

01  VITAL-STORAGE.
    05  UOW-COUNT                      PIC S9(4) COMP VALUE 0.
    05  OUTPUT-COUNT                  PIC S9(4) COMP VALUE 0.

    COPY SRSCDCL2.                                     <=== (2)

01  MISCELLANEOUS-VARIABLES.
    05  ERROR-FLAG                    PIC X(1) VALUE SPACE.
        88  API-ERROR                  VALUE 'A'.
        88  IO-ERROR                   VALUE 'I'.
        88  SQL-ERROR                   VALUE 'S'.
        88  PROCESSING-ERROR            VALUES ARE 'I' 'S'.
    05  QUERY-FLAG                    PIC X(1) VALUE SPACE.
        88  END-OF-QUERY                 VALUE 'E'.
    05  PGM-RETURN-CODE                PIC S9(4) COMP.
    05  DISPLAY-SQLCODE                PIC -9(9).

01  NON-CRITICAL-FIELDS.
    05  CHARACTER-FIELD                PIC X(04) VALUE SPACES.
    05  NUMERIC-FIELD                  REDEFINES
        CHARACTER-FIELD                PIC S9(4).
    05  OUTPUT-FILE-STATUS              PIC 9(02) VALUE ZEROS.
    05  RUN-SEQUENCE-COUNT              PIC 9(07) VALUE ZEROS.
    05  RUN-FLAG                        PIC X(01) VALUE 'N'.
        88  NORMAL-RUN                  VALUE 'N'.
        88  RESTART-RUN                  VALUE 'R'.
    05  PGM-NAME                        PIC X(08) VALUE 'SRS8CCB'.
    05  CURRENT-IO                      PIC X(08) VALUE ZEROS.

```



```

            88 IO-READ                VALUE 'READ'.
            88 IO-WRITE              VALUE 'WRITE'.
            88 IO-OPEN               VALUE 'OPEN'.
            88 IO-CLOSE              VALUE 'CLOSE'.
*
*   Define SQL host variables
*
01 HOST-VARIABLES.
   05 HV-COLNO-GROUP.
       15 HV-COLNO-TYPE    PIC X.
       15 HV-COLNO-SEQ    PIC 9(3).
   05 HV-COLNO             REDEFINES
       HV-COLNO-GROUP     PIC S9(5) COMP.
   05 FILLER               PIC X(05) VALUE SPACES.
   05 HV-NAME              PIC X(18).
   05 FILLER               PIC X(05) VALUE SPACES.
   05 HV-TBNAME            PIC X(18).
   05 HV-PGM-NAME          REDEFINES
       HV-TBNAME          PIC X(18).
   05 FILLER               PIC X(05) VALUE SPACES.
   05 HV-TBCREATOR         PIC X(8).
   05 FILLER               PIC X(05) VALUE SPACES.
   05 HV-COLTYPE           PIC X(8).
   05 FILLER               PIC X(4)  VALUE SPACES.
*
*   Declare SQL related entities
*
EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC SQL DECLARE CSR CURSOR FOR
    SELECT COLNO, NAME, TBNAME, TBCREATOR, COLTYPE
    FROM SYSIBM.SYSCOLUMNS
    WHERE TBNAME = 'SYSCOLUMNS'
    AND TBCREATOR = 'SYSIBM'
    AND COLNO >= :HV-COLNO
    ORDER BY COLNO
END-EXEC.
/
PROCEDURE DIVISION.

    COPY SRSCINIT.                                     <=== (3)

    IF RETURN-CODE = 2001 THEN
        DISPLAY 'SRS8CCB detected the RESTART environment'
        MOVE 'R' TO RUN-FLAG
    ELSE
        IF RETURN-CODE = ZERO
            MOVE 'N' TO RUN-FLAG
        ELSE
            DISPLAY 'SRS8CCB initialization error detected'
            GOBACK.

*   The flow of control for either the original or restart
*   run is substantially the same.

    PERFORM A100-INITIALIZE THRU A100-EXIT.
    PERFORM B100-UNIT-OF-WORK THRU B100-EXIT
        UNTIL UOW-COUNT > 5
        OR    PROCESSING-ERROR.

    PERFORM C100-TERMINATE THRU C100-EXIT.

    DISPLAY 'SRS8CCB processing completed successfully'.
    MOVE PGM-RETURN-CODE TO RETURN-CODE.

*   Signal to Smart/RESTART that SRS8CCB is about to

```

```

*      terminate its processing:

      COPY SRSCBACK.                                     <=== (4)
      .                                                 <=== (5)
      GOBACK.

*-----
*
*      Smart/RESTART automatically repositions those sequential
*      files whose position is monitored. Repositionable input
*      and output files are defined external to the application,
*      for example instream with the JCL.
*-----
A100-INITIALIZE.

      MOVE 'OPEN' TO CURRENT-IO.
      OPEN OUTPUT OUTPUT-FILE.
      IF OUTPUT-FILE-STATUS > 0 THEN
        PERFORM B400-OUTPUT-ERROR THRU B400-EXIT.

A100-EXIT.
      EXIT.

*-----
*
*      Each Logical Unit of Work fetches each row of the SQL query
*      result before taking a checkpoint.
*-----
B100-UNIT-OF-WORK.
      ADD 1 TO UOW-COUNT.

      DISPLAY 'Commencing new logical unit of work ' UOW-COUNT.

      EXEC SQL OPEN CSR END-EXEC.
      IF SQLCODE NOT = 0 THEN
        PERFORM B300-SQL-ERROR THRU B300-EXIT
        GO TO B100-EXIT.

      MOVE SPACE TO QUERY-FLAG.

*      Fetch and process each row of the SQL query result.
*      This represents our individual transaction.

      PERFORM B200-FETCH-ROW THRU B200-EXIT
        UNTIL END-OF-QUERY
        OR    PROCESSING-ERROR.

      EXEC SQL CLOSE CSR END-EXEC.
      IF SQLCODE NOT = 0 THEN
        PERFORM B300-SQL-ERROR THRU B300-EXIT
        GO TO B100-EXIT.

*
*      Commit all completed transactions as a single Unit of Work
*
*      Note: In the Smart/RESTART - Smart/RRSAF environment this
*      standard EXEC SQL COMMIT causes application processing to
*      be checkpointed automatically along with DB2 updates.
*
      EXEC SQL COMMIT WORK END-EXEC.
      IF SQLCODE NOT = 0 THEN
        PERFORM B300-SQL-ERROR THRU B300-EXIT
        GO TO B100-EXIT.
B100-EXIT.
      EXIT.

```

```

B200-FETCH-ROW.
  EXEC SQL FETCH CSR INTO
    :HV-COLNO,
    :HV-NAME,
    :HV-TBNAME,
    :HV-TBCREATOR,
    :HV-COLTYPE
  END-EXEC.

  IF SQLCODE = 100 THEN
    MOVE 'E' TO QUERY-FLAG
    GO TO B200-EXIT.

  IF SQLCODE < 0 THEN
    PERFORM B300-SQL-ERROR THRU B300-EXIT
    GO TO B200-EXIT.

  ADD 1 TO RUN-SEQUENCE-COUNT.

  MOVE RUN-SEQUENCE-COUNT TO HV-COLNO-SEQ.
  MOVE RUN-FLAG          TO HV-COLNO-TYPE.
  MOVE PGM-NAME           TO HV-PGM-NAME.

  MOVE HOST-VARIABLES     TO OUTPUT-RECORD.

  MOVE 'WRITE' TO CURRENT-IO.
  WRITE OUTPUT-RECORD.

  IF OUTPUT-FILE-STATUS > 0 THEN
    PERFORM B400-OUTPUT-ERROR THRU B400-EXIT
    GO TO B200-EXIT.

  ADD 1 TO OUTPUT-COUNT.
  IF NORMAL-RUN AND OUTPUT-COUNT = 68
    DISPLAY 'SRS8CCB deliberately abending with SOC7'

*      This statement will produce a deliberate SOC7 abend

      ADD 1 TO NUMERIC-FIELD.
B200-EXIT.
  EXIT.

B300-SQL-ERROR.
  DISPLAY 'Error on SQL call'.
  DISPLAY 'SQL Communications area'
  MOVE SQLCODE TO DISPLAY-SQLCODE.
  DISPLAY 'SQLCODE = ' DISPLAY-SQLCODE.
  DISPLAY 'SQLCA =' SQLCA.
  MOVE 'S' TO ERROR-FLAG.
  MOVE 8 TO PGM-RETURN-CODE.
B300-EXIT.
  EXIT.

B400-OUTPUT-ERROR.
  DISPLAY 'Error on output service while performing '
    CURRENT-IO
    ' STATEMENT'.

  DISPLAY 'Output file status item = ' OUTPUT-FILE-STATUS.

  MOVE 'I' TO ERROR-FLAG.
  MOVE 12 TO PGM-RETURN-CODE.
B400-EXIT.
  EXIT.

```

```

C100-TERMINATE.
  MOVE 'CLOSE' TO CURRENT-IO.
  CLOSE OUTPUT-FILE.
  IF OUTPUT-FILE-STATUS > 0 THEN
    PERFORM B400-OUTPUT-ERROR THRU B400-EXIT.
C100-EXIT.
EXIT.

```

where

- (1) Copy member SRSCDCL1 denotes the beginning of Working Storage fields that Smart/RESTART should save at checkpoint time and restore at restart or rollback time. It also expands the data structures Smart/RESTART needs to restart enable your application. Section 5.2 describes the SRSCDCL1 COPY member in greater detail.

NOTE: The use of COBOL COPY members or PL/I INCLUDEs is *not* suitable when multiple, non-contiguous ranges of application storage need to be saved at checkpoint time and restored in the event of rollback or restart. Smart/RESTART supports such requirements via Smart/PRECOMPILER or the call level API as described in Section 1 of Chapter 50.

- (2) COPY member SRSCDCL2 identifies the end of the storage areas Smart/RESTART should save at checkpoint time and restore at RESTART and ROLLBACK time.
- (3) Code the COPY SRSCINIT statement after the PROCEDURE DIVISION header to initialize the Smart/RESTART environment. This COPY code identifies the address and length of the storage area(s) to be saved at checkpoint time and restored at restart or rollback time. It also sets flags which enable your application to distinguish between an initial and a restart run (as described in Section 5.3)
- (4) Code the COPY SRSCBACK statement before a GOBACK statement to introduce a call to the Smart/RESTART TERMINATE service to signal normal module termination. When a *main module* calls the TERMINATE service, it informs Smart/RESTART that the application has completed its processing successfully and no restart is required. In contrast, the restart environment remains active when a *subprogram* calls the TERMINATE service. The subprogram simply returns to its caller.

NOTE: COBOL programs that end with a GOBACK statement should add a COPY SRSCBACK statement just before the GOBACK (as illustrated in this SRS8CCB sample program). In contrast, programs that end with a STOP RUN statement should code a COPY SRSCSTOP statement just prior to the STOP RUN.

- (5) The period is **required** when there is no IF test. Both SRSCBACK and SRSCSTOP can be coded *inside* an IF statement and therefore *do not end with a period*. When a COPY SRSCBACK or COPY SRSCSTOP statement is coded standalone (as in Figure 5.1), code a period on a line by itself -- *following* the COPY statement.

Figure 5.1 Restart enablement using COBOL COPY members

5.2 Identifying Critical Working Storage to Smart/RESTART

The COBOL COPY members SRSCDCL1 and SRSDCL2 allow you to identify to Smart/RESTART those storage areas within your program which are critical to your application's processing. Smart/RESTART saves the working storage structures defined between the SRSCDCL1 and SRSDCL2 COPY statements. This occurs each time your application requests a checkpoint (either explicitly through the Smart/RESTART checkpoint service or implicitly via an EXEC SQL COMMIT request or call to SRRRCMIT). During application restart, Smart/RESTART restores storage contents and resets sequential file position in synch with the point of consistency maintained by RRS compliant resources managers such as DB2, MQ and/or IMS.

However, not all applications require you to checkpoint storage in its entirety. During application design, you should identify which storage areas should be checkpointed in accordance with the principle *less is more*. That's because it's more efficient to checkpoint only those vital fields (flags, counters, key values and the like) whose values must *persist* beyond failure and be *restored* on restart.

For example, it is not necessary to checkpoint storage areas (such as the HOST-VARIABLES structure in Figure 5.1) which are constantly being overwritten. Nor is it appropriate to checkpoint large storage areas whose values are set during program initialization and never change. An in-storage table loaded from an external file is a good example of stable data which need not be checkpointed. Typically, it's more efficient to *reload* such a table rather than checkpoint it.

NOTE: *The use of COBOL COPY members or PL/I INCLUDEs is not suitable when multiple, non-contiguous ranges of application storage need to be saved at checkpoint time and restored in the event of rollback or restart. Smart/RESTART supports such requirements via either Smart/PRECOMPILER or the call level API described in section 1 of Chapter 50.*

5.3 Distinguishing between Initial and Restart Runs

Your application may need to distinguish between an initial run and a restart run. The FCA data structure (introduced via the COPY SRSCDCL1 statement) allows your application and Smart/RESTART to communicate. The INITIALIZE service (called by code introduced via the COPY SRSCINIT statement) sets the value of a field named FCA-STATUS. A value of spaces for the FCA-STATUS field denotes an initial run while a value of 'REST' indicates an application restart. In addition, Smart/RESTART defines the following two COBOL condition names through which you can refer to an original or restart run respectively:

```
FCA-INITIAL-RUN  VALUE SPACES.  
FCA-RESTART-RUN  VALUE 'REST'.
```

NOTE: *Appendix B lists the FCA and other Smart/RESTART data structures in each of the languages Smart/RESTART supports. The Smart/RESTART Reference provides detailed descriptions of these data structures and the fields which comprise them.*

The COBOL code fragment in Figure 5.2 references the COBOL condition name FCA-RESTART-RUN. The program displays a different message under in the restart environment to distinguish it from an initial run.

```
A100-INITIALIZE.  
  IF FCA-RESTART-RUN THEN                                <=== (1)  
    DISPLAY 'THIS IS A RESTART RUN OF THE EXAMPLE PROGRAM'  
  ELSE  
    DISPLAY 'THIS IS THE INITIAL RUN OF THE EXAMPLE PROGRAM'.
```

where

(1) Code makes reference to the condition name FCA-RESTART-RUN

Figure 5.2 *Distinguish between initial and restart environments*

5.4 COBOL COPY Members

The following are some of the COBOL COPY members in the Smart/RESTART sample library whose low level qualifier is DCASAMP:

SRSCAPI	Service API names
SRSCBACK	GOBACK service
SRSCCKPT	CHECKPOINT service
SRSCDCL1	Working-Storage Begin
SRSCDCL2	Working-Storage End
SRSCEDA	Environment Description Area
SRSCENVC	ENVCHECK service
SRSCINIT	INITIALIZE service
SRSCRCHK	Check current run type
SRSCROLL	ROLLBACK service
SRSCSTOP	STOP RUN service
SRSCTERM	TERMINATE service

5.5 PL/I Support

PL/I programs can be restart enabled using INCLUDE text distributed in the Smart/RESTART sample library whose low level qualifier is DCASAMP. The SRS8P3 member of the DCASAMP library is a restartable PL/I application that makes use of Smart/RESTART INCLUDE text.

Figure 5.3 is a code fragment from the SRS8CCB sample program which makes *explicit* reference to Smart/RESTART data structures and services (by way of PL/I INCLUDE text). In Figure 5.3, the bold, italic numbers within parentheses correspond to the numbered annotations which follow the sample code.

```

DCL
  1 VITAL_AREA1,                               <=== (1)
    2 UOW_COUNT          FIXED      INIT(0),
    2 OUTPUT_COUNT       FIXED      INIT(0);

DCL                                     <=== (2)
  1 VITAL_AREA2,
    2 FETCH_COUNT        PIC'9999'   INIT(0),
    2 OPEN_CSR_COUNT     PIC'9999'   INIT(0),
    2 CLOSE_CSR_COUNT    PIC'9999'   INIT(0),
    2 VITAL_AREA2_END    PIC'9999'   INIT(0);

%DCL SRS_AREAS CHAR;                       <=== (3)
%SRS_AREAS = 'VITAL_AREA1,VITAL_AREA2';    <=== (4)
%INCLUDE SRSPINIT;                         <=== (5)

```

where

- (1) This is the first data structure that Smart/RESTART should save at checkpoint time and restore at restart or rollback time.
- (2) This is the second data structure that Smart/RESTART should save at checkpoint time and restore at restart or rollback time.
- (3) Code this statement exactly as shown to declare a Smart/RESTART field to the PL/I preprocessor.
- (4) Assign to the %SRS_AREAS variable, the names of the data structure(s) that Smart/RESTART should save at checkpoint time and restore at restart and rollback time. The list of data names should be separated by commas and enclosed in quotes. This assignment identifies your program's vital data structures and implicitly registers the program name as well.
- (5) Code the %INCLUDE SRSPINIT statement exactly as shown. The SRSPINIT INCLUDE expands the data structures and executable code required to restart enable your application. SRSPINIT initializes the Smart/RESTART environment when included within a main PL/I program. SRSPINIT can optionally be included in PL/I sub-programs as well to register them with Smart/RESTART.

IMPORTANT REMINDER: The storage to be saved at checkpoint time (and restored during restart and rollback) must be declared *prior* to coding the SRSPINIT %INCLUDE, as illustrated in the code fragment above.

Figure 5.3 *PL/I Program Fragment*

5.6 PL/I INCLUDEs supplied with Smart/RESTART

SRSPAPI	Service API names
SRSPEDA	Environment Description Area
SRSPENV	ENVCHECK service
SRSPFCA	Function Communications Area
SRSPINIT	INITIALIZE service
SRSPROLL	ROLLBACK service
SRSPRCHK	Check current run type
SRSPREST	Explicit restart processing
SRSPTERM	TERMINATE service

The Smart/RESTART EXEC SRS Interface

This chapter describes how to embed EXEC SRS statements in your applications. EXEC SRS is a high level interface to Smart/RESTART services that is analogous to EXEC SQL and EXEC CICS. Recall that at program preparation time, Smart/PRECOMPILER adds the data structures and procedural code to restart enable your application. These compilable statements are introduced *automatically* at program preparation time.

In addition, Smart/PRECOMPILER (like the IBM supplied precompilers for DB2 and CICS) *translates* any EXEC SRS requests it encounters in your source module into compilable source statements. EXEC SRS affords additional function and flexibility beyond that available in the ‘Source Transparency’ mode described in Chapter 2. For example, granular checkpointing affords a more flexible and efficient alternative to saving application storage in its entirety -- as occurs in Transparency Mode.

10.1 EXEC SRS Syntax

Figure 10.1 lists the EXEC SRS services in alphabetical order. Part I of the Smart/RESTART Reference describes each of these services in detail. The services shown in *italics* in Figure 10.1 are discussed in this chapter.

ABEND	ENVCHECK	ROLLIMS
CANCEL	<i>INITIALIZE</i>	SNAP
<i>CHECKPOINT</i>	MESSAGE_LOG	STOP_RUN
CKPT_IMMED	<i>PROFILE</i>	<i>TERMINATE</i>
END_OF_JOB	ROLLBACK	

Figure 10.1 EXEC SRS Services

The syntax for all EXEC SRS statements is as follows:

```
EXEC SRS service_name ending_delimiter
```

where *service_name* specifies one of the services listed in Figure 10.1. In addition, each EXEC SRS statement requires an ending delimiter. In COBOL, each EXEC SRS statement must end with END EXEC -- as in EXEC SRS INITIALIZE END EXEC. The PL/1 delimiter is the semi-colon (';') -- as in EXEC SRS INITIALIZE;

10.2 Sample DB2/COBOL Application

This section presents a complete DB2/COBOL program named EXAMPLE2. There is awareness of Smart/RESTART in the source code of EXAMPLE2 -- in contrast to the source transparent EXAMPLE program in Chapter 2. The bold, italic numbers within parentheses correspond to the numbered annotations which follow the source statements in Figure 10.2.

```
ID DIVISION.
PROGRAM-ID.    EXAMPLE2.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INPUT-FILE ASSIGN TO INFILE
        FILE STATUS IS INPUT-FILE-STATUS.

DATA DIVISION.
FILE SECTION.

FD  INPUT-FILE
01  INPUT-RECORD                PIC X(80).

WORKING-STORAGE SECTION.
EXEC SRS BEGIN SAVE END-EXEC.                                     <=== (1)

01  VITAL-STORAGE.
    05  UOW-COUNT                PIC S9(4) COMP VALUE 1.
    05  TRANSACTION-COUNT        PIC S9(4) COMP VALUE 0.

EXEC SRS END SAVE END-EXEC.                                       <=== (2)

01  HOST-VARIABLES.
    05  HOST-VARIABLE-1          PIC S9(5) COMP.
    05  HOST-VARIABLE-2          PIC S9(5) COMP.
    05  HOST-VARIABLE-3          PIC S9(5) COMP.

EXEC SQL INCLUDE SQLCA END-EXEC.

01  NON-CRITICAL-FIELDS.
    05  PROCESSING-FLAG          PIC X(1) VALUE SPACE.
    88  SUCCESSFUL-PROCESSING    VALUES ARE ' ' 'E'
    88  END-OF-FILE              VALUE 'E'.
    88  IO-ERROR                 VALUES ARE 'I'.
    88  PROCESSING-ERROR         VALUES ARE 'I' 'P'.
    05  INPUT-FILE-STATUS        PIC 9(02) VALUE ZEROS.

PROCEDURE DIVISION.
EXEC SRS INITIALIZE END EXEC.                                     <=== (3)
    PERFORM A100-INITIALIZE THRU A100-EXIT.
    PERFORM B100-UNIT-OF-WORK THRU B100-EXIT
```

```

        UNTIL END-OF-FILE
        OR    PROCESSING-ERROR.
    DISPLAY 'EXAMPLE - PROCESSING COMPLETED'.
    EXEC SRS TERMINATE END EXEC.                                     <=== (4)
    GOBACK.

A100-INITIALIZE.
    IF FCA-RESTART-RUN THEN                                         <=== (5)
        DISPLAY 'THIS IS A RESTART RUN OF PROGRAM EXAMPLE2'
    ELSE
        DISPLAY 'THIS IS THE INITIAL RUN OF PROGRAM EXAMPLE2'.
    OPEN INPUT INPUT-FILE.
    EXEC SRS PROFILE 'CKPT_PACE(10)' END EXEC                       <=== (6)
A100-EXIT.
    EXIT.

B100-UNIT-OF-WORK.
    MOVE 0 TO READ-COUNT.
    PERFORM B200-PROCESS-TRANSACTION THRU B200-EXIT
        UNTIL TRANSACTION-COUNT > 10
        OR    END-OF-FILE OR PROCESSING-ERROR.

    IF SUCCESSFUL-PROCESSING
        EXEC SRS COMMIT WORK END-EXEC                               <=== (7)
        ADD 1 TO UOW-COUNT
    ELSE
        EXEC SRS ROLLBACK WORK END-EXEC.                             <=== (8)
B100-EXIT.
    EXIT.

B200-PROCESS-TRANSACTION.
    READ INPUT-FILE INTO HOST-VARIABLES
        AT END MOVE 'E' TO PROCESSING-FLAG
        GO TO B200-EXIT.
    IF INPUT-FILE-STATUS NOT = '00' THEN
        PERFORM IO-ERROR-ROUTINE
        GO TO B200-EXIT.

    ADD 1 TO TRANSACTION-COUNT.
    EXEC SQL INSERT INTO SOME.TABLE
        VALUES (:HOST-VARIABLE-1, :HOST-VARIABLE-2, :HOST-VARIABLE-3)
    END-EXEC.
    IF SQLCODE < 0 THEN ...
B200-EXIT.
    EXIT.

```

where

- (1) The EXEC SRS BEGIN SAVE statement brackets the beginning of an area of storage that should be saved at checkpoint time and restored at restart time. (Discussed in Section 10.3).
- (2) The EXEC SRS END SAVE statement denotes the end of a storage area that must be saved and restored.
- (3) The EXEC SRS INITIALIZE statement generates code that identifies the storage to be saved at checkpoint time. The INITIALIZE service also sets flags which enable your application to distinguish between an initial run and a restart run. During restart, EXEC SRS INITIALIZE restores those areas of application storage saved at checkpoint time. (Discussed in Section 10.4.)

NOTE: By default, Smart/PRECOMPILER inserts a call to the Smart/RESTART INITIALIZE service immediately following the PROCEDURE DIVISION header of a COBOL module.

- (4) The EXEC SRS TERMINATE statement signals normal module termination. When a *main module* issues an EXEC SRS TERMINATE request, it signals that the application has completed its processing successfully and no restart is required. In contrast, the restart environment remains active when a *subprogram* issues TERMINATE. The subprogram simply returns to its caller. (Discussed in Section 10.6.)

NOTE: By default, Smart/PRECOMPILER inserts a call to the Smart/RESTART TERMINATE service right before the program's exit point -- in this case a COBOL GOBACK statement.

- (5) Smart/PRECOMPILER inserts a data structure that defines the COBOL condition names FCA-INITIAL-RUN and FCA-RESTART-RUN. Your application can reference these condition names to distinguish between an initial run and a restart run and perform different initialization processing if appropriate. (Discussed in Section 10.5.)
- (6) Your application can issue EXEC SRS PROFILE commands to control Smart/RESTART's run-time behavior. For example, the CKPT_PACE parameter governs checkpoint pacing. The profile parameter string can be specified as a literal within quotes -- as in EXEC SRS PROFILE 'CKPT_PACE(10)' END EXEC. Alternatively, the profile request can reference a dataname which contains the parameter string -- as in EXEC SRS PROFILE DATANAME END EXEC.

NOTE: Part II of the Smart/RESTART Reference describes each of the Smart/RESTART profile commands in detail.

- (7) When your application successfully completes a unit-of-work, it should issue an EXEC SRS COMMIT request. The scope of this restartable unit-of-work spans both application processing and resource manager activity. Thus, storage contents and sequential file position are guaranteed to be in synch with changes to 'protected' resources. EXEC SRS COMMIT is suitable for all RRS compliant resources (such as DB2, IMS and MQ) as well as for DB2 attachment facilities other than Smart/RRSAF (Discussed in Section 10.7.)

NOTE: The Smart/RESTART checkpoint service receives control implicitly when your application issues an EXEC SQL COMMIT request **and** Smart/RRSAF is the DB2 attachment facility.

- (8) Your application should issue an EXEC SRS ROLLBACK request when a unit-of-work fails to complete successfully. The EXEC SRS ROLLBACK service ensures that the backout of changes to application storage and sequential file position occur in coordination with the backout of changes to RRS compliant resources (such as those managed by DB2, IMS or WebSphere MQ). (Discussed in Section 10.8.)

NOTE: The Smart/RESTART rollback service receives control implicitly when your application issues an EXEC SQL ROLLBACK request **and** Smart/RRSAF is the DB2 attachment facility.

Figure 10.2 Sample DB2/COBOL Application Program Using EXEC SRS

10.3 What Storage Areas Should be Checkpointed?

By default, Smart/PRECOMPILER generates code to save all application storage at checkpoint time so it can be restored upon restart. However, not all applications require you to checkpoint storage in its entirety. During application design, you should identify which storage areas should be checkpointed in accordance with the principle ***less is more*** because it's more efficient to checkpoint only those vital fields (flags, counters, key fields and the like) whose values must ***persist*** beyond failure and be ***restored*** on restart.

For example, it is not necessary to checkpoint storage areas (such as the INPUT-RECORD in Figure 10.2) which are constantly being overwritten. Nor is it appropriate to checkpoint large storage areas whose values are set during program initialization and never change. An in-storage table loaded from an external file is a good example of stable data which need not be checkpointed. Typically, it's more efficient to *reload* such a table rather than checkpoint it.

NOTE: *Smart/RESTART places no limit on the amount of application storage which can be checkpointed. Moreover, the checkpoint service is extremely efficient, so it's not worth spending a great deal of time to minimize the amount of storage that is checkpointed.*

10.3.1 Examples

The EXEC SRS BEGIN SAVE and EXEC SRS END SAVE statements let you identify your application's critical storage areas. Once identified, Smart/RESTART saves their contents each time an EXEC SQL COMMIT request is issued (or your application explicitly calls the Smart/RESTART checkpoint service). The COBOL source fragment in Figure 10.3 illustrates how to bracket vital storage areas between EXEC SRS BEGIN SAVE and EXEC SRS END SAVE requests.

```
DATA DIVISION.
...
EXEC SRS BEGIN SAVE END-EXEC                                <=== (1)

01 VITAL-STORAGE.                                           <=== (2)
   05 VITAL-COUNTER          PIC S9(5) COMP.
   05 VITAL-SUBSCRIPT        PIC S9(5) COMP.
   05 VITAL-PAGE-NUMBER      PIC S9(5) COMP.
   05 VITAL-POINTER          PIC S9(5) COMP.
   05 VITAL-FLAGS.
      10 VITAL-FLAG1         PIC X(1).
      10 VITAL-FLAG2         PIC X(1).
      10 VITAL-FLAG3         PIC X(1).

77 DATANAME-1                                           <=== (3)
77 DATANAME-2
77 DATANAME-3

EXEC SRS END SAVE END-EXEC                                <=== (4)
```

where

- (1) The EXEC SRS BEGIN SAVE statement denotes the beginning of the storage areas that should be saved at checkpoint time so they can be restored if the application fails and needs to be restarted.
- (2) Several critical fields are grouped under the structure named VITAL-STORAGE.
- (3) Three discrete areas (DATANAME-1, DATANAME-2 and DATANAME-3) should also be saved at checkpoint time and restored at restart time.
- (4) The EXEC SRS END SAVE statement delimits the end of the storage areas to be saved and restored.

Figure 10.3 *Identifying storage to save and restore*

As an alternative to EXEC SRS BEGIN SAVE and EXEC SRS END SAVE requests, you can identify data structures by name -- instream with the JCL used to run Smart/PRECOMPILER. The names of the data structures to be saved at checkpoint time and restored at restart time are specified through the Smart/PRECOMPILER parameters STORAGE and STORAGE_RANGE (as described in Chapter 3). If a module has **no** storage to save, specify STORAGE_RANGE(NONE) as a Smart/PRECOMPILER parameter.

NOTE: *Smart/RESTART permits you to save multiple, discontinuous data areas through the checkpoint service. Sections 50.1.3 and 50.1.4 of Chapter 50 describe and illustrate how to do so with EXEC SRS directives.*

10.3.2 Default Storage Areas

RAI recommends you define the names of one or more data structures as *defaults* when Smart/PRECOMPILER is installed. When Smart/PRECOMPILER encounters one of these data structures, it *automatically* generates code to save and restore the structure and all the fields which comprise it. For example, suppose your site defines VITAL-STORAGE as a default. If you group your application's critical fields under VITAL-STORAGE (as in Figure 10.2), there's no need to code EXEC SRS BEGIN SAVE and EXEC SRS END SAVE statements.

NOTE: *Ask your Smart/RESTART administrator for the names of the default data structures defined at your site.*

10.4 INITIALIZE Service and Dynamically Acquired Storage

Among the first actions your program should perform is to issue an EXEC SRS INITIALIZE statement. If you omit an EXEC SRS INITIALIZE request, Smart/PRECOMPILER adds one on your behalf -- immediately after program entry.

However, any storage that Smart/RESTART is to save at checkpoint time and restore at restart time must be allocated *before* the EXEC SRS INITIALIZE request is executed. For example, the PL/1 ALLOCATE statement dynamically acquires storage. Similarly, a COBOL program can invoke an Assembler Language routine to issue a STORAGE or GETMAIN macro. In such cases, it is necessary to explicitly code an EXEC SRS INITIALIZE statement *after* such dynamic storage is acquired.

10.5 Distinguishing between Initial and Restart Runs

Your application may need to distinguish between an initial and restart run. Smart/PRECOMPILER introduces a data structure called the FCA (Function Communications Area) through which your application and Smart/RESTART communicate. The INITIALIZE service sets the value of a field named FCA-STATUS. A value of spaces for the FCA-STATUS field denotes an initial run while a value of 'REST' indicates an application restart. In addition, Smart/RESTART defines the following two COBOL condition names through which you can refer to an original or restart run respectively:

```
FCA-INITIAL-RUN  VALUE SPACES.  
FCA-RESTART-RUN VALUE 'REST'.
```

NOTE: Appendix B lists the FCA and other Smart/RESTART data structures in each of the languages Smart/RESTART supports. The Smart/RESTART Reference provides detailed descriptions of these data structures and the fields which comprise them.

The COBOL code fragment in Figure 10.4 references the COBOL condition name FCA-RESTART-RUN. The program EXAMPLE2 displays a different message in the restart environment to distinguish it from an initial run.

```
A100-INITIALIZE.  
  IF FCA-RESTART-RUN THEN  
    DISPLAY 'THIS IS A RESTART RUN OF PROGRAM EXAMPLE2' <=== (1)  
  ELSE  
    DISPLAY 'THIS IS THE INITIAL RUN OF PROGRAM EXAMPLE2'.
```

where

(1) Code makes reference to the condition name FCA-RESTART-RUN

Figure 10.4 Distinguish between initial and restart environments

10.6 TERMINATE Service

The last action your program should perform is to issue an EXEC SRS TERMINATE statement to signal normal module termination. Smart/PRECOMPILER adds the EXEC SRS TERMINATE request to each module for which an explicit request is absent.

For modules written in COBOL, an explicitly coded EXEC SRS TERMINATE request should be positioned before any GOBACK or STOP RUN statement. For other languages, the placement of the TERMINATE request should be before any point in the module where control is returned to its caller.

10.7 Checkpointing with Smart/RESTART

Your application should signal the completion of a *restartable-unit-of-work* when it reaches a point of consistency in its processing (with respect to application storage, sequential file position and resource manager updates). This can be accomplished *explicitly* through the EXEC SRS COMMIT statement or *implicitly* via an EXEC SQL COMMIT request.

NOTE: Applications which do not use Smart/RRSAF as a DB2 attachment facility must use EXEC SRS COMMIT instead of EXEC SQL COMMIT. Smart/PRECOMPILER can **automatically** replace EXEC SQL COMMIT with EXEC SRS COMMIT. Specify the Smart/PRECOMPILER parameter CAF(OTHER) to request this automatic conversion.

The code fragment illustrated in Figure 10.5 issues an EXEC SQL COMMIT request when the unit-of-work completes successfully. The Smart/RRSAF component of Smart/RESTART intercepts this COMMIT request and calls the Smart/RESTART CHECKPOINT service. This facility saves an execution-time 'snapshot' of application storage and sequential file position that is guaranteed to be in synch with the DB2 changes made permanent by EXEC SQL COMMIT. Because the scope of this restartable unit-of-work spans both application processing and resource manager activity, it represents a point of consistency from which processing can resume successfully upon restart.

```
B100-UNIT-OF-WORK.  
  MOVE 0 TO READ-COUNT.  
  PERFORM B200-PROCESS-TRANSACTION THRU B200-EXIT  
    UNTIL TRANSACTION-COUNT > 10  
    OR    END-OF-FILE OR PROCESSING-ERROR.  
  
  IF SUCCESSFUL-PROCESSING  
    EXEC SQL COMMIT END-EXEC          <=== Implicit Checkpoint Request  
    ADD 1 TO UOW-COUNT  
  ELSE  
    EXEC SQL ROLLBACK WORK END-EXEC.  
B100-EXIT.  
EXIT.
```

Figure 10.5 *Implicit checkpoint request via EXEC SQL COMMIT*

In the explicit case illustrated in Figure 10.6, Smart/RESTART knows what ‘protected’ resources are accessed by your application and issues a checkpoint request appropriate for that resource manager. In this way, Smart/RESTART guarantees that application processing and resource manager activity succeed or fail as a unit.

```
B100-UNIT-OF-WORK.  
  MOVE 0 TO READ-COUNT.  
  PERFORM B200-PROCESS-TRANSACTION THRU B200-EXIT  
    UNTIL TRANSACTION-COUNT > 10  
    OR    END-OF-FILE OR PROCESSING-ERROR.  
  
  IF SUCCESSFUL-PROCESSING  
    EXEC SRS CHECKPOINT END-EXEC      <=== Explicit Checkpoint Request  
    ADD 1 TO UOW-COUNT  
  ELSE  
    EXEC SRS ROLLBACK WORK END-EXEC.  
B100-EXIT.  
EXIT.
```

Figure 10.6 *Explicit checkpoint request via EXEC SRS CHECKPOINT*

Application storage, sequential file position and resource manager activity are all checkpointed at the same time. Therefore, it is *incorrect* to request a checkpoint through EXEC SRS COMMIT and then issue a separate synchpoint request (such as EXEC SQL COMMIT) to the resource manager.

10.8 Rollback with Smart/RESTART

Your application should signal Smart/RESTART when a unit-of-work *fails* to complete successfully. This can be accomplished *explicitly* through the EXEC SRS ROLLBACK statement or *implicitly* via an EXEC SQL ROLLBACK request. The coordinated back-out facility implemented by Smart/RESTART restores application storage, sequential file position and protected resources (DB2 tables, IMS databases and/or MQ queues) to their last checkpointed state. This represents a point of consistency from which processing can successfully resume.

NOTE: Applications which do not use Smart/RRSAF as a DB2 attachment facility must use EXEC SRS ROLLBACK instead of EXEC SQL ROLLBACK. Smart/PRECOMPILER can **automatically** replace EXEC SQL ROLLBACK with EXEC SRS ROLLBACK. Specify the Smart/PRECOMPILER parameter CAF(OTHER) to request this automatic conversion.

The code fragment illustrated in Figure 10.7 issues an EXEC SQL ROLLBACK request when the unit-of-work fails to complete successfully. The Smart/RRSAF component of Smart/RESTART intercepts the ROLLBACK request and calls Smart/RESTART's more comprehensive ROLLBACK service.

```

B100-UNIT-OF-WORK.
  MOVE 0 TO READ-COUNT.
  PERFORM B200-PROCESS-TRANSACTION THRU B200-EXIT
    UNTIL TRANSACTION-COUNT > 10
    OR    END-OF-FILE OR PROCESSING-ERROR.

  IF SUCCESSFUL-PROCESSING
    EXEC SQL COMMIT WORK END-EXEC
    ADD 1 TO UOW-COUNT
  ELSE
    EXEC SQL ROLLBACK WORK END-EXEC.      <=== implicit ROLLBACK request
B100-EXIT.
EXIT.

```

Figure 10.7 *Implicit backout request via EXEC SQL ROLLBACK*

You should issue an EXEC SRS ROLLBACK request when your application accesses IMS databases, MQ queues or uses an attachment facility other than Smart/RRSAF to access DB2. In the explicit case illustrated in Figure 10.8, Smart/RESTART knows what 'protected' resources are accessed by your application and issues a backout request appropriate for that resource manager.

```

B100-UNIT-OF-WORK.
  MOVE 0 TO READ-COUNT.
  PERFORM B200-PROCESS-TRANSACTION THRU B200-EXIT
    UNTIL TRANSACTION-COUNT > 10
    OR    END-OF-FILE OR PROCESSING-ERROR.

  IF SUCCESSFUL-PROCESSING
    EXEC SRS COMMIT WORK END-EXEC
    ADD 1 TO UOW-COUNT
  ELSE
    EXEC SRS ROLLBACK WORK END-EXEC.      <=== explicit ROLLBACK request
B100-EXIT.
EXIT.

```

Figure 10.8 *Explicit backout request via EXEC SRS ROLLBACK*

Coding Restartable Applications with the Smart/RESTART Call Level Interface

This chapter addresses the design and coding of restartable applications using the Call Level Interface to Smart/RESTART services. Smart/RESTART's callable 'API' affords additional function and flexibility beyond that available in Smart/RESTART's transparent modes of operation. For example, granular checkpointing affords a more efficient alternative to saving working storage in its entirety -- as often occurs in Transparency Mode.

11.1 The Smart/RESTART API

The Smart/RESTART Applications Programming Interface (API) lets you request Smart/RESTART services from any programming language (e.g. COBOL, PL/1, Assembler Language or C/C++) which adheres to OS calling conventions. Programs simply call the API using the calling conventions applicable to that language, as in the following COBOL fragment:

```
CALL 'RAIAPI' USING RAI-CHECKPOINT,  
                  FCA.
```

'RAIAPI' is the name of the subroutine which provides an interface to Smart/RESTART services. The first parameter identifies the Smart/RESTART service being requested while the second parameter identifies the Smart/RESTART **Function Communications Area (FCA)**. The FCA, defined as a structure within your application's storage, lets you pass information to and receive feedback from Smart/RESTART.

The Smart/RESTART dataset whose low level qualifier is DCASAMP contains these structures in members whose names start with the Smart/RESTART component code 'SRS'.

The structure which defines the services available through the Call Level Interface is called SRSxAPI, while the FCA is contained in copy member SRSxFCA. The fourth character of the member name denotes the *host language* in which your application is coded. For example, C denotes a COBOL copy member, P denotes PL/1, @ stands for Assembly language and H designates a C header file.

You can incorporate these data structures into your programs through language-dependent COPY or INCLUDE statements or you can explicitly code them yourself.

The Smart/RESTART Reference provides detailed descriptions of these data structures and the fields which comprise them. Appendix B in this manual simply lists the structures in each of the languages Smart/RESTART supports. Figure 11.1 enumerates some of the Smart/RESTART services available through the Call Level Interface in alphabetical order. Part I of the Smart/RESTART Reference describes each of these services in detail. The services shown in bold, italic type in Figure 11.1 are discussed in this chapter.

ABEND	ENVCHECK	RESTART
ALLOCATE	FILE_INFO	<i>ROLLBACK</i>
CANCEL	FREE	ROLLIMS
<i>CHECKPOINT</i>	<i>INITIALIZE</i>	SNAP
CKPT_IMMED	MESSAGE_LOG	STOP_RUN
END_OF_JOB	<i>PROFILE</i>	<i>TERMINATE</i>

Figure 11.1 *Smart/RESTART Services available through the Call Level Interface*

11.2 Call Level Interface Example

This section presents a fully annotated example of a restartable application program that makes use of Smart/RESTART services available through its Call Level Interface. In contrast to the ‘source transparent’ program in Chapter 2, the source code of the EXAMPLE3 application makes explicit reference to Smart/RESTART data structures and services. In Figure 11.2, the bold, italic numbers within parentheses correspond to the numbered annotations which follow the sample code.

```

ID DIVISION.
PROGRAM-ID.    EXAMPLE3.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INPUT-FILE ASSIGN TO INFILE
        FILE STATUS IS INPUT-FILE-STATUS.

DATA DIVISION.
FILE SECTION.

FD INPUT-FILE
01 INPUT-RECORD                      PIC X(80).

WORKING-STORAGE SECTION.

01 VITAL-STORAGE.                                     <=== (1)
    05 VITAL-COUNTER                      PIC S9(5) COMP.
    05 VITAL-SUBSCRIPT                    PIC S9(5) COMP.
    05 VITAL-PAGE-NUMBER                  PIC S9(5) COMP.
    05 VITAL-POINTER                      PIC S9(5) COMP.
    05 VITAL-FLAGS.
        10 VITAL-FLAG1                    PIC X(1).
        10 VITAL-FLAG2                    PIC X(1).
        10 VITAL-FLAG3                    PIC X(1).

01 VITAL-STORAGE-LENGTH                PIC S9(5) COMP.                                     <=== (2)

    COPY SRSCAPI.                           <=== (3)
    COPY SRSCFCA.                           <=== (4)

01 HOST-VARIABLES.
    05 HOST-VARIABLE-1                    PIC S9(5) COMP.
    05 HOST-VARIABLE-2                    PIC S9(5) COMP.
    05 HOST-VARIABLE-3                    PIC S9(5) COMP.

EXEC SQL INCLUDE SQLCA END-EXEC.

01 NON-CRITICAL-FIELDS.
    05 PROCESSING-FLAG                    PIC X(1) VALUE SPACE.
    88 SUCCESSFUL-PROCESSING              VALUES ARE ' ' 'E'
    88 END-OF-FILE                        VALUE 'E'.
    88 IO-ERROR                           VALUES ARE 'I'.
    88 PROCESSING-ERROR                   VALUES ARE 'I' 'P'.
    05 INPUT-FILE-STATUS                  PIC 9(02) VALUE ZEROS.
    05 PROFILE-COMMAND-BUFFER             PIC X(20) VALUE SPACES.

PROCEDURE DIVISION.
    MOVE LENGTH OF VITAL-STORAGE TO VITAL-STORAGE-LENGTH.
    CALL 'RAIAPI' USING RAI-INITIALIZE,                                     <=== (5)
        FCA,
        VITAL-STORAGE-BEGIN,
        VITAL-STORAGE-END.
    PERFORM A100-INITIALIZE THRU A100-EXIT.
    PERFORM B100-UNIT-OF-WORK THRU B100-EXIT
        UNTIL END-OF-FILE
        OR    PROCESSING-ERROR.
    DISPLAY 'EXAMPLE - PROCESSING COMPLETED'.
    CALL 'RAIAPI' USING RAI-TERMINATE,                                     <=== (6)
        FCA.
    GOBACK.

A100-INITIALIZE.
    IF FCA-RESTART-RUN THEN                                               <=== (7)
        DISPLAY 'THIS IS A RESTART RUN OF THE EXAMPLE3 PROGRAM'

```

```

ELSE
    DISPLAY 'THIS IS THE INITIAL RUN OF THE EXAMPLE3 PROGRAM'.
    OPEN INPUT INPUT-FILE.
    MOVE 'CKPT_PACE(10)' TO PROFILE-COMMAND-BUFFER.
    CALL 'RAIAPI' USING RAI-PROFILE,                <=== (8)
                        FCA,
                        PROFILE-COMMAND-BUFFER.

A100-EXIT.
EXIT.

B100-UNIT-OF-WORK.
MOVE 0 TO READ-COUNT.
PERFORM B200-PROCESS-TRANSACTION THRU B200-EXIT
    UNTIL TRANSACTION-COUNT > 10
    OR    END-OF-FILE OR PROCESSING-ERROR.

IF SUCCESSFUL-PROCESSING
    CALL 'RAIAPI' USING RAI-CHECKPOINT,            <=== (9)
                        FCA
    ADD 1 TO UOW-COUNT
ELSE
    CALL 'RAIAPI' USING RAI-ROLLBACK,              <=== (10)
                        FCA.

B100-EXIT.
EXIT.

B200-PROCESS-TRANSACTION.
READ INPUT-FILE INTO HOST-VARIABLES
    AT END MOVE 'E' TO PROCESSING-FLAG
    GO TO B200-EXIT.
IF INPUT-FILE-STATUS NOT = '00' THEN
    PERFORM IO-ERROR-ROUTINE
    GO TO B200-EXIT.

ADD 1 TO TRANSACTION-COUNT.
EXEC SQL INSERT INTO SOME.TABLE
    VALUES (:HOST-VARIABLE-1, :HOST-VARIABLE-2, :HOST-VARIABLE-3)
END-EXEC.
IF SQLCODE < 0 THEN ...
B200-EXIT.
EXIT.

```

where

- (1) Identifies the storage area that should be saved at checkpoint time and restored at restart time.
- (2) Defines a full word binary data item that contains the length of the storage area that must be saved and restored.
- (3) Copy the COBOL structure which defines the names of Smart/RESTART services available through the Call Level Interface.
- (4) Copy the structure which defines the COBOL version of the Smart/RESTART Function Communications Area (FCA).
- (5) The call to the Smart/RESTART INITIALIZE service immediately follows the PROCEDURE DIVISION statement. The INITIALIZE service identifies the address and length of the storage to be saved at checkpoint time and restored at restart time. The INITIALIZE service also sets flags which enable your application to distinguish between an initial and a restart run. (Discussed in Section 11.4.)

- (6) The call to the Smart/RESTART TERMINATE service signals normal module termination. When a *main module* calls the TERMINATE service, it informs Smart/RESTART that the application has completed its processing successfully and no restart is required. In contrast, the restart environment remains active when a *subprogram* calls the TERMINATE service. The subprogram simply returns to its caller. (Discussed in Section 11.5.)
- (7) The FCA data structure (embedded via the COPY SRSCFCA statement) defines the COBOL condition names FCA-INITIAL-RUN and FCA-RESTART-RUN. Your application can reference these condition names to distinguish between an initial run and a restart run and perform different initialization processing if appropriate.
- (8) Your application can call the PROFILE service to control Smart/RESTART's run-time behavior. The third parameter on the call to the PROFILE service contains the profile command. In this example, the string 'CKPT_PACE(10)' is moved to the field named PROFILE-COMMAND-BUFFER to govern checkpoint pacing.

NOTE: *Part II of the Smart/RESTART Reference describes each of the Smart/RESTART profile commands in detail.*

- (9) The EXAMPLE3 program calls the Smart/RESTART CHECKPOINT service when it successfully completes a unit-of-work. The scope of this restartable-unit-of-work spans both application processing and resource manager activity. Thus, storage contents and sequential file position are guaranteed to be in synch with changes to 'protected' resources managed by RRS compliant resource managers such as DB2, WebSphere MQ and/or IMS. (Discussed in Section 11.6.)
- (10) The application calls the Smart/RESTART ROLLBACK service if the unit-of-work fails to complete successfully. Smart/RESTART's coordinated ROLLBACK service backs out changes to application storage, sequential file position and 'protected' resources managed by RRS compliant resource managers such as DB2, WebSphere MQ and IMS to a prior point of consistency. (Discussed in Section 11.7.)

Figure 11.2 *Sample DB2/COBOL Program Using the Call Level Interface*

11.3 The INITIALIZE service

Among the first actions your application should perform is to call the Smart/RESTART INITIALIZE service. Through the INITIALIZE service, you identify to Smart/RESTART those storage areas within your program which are critical to your application's processing. Once identified, Smart/RESTART saves their contents each time you request a checkpoint (either explicitly through the Smart/RESTART checkpoint service or implicitly via EXEC SQL COMMIT). During application restart, the INITIALIZE service restores storage contents and resets sequential file position in synch with the point of consistency maintained by RRS compliant resource managers such as DB2, IMS and WebSphere MQ.

However, not all applications require you to checkpoint storage in its entirety. During application design, you should identify which storage areas should be checkpointed in accordance with the principle ***less is more***. That is because it is more efficient to checkpoint only those vital fields (flags, counters, key values and the like) whose values must ***persist*** beyond failure and be ***restored*** on restart.

For example, it is not necessary to checkpoint storage areas (such as the INPUT-RECORD in Figure 11.2) which are constantly being overwritten. Nor is it appropriate to checkpoint large storage areas whose values are set during program initialization and never change. An in-storage table loaded from an external file is a good example of stable data which need not be checkpointed. Typically, it is more efficient to *reload* such a table rather than checkpoint it.

NOTE: Smart/RESTART places no limit on the amount of application storage which can be checkpointed. Moreover, the checkpoint service is extremely efficient, so it's not worth spending a great deal of time to minimize the amount of storage that is checkpointed.

NOTE: Smart/RESTART permits you to save multiple, discontinuous data areas through the checkpoint service. Sections 50.1.1 and 50.1.2 of Chapter 50 describe and illustrate how to do so via Smart/RESTART's Call Level API.

11.3.1 The INITIALIZE service -- Format I

You identify the storage you wish Smart/RESTART to save at checkpoint time as ***pairs of parameters*** in either of two formats. In the first format, each pair of parameters is specified as a storage address followed by a full word binary field containing the length of the field or structure you wish Smart/RESTART to save. The COBOL source fragment in Figure 11.3 illustrates this first technique:

```

01 VITAL-STORAGE.
   05 VITAL-COUNTER      PIC S9(5) COMP.
   05 VITAL-SUBSCRIPT    PIC S9(5) COMP.
   05 VITAL-PAGE-NUMBER  PIC S9(5) COMP.
   05 VITAL-POINTER      PIC S9(5) COMP.
   05 VITAL-FLAGS.
      10 VITAL-FLAG1     PIC X(1).
      10 VITAL-FLAG2     PIC X(1).
      10 VITAL-FLAG3     PIC X(1).

01 VITAL-STORAGE-LENGTH  PIC S9(5) COMP.          <=== (1)

MOVE 'L' TO FCA-INIT-STYLE.
MOVE LENGTH OF VITAL-STORAGE TO VITAL-STORAGE-LENGTH.  <=== (2)
CALL 'RAIAPI' USING RAI-INITIALIZE,
                    FCA,
                    VITAL-STORAGE,
                    VITAL-STORAGE-LENGTH.

```

where

- (1) The length field is defined as a signed fullword binary numeric data item whose COBOL picture clause is S9(5) COMP.
- (2) Set the length of the structure named VITAL-STORAGE.

Figure 11.3 *INITIALIZE call using Address / Length Technique*

Alternatively, you can make use of the COBOL 'LENGTH OF' special register on your calls to the INITIALIZE service as illustrated in the coding fragment in Figure 11.4. This makes it unnecessary to define a length field or set its value.

```

01 VITAL-STORAGE.
   05 VITAL-COUNTER      PIC S9(5) COMP.
   05 VITAL-SUBSCRIPT    PIC S9(5) COMP.
   05 VITAL-PAGE-NUMBER  PIC S9(5) COMP.
   05 VITAL-POINTER      PIC S9(5) COMP.
   05 VITAL-FLAGS.
      10 VITAL-FLAG1     PIC X(1).
      10 VITAL-FLAG2     PIC X(1).
      10 VITAL-FLAG3     PIC X(1).
   . . .
MOVE 'L' TO FCA-INIT-STYLE.
CALL 'RAIAPI' USING RAI-INITIALIZE
                    FCA
                    VITAL-STORAGE
                    BY CONTENT LENGTH OF VITAL-STORAGE.          <=== (1)

```

where

- (1) Note the use of the BY CONTENT clause on the CALL statement. The BY CONTENT clause is mandatory because the default (BY ADDRESS) produces erroneous results.

Figure 11.4 *INITIALIZE call using LENGTH OF special register*

11.3.2 The INITIALIZE service - Format II

In the second format, Smart/RESTART interprets the pair of parameters specified on the INITIALIZE call as a starting address and ending address. In this case, Smart/RESTART saves the entire range of storage between (and including) the starting and ending addresses. The COBOL fragment in Figure 11.5 illustrates how to save the storage areas between VITAL-STORAGE-BEGIN and VITAL-STORAGE-END using this technique.

```
01 VITAL-STORAGE-BEGIN.
05 VITAL-COUNTER      PIC S9(5) COMP.
05 VITAL-SUBSCRIPT    PIC S9(5) COMP.
05 VITAL-PAGE-NUMBER  PIC S9(5) COMP.
05 VITAL-POINTER      PIC S9(5) COMP.
05 VITAL-FLAGS.
10 VITAL-FLAG1        PIC X(1).
10 VITAL-FLAG2        PIC X(1).
10 VITAL-FLAG3        PIC X(1).

77 VITAL-STORAGE-END  PIC X.

PROCEDURE DIVISION.
  MOVE 'E' TO FCA-INIT-STYLE.
  CALL 'RAIAPI' USING RAI-INITIALIZE
                      FCA
                      VITAL-STORAGE-BEGIN
                      VITAL-STORAGE-END.
```

Figure 11.5 INITIALIZE call using Begin Address / End Address Technique

NOTE: The FCA-INIT-STYLE variable governs whether Smart/RESTART interprets each parameter pair as a storage address and length or as a starting-address and ending address. The default supplied by RAI is address/length.

11.4 Distinguishing between Initial and Restart Runs

Your application may need to distinguish between an initial run and a restart run. The FCA data structure allows your application and Smart/RESTART to communicate. The INITIALIZE service sets the value of a field named FCA-STATUS. A value of spaces for the FCA-STATUS field denotes an initial run while a value of 'REST' indicates an application restart. In addition, Smart/RESTART defines the following two COBOL condition names through which you can refer to an original or restart run respectively:

```
FCA-INITIAL-RUN  VALUE SPACES.
FCA-RESTART-RUN  VALUE 'REST'.
```

NOTE: Appendix B lists the FCA and other Smart/RESTART data structures in each of the languages Smart/RESTART supports. The Smart/RESTART Reference provides detailed descriptions of these data structures and the fields which comprise them.

The COBOL code fragment in Figure 11.6 references the COBOL condition name FCA-RESTART-RUN. The program EXAMPLE3 displays a different message in the restart environment to distinguish it from an initial run.

```
A100-INITIALIZE.  
    IF FCA-RESTART-RUN THEN                                <=== (1)  
        DISPLAY 'THIS IS A RESTART RUN OF PROGRAM EXAMPLE3'  
    ELSE  
        DISPLAY 'THIS IS THE INITIAL RUN OF PROGRAM EXAMPLE3'.
```

where

(1) Code makes reference to the condition name FCA-RESTART-RUN

Figure 11.6 Distinguishing between initial and restart environments

11.5 The TERMINATE Service

Among the last actions your program should perform is to call the TERMINATE service to signal normal module termination. For modules written in COBOL, the call to the TERMINATE service should be positioned before any GOBACK or STOP RUN statement. For other languages, the placement of the TERMINATE call should be before any point in the module where control is returned to its caller.

NOTE: No further Smart/RESTART service calls are permitted after a module calls the TERMINATE service.

11.6 The CHECKPOINT Service

Your application should signal the completion of a *restartable*-unit-of-work when it reaches a point of consistency in its processing (with respect to application storage, sequential file position and updates to protected resources managed by RRS compliant resources managers such as DB2, IMS, and WebSphere MQ. This can be accomplished *explicitly* by calling the Smart/RESTART CHECKPOINT service or *implicitly* via an EXEC SQL COMMIT request.

NOTE: You should call the Smart/RESTART CHECKPOINT service in lieu of issuing an EXEC SQL COMMIT request if you use a DB2 attachment facility other than Smart/RRSAF. In this latter case, Smart/RESTART does not receive control when an EXEC SQL COMMIT request is issued.

In the explicit case illustrated in Figure 11.7, Smart/RESTART knows what ‘protected’ resources are accessed by your application and issues a checkpoint request appropriate for the resource manager(s) accessed. Because the scope of this restartable unit-of-work spans both application processing and resource manager activity, it represents a point of consistency from which processing can successfully resume upon restart.

```

B100-UNIT-OF-WORK.
  MOVE 0 TO READ-COUNT.
  PERFORM B200-PROCESS-TRANSACTION THRU B200-EXIT
    UNTIL TRANSACTION-COUNT > 10
    OR    END-OF-FILE OR PROCESSING-ERROR.

  IF SUCCESSFUL-PROCESSING
    CALL 'RAIAPI' USING RAI-CHECKPOINT,      <=== Explicit Checkpoint Request
                                     FCA
    ADD 1 TO UOW-COUNT
  ELSE
    CALL 'RAIAPI' USING RAI-ROLLBACK,
                                     FCA.
B100-EXIT.
EXIT.

```

Figure 11.7 *Explicit call to the CHECKPOINT service*

NOTE: Application storage, sequential file position and resource manager activity are all checkpointed at the same time. Therefore, it is **incorrect** to call the Smart/RESTART CHECKPOINT service and then issue a separate EXEC SQL COMMIT request.

The code fragment illustrated in Figure 11.8 issues an EXEC SQL COMMIT request when the unit-of-work completes successfully. In this case, the Smart/RRSAF component of Smart/RESTART intercepts the COMMIT request and calls the Smart/RESTART CHECKPOINT service. The CHECKPOINT service saves an execution-time ‘snapshot’ of application storage and sequential file position that is guaranteed to be in synch with the DB2 changes made permanent by EXEC SQL COMMIT.

```

B100-UNIT-OF-WORK.
  MOVE 0 TO READ-COUNT.
  PERFORM B200-PROCESS-TRANSACTION THRU B200-EXIT
    UNTIL TRANSACTION-COUNT > 10
    OR    END-OF-FILE OR PROCESSING-ERROR.

  IF SUCCESSFUL-PROCESSING
    EXEC SQL COMMIT END-EXEC                <=== Implicit Checkpoint Request
    ADD 1 TO UOW-COUNT
  ELSE
    EXEC SQL ROLLBACK WORK END-EXEC.
B100-EXIT.
EXIT.

```

Figure 11.8 *Implicit checkpoint request via EXEC SQL COMMIT*

11.7 The ROLLBACK Service

Your application should signal Smart/RESTART when a unit-of-work *fails* to complete successfully. This can be accomplished *explicitly* by calling the Smart/RESTART ROLLBACK service or *implicitly* by issuing an EXEC **SQL** ROLLBACK request. Smart/RESTART's coordinated backout facility restores application storage, sequential file position and RRS compliant protected resources (such as DB2 tables, IMS databases and/or WebSphere MQ queues) to their last checkpointed state. This represents a point of consistency from which processing can successfully resume.

You should call the Smart/RESTART ROLLBACK service when your application accesses IMS databases, WebSphere MQs queues, or uses an attachment facility other than Smart/RRSAF to access DB2. In the explicit case illustrated in Figure 11.9, Smart/RESTART knows what 'protected' resources are accessed by your application and issues a backout request appropriate for those resource manager(s).

```
B100-UNIT-OF-WORK.  
  MOVE 0 TO READ-COUNT.  
  PERFORM B200-PROCESS-TRANSACTION THRU B200-EXIT  
    UNTIL TRANSACTION-COUNT > 10  
    OR    END-OF-FILE OR PROCESSING-ERROR.  
  
  IF SUCCESSFUL-PROCESSING  
    CALL 'RAIAPI' USING RAI-CHECKPOINT,      <=== Explicit Rollback Request  
                                FCA  
    ADD 1 TO UOW-COUNT  
  ELSE  
    CALL 'RAIAPI' USING RAI-ROLLBACK,  
                                FCA.  
B100-EXIT.  
EXIT.
```

Figure 11.9 *Explicit call to the Smart/RESTART ROLLBACK service*

The code fragment illustrated in Figure 11.10 issues an EXEC SQL ROLLBACK request when the unit-of-work fails to complete successfully. The Smart/RRSAF component of Smart/RESTART intercepts the ROLLBACK request and calls Smart/RESTART's more comprehensive ROLLBACK service.

```

B100-UNIT-OF-WORK.
  MOVE 0 TO READ-COUNT.
  PERFORM B200-PROCESS-TRANSACTION THRU B200-EXIT
    UNTIL TRANSACTION-COUNT > 10
    OR    END-OF-FILE OR PROCESSING-ERROR.

  IF SUCCESSFUL-PROCESSING
    EXEC SQL COMMIT WORK END-EXEC
    ADD 1 TO UOW-COUNT
  ELSE
    EXEC SQL ROLLBACK WORK END-EXEC.      <=== implicit ROLLBACK request
B100-EXIT.
EXIT.

```

Figure 11.10 *Implicit backout request via EXEC SQL ROLLBACK*

NOTE: You should call the Smart/RESTART ROLLBACK service instead of issuing an EXEC SQL ROLLBACK request if you use a DB2 attachment facility other than Smart/RRSAF. In this latter case, Smart/RESTART does not receive control when EXEC SQL ROLLBACK is issued.

11.8 Designating an Explicit Restart Point

By default, Smart/RESTART restores the run-time environment at the *start* of a restart run. This places a burden on your program to ensure that the procedural initialization of variables only occurs in an initial run. Restart points are intended to be placed *after* application initialization and *before* the main processing loop. This allows application initialization to run unconditionally. Any changes made to variables in the initialization routines are reset when application storage areas are restored (at the restart point) to their state as of the last successful checkpoint.

Smart/RESTART supports RESTART processing at explicit restart points (which Chapter 51 describes in detail). Restart Points let you control **when** during a restart run application storage is to be restored and sequential files repositioned -- a process known as restoring the run-time environment. Restart Points can also be thought of as restore points.

Chapter 12

Object Transparency

Object Transparency refers to the Smart/RESTART facility that endows existing DB2/COBOL applications with restart support -- without source code changes of any kind and without the need to recompile. This makes Object Transparency appropriate for software supplied in object code only format or when program preparation, for whatever reason, is not feasible. Object Transparency is an attractive alternative to the MVS CKPT macro. It has extremely modest overhead in comparison to the MVS CKPT facility, yet it lets you restart after recompiles or system IPLs.

You should have realistic expectations about Object Transparency. You can not use it to restart enable an entire portfolio of existing applications -- regardless of their structure. This is because Smart/RESTART's role is limited to keeping application storage and sequential file position in synch with updates to RRS compliant resources (such as those managed by DB2 and WebSphere MQ). Therefore, Object Transparency is suitable only for 'well behaved' applications which adhere to the design principles described in Chapter 2. Otherwise, these applications will be **unable** to restart successfully.

12.1 Requirements and Restrictions on Object Transparency

The following requirements and restrictions apply to applications which are candidates for Object Transparency -- in addition to the requirements described in Chapter 2.

- Object Transparency only supports applications written in COBOL.
- Object Transparency supports access to sequential files, VSAM datasets and RRS compliant resource managers. Access to IMS databases is supported only via the RRS compliant ODBA interface.
- Object Transparency only supports Smart/RRSAF as a DB2 attachment facility. No other attachment facility (including those supplied by IBM) are supported.
- Object Transparency supports applications comprised of multiple modules only if the unit-of-work is logically defined in the main routine (although the actual checkpoint request can be issued by a subroutine). Smart/RESTART can checkpoint the working-storage of any dynamically called subprogram that is linked with the REUS option. Statically called subprograms can access the storage defined within the main routine through Linkage Section definitions.
- Object Transparency does not support data areas defined as EXTERNAL or GLOBAL.
- The CA-Optimizer is not supported.
- Object Transparency does not support the COBOL CMPR2 option.
- If the main COBOL module is compiled with the OPTIMIZE option, the main module's name must be defined in the RAINPUT stream via the MAIN_DBRM parameter (described in Chapter 12 of the Smart/RESTART Reference).

12.2 Object Transparency Example

This section presents a complete DB2/COBOL program that operates in **Object Transparency** mode. It is identical to the source transparent application in Chapter 2. It illustrates how restartable applications can be developed without any reference to Smart/RESTART whatsoever. The bold, italic numbers within parentheses correspond to the numbered annotations which follow the program.

In **Object Transparency** mode, Smart/RESTART intercepts all sequential I/O and SQL requests. Smart/RESTART also saves all storage when your application issues an EXEC SQL COMMIT request. Should an Object Transparent application fail and be restarted, *all* application storage is restored to its state at checkpoint time.

```

ID DIVISION.
PROGRAM-ID.      EXAMPLE.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INPUT-FILE ASSIGN TO INFILE
        FILE STATUS IS INPUT-FILE-STATUS.

DATA DIVISION.
FILE SECTION.

FD  INPUT-FILE
01  INPUT-RECORD                      PIC X(80).

WORKING-STORAGE SECTION.

01  VITAL-STORAGE.
    05  UOW-COUNT                      PIC S9(4) COMP VALUE 1.
    05  TRANSACTION-COUNT              PIC S9(4) COMP VALUE 0.

01  HOST-VARIABLES.
    05  HOST-VARIABLE-1                PIC S9(5) COMP.
    05  HOST-VARIABLE-2                PIC S9(5) COMP.
    05  HOST-VARIABLE-3                PIC S9(5) COMP.

EXEC SQL INCLUDE SQLCA END-EXEC.

01  NON-CRITICAL-FIELDS.
    05  PROCESSING-FLAG                PIC X(1) VALUE SPACE.
    88  SUCCESSFUL-PROCESSING          VALUES ARE ' ' 'E'
    88  END-OF-FILE                    VALUE 'E'.
    88  IO-ERROR                       VALUES ARE 'I'.
    88  PROCESSING-ERROR               VALUES ARE 'I' 'P'.
    05  INPUT-FILE-STATUS              PIC 9(02) VALUE ZEROS.

PROCEDURE DIVISION.
    PERFORM A100-INITIALIZE THRU A100-EXIT.
    PERFORM B100-UNIT-OF-WORK THRU B100-EXIT      <=== (1)
        UNTIL END-OF-FILE
        OR    PROCESSING-ERROR.
    DISPLAY 'EXAMPLE - PROCESSING COMPLETED'.
    GOBACK.

A100-INITIALIZE.
    IF RETURN-CODE = +2001                      <=== (2)
        DISPLAY 'THIS IS A RESTART RUN OF THE EXAMPLE PROGRAM'
    ELSE
        DISPLAY 'THIS IS THE INITIAL RUN OF THE EXAMPLE PROGRAM'.
    OPEN INPUT INPUT-FILE.                      <=== (3)
A100-EXIT.
    EXIT.

B100-UNIT-OF-WORK.
    MOVE 0 TO READ-COUNT.
    PERFORM B200-PROCESS-TRANSACTION THRU B200-EXIT
        UNTIL TRANSACTION-COUNT > 10
        OR    END-OF-FILE OR PROCESSING-ERROR.

    IF SUCCESSFUL-PROCESSING
        EXEC SQL COMMIT WORK END-EXEC          <=== (4)
        ADD 1 TO UOW-COUNT
    ELSE
        EXEC SQL ROLLBACK WORK END-EXEC.        <=== (5)
B100-EXIT.
    EXIT.

```

```

B200-PROCESS-TRANSACTION.
  READ INPUT-FILE INTO HOST-VARIABLES
  AT END MOVE 'E' TO PROCESSING-FLAG
  GO TO B200-EXIT.
  IF INPUT-FILE-STATUS NOT = '00' THEN
    PERFORM IO-ERROR-ROUTINE
    GO TO B200-EXIT.

  ADD 1 TO TRANSACTION-COUNT.
  EXEC SQL INSERT INTO SOME.TABLE
    VALUES (:HOST-VARIABLE-1, :HOST-VARIABLE-2, :HOST-VARIABLE-3)
  END-EXEC.
  IF SQLCODE < 0 THEN ...
B200-EXIT.
EXIT.

```

where

- (1) The flow of control of an Object Transparent application should enter its main unit-of-work loop at the same point in either an initial run or a restart run. Thus, execution begins at a unit-of-work boundary whether processing *commences* on an initial run or *resumes* during a restart run.
- (2) During a restart run, Smart/RESTART sets the COBOL special register RETURN-CODE to the value +2001 to enable an Object Transparent application to distinguish between an initial run and restart run.
- (3) OPEN works normally during an initial run. However, during a restart run, Smart/RESTART sets sequential files to their position at checkpoint time as part of OPEN processing.
- (4) An Object Transparent application must issue an EXEC SQL COMMIT request or call SRRCMIT when the unit-of-work completes successfully. Smart/RESTART intercepts either request and calls the Smart/RESTART CHECKPOINT service. This Smart/RESTART facility saves an execution-time 'snapshot' of application storage and sequential file position that is guaranteed to be in synch with the DB2 changes made permanent by EXEC SQL COMMIT. Because the scope of this restartable unit-of-work spans both application processing and resource manager activity, it represents a point of consistency from which processing can resume successfully upon restart.
- (5) An Object Transparent application must issue an EXEC SQL ROLLBACK request or call SRRBACK if the unit-of-work fails to complete successfully. Smart/RESTART intercepts either request and calls Smart/RESTART's more comprehensive ROLLBACK service. The ROLLBACK facility implemented by Smart/RESTART backs out changes to application storage, sequential file position and DB2 to their last checkpointed state. This represents a point of consistency from which processing can successfully resume.

Figure 12.1 *Object Transparent DB2/COBOL Application*

12.3 JCL to run the Object Transparent Application

Figures 12.2.1 and 12.2.2 illustrate JCL to run the Object Transparent EXAMPLE program. Chapter 4 (which discusses how to run applications with Smart/RESTART) applies to Object Transparent applications as well – with the exception of the JCL EXEC statement which is described in this section. The same JCL can be used for both initial and restart runs because all files reference pre-allocated datasets. The numbers in the right hand margin (within parentheses) correspond to the numbered annotations which follow the JCL.

```
//STEP1 EXEC PGM=SRS,PARM='EXAMPLE,EXAMPLE PARS' <=== (1)
//STEPLIB DD DSN=user.program.library,DISP=SHR
//          DD DSN=DCAvrn.DCALOAD,DISP=SHR
//          DD DSN=DSNxxx.SDSNLOAD,DISP=SHR
//SRSCHECK DD DSN=check.point.dataset,DISP=OLD
//SRSPRINT DD SYSOUT=*
//INFILE   DD DSN=program.input.dataset,DISP=OLD
//OUTFILE  DD DSN=program.output.dataset,DISP=OLD
//RAINPUT  DD *
           CKPT_PACE(50)
//
```

where

- (1) To run in Object Transparency mode, the Smart/RESTART front end program named SRS must be invoked through the PGM parameter of the JCL EXEC statement. The name of the Object Transparent application (in this case EXAMPLE) is specified as the first parameter of the PARM operand. Any parameters the Object Transparent application requires can be specified after a comma ‘,’ that follows the program name.

Figure 12.2.1 JCL to run an Object Transparent Application with altered PARM

```
//STEP1 EXEC PGM=SRS,PARM='EXAMPLE PARS' <=== (1)
//STEPLIB DD DSN=user.program.library,DISP=SHR
//          DD DSN=DCAvrn.DCALOAD,DISP=SHR
//          DD DSN=DSNxxx.SDSNLOAD,DISP=SHR
//SRSCHECK DD DSN=check.point.dataset,DISP=OLD
//SRSPRINT DD SYSOUT=*
//INFILE   DD DSN=program.input.dataset,DISP=OLD
//OUTFILE  DD DSN=program.output.dataset,DISP=OLD
//RAINPUT  DD *
MAIN_PROGRAM(EXAMPLE) <=== (2)
CKPT_PACE(50)
//
```

where

- (1) To run in Object Transparency mode, the Smart/RESTART front end program named SRS must be invoked through the PGM parameter of the JCL EXEC statement.

- (2) The name of the Object Transparent application (in this case EXAMPLE) is specified using the `MAIN_PROGRAM` directive.

Figure 12.2.2 JCL to run an Object Transparent Application without altered PARM

Object Transparent applications behave like regular OS jobs because they use Smart/RRSAF as a DB2 attachment facility. As such, job step completion codes accurately reflect the outcome of processing. Dataset dispositions also behave as expected and intended. However, SMF charge-back accounting for all Object Transparent applications is recorded under the name of the SRS front-end module.

12.4 Implementing Object Transparency

This section describes how to alter an existing application and its JCL to run in Object Transparency mode. These steps are *in addition* to the material presented in Chapter 4.

If your DB2 application was compiled with the COBOL NODYNAM option (i.e. called subroutines are physically part of the load module), you must re-link your application to include the DB2 attachment facility supplied with Smart/RRSAF as well as the SRRCMIT and SRRBACK routines supplied with Smart/RESTART. Alternatively, applications which call their subroutines dynamically must ensure the routines supplied with Smart/RRSAF and Smart/RESTART are invoked at execution time. See Chapter 4.1.1 of this manual and the Smart/RRSAF User Guide for details.

The JCL to run an existing application should be altered as follows:

- > The JCL EXEC statement should specify `PGM=SRS` rather than the name of application load module to be run in Object Transparency mode.
- > Identify the application program name as the first field of the `PARM` operand as follows:

Original EXEC statement: `//STEP1 EXEC PGM=EXAMPLE`

Modified EXEC statement: `//STEP1 EXEC PGM=SRS,PARM='EXAMPLE'`

Alternatively, use the Smart/RESTART `MAIN_PROGRAM` directive as described in the figure 12.2.2 above.

- > Append any parameters your application requires after a comma that follows the program name.

Original EXEC statement: `//STEP1 EXEC PGM=EXAMPLE,PARM='PARMS'`

Modified EXEC statement: `//STEP1 EXEC PGM=SRS,PARM='EXAMPLE,PARMS'`

Resource Recovery Services and Applications that Access Multiple Resource Managers

Smart/RESTART supports applications that access any combination of RRS compliant resources (whose resource managers are registered with MVS Resource Recovery Services). A single Smart/RESTART application can access and update *multiple* protected resources managed by RRS compliant products such as DB2, WebSphere MQ, IMS and IDMS. Your resource manager administrator can tell you whether a particular resource manager participates in RRS.

NOTE: *Smart/RESTART supports RRS compliant access to IMS databases only via the Open Database Access (ODBA) callable interface named AERTDLI. Chapter 23 discusses Smart/RESTART support for applications which access IMS resources within DLIBATCH and BMP environments.*

When a restartable application is ready to commit or backout its changes, the application invokes RRS to begin the two-phase commit process for a unit of recovery (UR). Call the RRS commit service (SRRCMIT) to indicate that the changes made during a UR are to be made permanent. Alternatively, call the RRS backout service (SRRBACK) to request that the changes during a UR are to be undone – i.e. backed out as if they never took place. A UR represents the application's changes to resources since the last commit or backout request or, for the first UR, since the beginning of the application.

Smart/RESTART provides an application stub named SRS4SRR that defines SRRCMIT, SRRBACK and ATRSCSS as *alternate entry points*. SRS4SRR provides the link between your batch program, Smart/RESTART and RRS.

As documented in Chapter 3.5, the SRS4SRR stub must appear first in the SYSLIB concatenation for applications that statically link their subroutines at compile / link edit time – *ahead* of the SRRCMIT and SRRBACK stubs provided by RRS in data set SYS1.CSSLIB or the RRS stubs provided with resource managers like DB2 and WebSphere MQ.

Similarly, applications that load and call their subroutines dynamically at run-time must ensure the Smart/RESTART stub SRS4SRR appears first in the STEPLIB or JOBLIB concatenation (as described in Chapter 4.1.1). Two copies of the SRS4SRR module are supplied with Smart/RESTART. One copy resides in the DCANCAL library, to be referenced in link edit JCL while another copy resides in the DCALOAD library to be referenced in run-time JCL.

Your applications can also call Smart/RESTART's own checkpoint or rollback services *directly* to request commit or backout processing. Furthermore, when Smart/RRSAF is used, Smart/RESTART can transparently call the RRS commit or backout services respectively when your program issues a standard SQL COMMIT or ROLLBACK request at run-time. This enables existing applications to take advantage of RRS support immediately, without revisions to source code.

A sample COBOL program named SRS8DBMQ is supplied in the DCASAMP library to demonstrate a restartable application which accesses both DB2 and WebSphere MQ resources within the *same* unit of recovery. The SRSBDBMQ member of the DCASAMP library contains the JCL with which to *build* the SRS8DBMQ application while the SRSRDBMQ member contains the JCL to *run* the sample application.

The run-time requirements for restartable applications that access DB2, WebSphere MQ and other RRS compliant resources are illustrated in Figure 20.1 below which shows the run-time parameters passed to Smart/RESTART (through the RAINPUT file). The bold, italic numbers within parentheses correspond to the numbered annotations which follow the figure.

```

. . .
/**
//RAINPUT DD *
*
* Smart/RRSAF related profile commands follow:
*
ATTACH(RRSAF) (1)
SYSTEM(?dsn?) (2)
PLAN(?plan_name?) (3)
*
* Smart/RESTART related profile commands follow
*
RMI(SRSRDB2) (4)
QMGR(?qmgr?) (5)
. . .
/*

```

where

- (1) The ATTACH(RRSAF) parameter directs Smart/RRSAF to use the RRSAF attachment to access DB2. Smart/RRSAF is the DB2 Call Attachment Facility from Relational Architects.
 - (2) The SYSTEM(?dsn?) parameter identifies the 1 to 4 character name of the DB2 subsystem the program should access.
 - (3) The PLAN(?plan_name?) parameter specifies the name of the DB2 application plan.
 - (4) The RMI(SRSRDB2) parameter directs Smart/RESTART to use SRSRDB2 as its Resource Manager Interface routine. SRSRDB2 uses RRS (the Resource Recovery Services component of z/OS) to handle the two phase commit protocol for RRS compliant resources such as DB2, WebSphere MQ and IMS and maintains Smart/RESTART synchronization data in a DB2 table.
 - (5) The QMGR(?qmgr?) parameter identifies the WebSphere MQ queue manager to which your application and optionally Smart/RESTART will connect.
-

Additional DB2 Topics

21.1 Set Current Packageset

DB2 applications that set the CURRENT PACKAGESET special register change the normal search order for finding plan elements (DBRM's and packages) in a DB2 application plan. When the CURRENT PACKAGESET special register is set to a non-blank value, the search for packages is restricted to the collection specified in the SET CURRENT PACKAGESET statement. Any other collections in the package list of the plan are ignored.

To assist applications that issue SQL SET CURRENT PACKAGESET statements, Smart/RESTART provides two run-time parameters, SET_PACKAGE and COLLECTION. These parameters:

- direct the SRSDBRM program to save the value of the current package collection,
- issue a SET CURRENT PACKAGESET statement to specify a collection ID known to contain the SRSDBRM package,
- issue its SQL statements and, finally
- return the CURRENT PACKAGESET to its initial value with another SET CURRENT PACKAGESET statement.

This processing is intended for development and test environments **only** since PACKAGESET switching of a routine nature incurs high overhead. For production applications that set the CURRENT PACKAGESET special register, RAI strongly recommends the SRSDBRM package be bound into every package collection that may be included within the DB2 plans of restartable applications.

NOTE: The SET_PACKAGE and COLLECTION run time parameters are described and illustrated in Chapter 14 of the Smart/RESTART Reference.

21.2 DB2 Savepoints

Although the SQL statements

```
EXEC SQL SAVEPOINT
```

```
EXEC SQL ROLLBACK TO SAVEPOINT
```

can be used in conjunction with Smart/RESTART applications, RAI recommends these SQL statements ***not be used*** because their scope applies to relational database resources only. As such, Smart/RESTART does not checkpoint non-DB2 resources in response to SQL SAVEPOINT requests, nor does it rollback non-DB2 resources in response to a ROLLBACK TO SAVEPOINT command.

The scope of a restartable unit-of-work encompasses not just DB2 but all RRS compliant resources an application accesses as well as storage contents, VSAM data and sequential file position. Applications that issue SQL SAVEPOINT and/or ROLLBACK TO SAVEPOINT requests risk leaving resources within the scope of a restartable unit-of-work in an inconsistent state.

21.3 DB2 Stored Procedures

Smart/RESTART supports restartable batch applications that call DB2 stored procedures *provided* the restartable application manages the unit-of-work. This means any DB2 stored procedure(s) called by the restartable application:

- must be defined with COMMIT ON RETURN NO
- must not issue EXEC SQL COMMITs or ROLLBACK statements
- must not call SRRCMIT or SRRBACK to commit or rollback changes respectively to RRS compliant resources

In addition, automated error handling for SQL statements (via Smart/RESTART's SQL_TRAP facility) does not extend to the SQL statements issued by any DB2 stored procedures called by the restartable application may.

Chapter 22

DB2 Applications without Smart/RRSAF

Smart/RESTART is integrated with Smart/RRSAF, the DB2 attachment package from Relational Architects Intl. Smart/RESTART and Smart/RRSAF work together to provide valuable benefits -- such as automatic detection of and recovery from -'resource unavailable' conditions. However, Smart/RESTART can work (if necessary) with DB2 attachment packages other than Smart/RRSAF. These include the attachment packages supplied by IBM, any locally developed package or the DB2 attachment facilities supplied by other vendors. This chapter discusses how to use DB2 attachment packages other than Smart/RRSAF. It also describes those Smart/RESTART features that require Smart/RRSAF.

22.1 Requesting a Call Attach facility other than Smart/RRSAF

For Smart/RESTART to work with a DB2 attachment package other than Smart/RRSAF, the Smart/RESTART parameters CAF and RMI must be specified at run-time or be defined as Smart/RESTART defaults during installation, as described in Sections 22.1.1 and 22.1.2 respectively.

22.1.1 The RMI Parameter

The RMI parameter identifies the Smart/RESTART Resource Manager Interface module to be used at execution time. Its purpose is to store checkpoint synchronization information in a resource manager structure and then commit that data along with the application's own unit-of-work. An application that operates with DB2 attachment packages other than Smart/RRSAF should specify `RMI(NNNNNNNN)` as a Smart/RESTART run-time parameter where `NNNNNNNN` is the name of the load module Smart/RESTART should call at checkpoint time in order to issue an EXEC SQL COMMIT request. This load module must reside in a library defined in the STEPLIB concatenation of the Smart/RESTART jobstep or it must reside in the MVS link list.

Figure 22.1 enumerates the DB2 interface modules delivered with Smart/RESTART. Alternatively, you can build your own DB2 interface module as described in Section 22.3.1.

RM Interface	Remarks
SRSDBRM	Specify SRSDBRM to use the resource manager interface module that accesses DB2 through Smart/RRSAF. This is the default.
SRSDBRMI	Specify SRSDBRMI to use the resource manager interface module that accesses DB2 through the IMS attachment package.
SRSDBRMT	Specify SRSDBRMT to use the resource manager interface module that accesses DB2 through the TSO attachment package.
SRSDBRMO	Specify SRSDBRMO to use the resource manager interface module that accesses DB2 through a locally developed attachment facility or an attachment facility supplied by another vendor.

Figure 22.1 *DB2 Interface Routines*

22.2 Application Development Considerations

A Smart/RESTART application that runs with a DB2 attachment package other than Smart/RRSAF must be coded and prepared in a way that satisfies the requirements of both Smart/RESTART and the DB2 attachment package. This section discusses those requirements in detail.

22.2.1 SQL COMMIT and ROLLBACK statements

When Smart/RRSAF is the DB2 attachment mechanism, any EXEC SQL COMMIT request triggers the synchronized checkpoint of application storage, VSAM data, sequential file position and updates to DB2. Similarly, EXEC SQL ROLLBACK triggers coordinated backout. When a DB2 attachment package other than Smart/RRSAF is used, SQL COMMIT and ROLLBACK statements must be replaced with calls to the Smart/RESTART CHECKPOINT and ROLLBACK services, respectively. Smart/PRECOMPILER can make these changes automatically -- or you can make them yourself. Specify the Smart/PRECOMPILER parameter CAF(OTHER) to request this automatic conversion.

22.2.2 DB2 BIND

The Smart/RESTART package or DBRM named SRSDBRM must still be bound into any DB2 application plan that requests Smart/RESTART services -- even if the application makes use of a DB2 attachment package other than Smart/RRSAF. See Chapter 3.6 for details.

22.3 Smart/RESTART Installation Considerations

The Smart/RESTART installation process makes use of Smart/RRSAF. Therefore, Smart/RRSAF must be installed in order to install Smart/RESTART itself and then run the IVP's supplied with Smart/RESTART. Once Smart/RESTART is successfully installed, the Smart/RRSAF load modules may be deleted from the Smart/RESTART load library.

22.3.1 Using Other DB2 Attachment Facilities

When using a site developed DB2 attachment facility in conjunction with Smart/RESTART or one supplied by another vendor, you must link edit a Resource Manager Interface routine that accesses DB2 through this other attachment facility.

The JCL in Figure 22.2 illustrates how to build such a Resource Manager Interface routine and name the load module SRSDBRMO. The Smart/RESTART interface routine link edited in this example is identical to the Smart/RESTART interface routine for DB2 named SRSDBRM. However, it uses your locally developed attachment facility to access DB2 or one supplied by a third party vendor. The numbers in the right hand margin (within parentheses) correspond to the numbered annotations which follow the JCL.

Member SRSJDBRM of the DCACNTL library contains JCL you can customize to link the Smart/RESTART load module SRSDBRM with a site developed DB2 Attachment facility or with one supplied by another vendor.

NOTE: *The DBRM or package for SRSDBRM must still be bound into any DB2 application plan that requests Smart/RESTART services (as described in Section 22.2.2)*

```

//LINK      EXEC PGM=IEWL
//SYSLIB    DD DSN=automatic.call.library,DISP=SHR
//RAILIB    DD DSN=DCAvrml.DCANCAL,DISP=SHR          <--- (1)
//CAFLIB    DD DSN=other.caf.ncallib,DISP=SHR        <--- (2)
//SYSLMOD   DD DSN=other.caf.loadlib,DISP=SHR        <--- (3)
//SYSLIN    DD *
REPLACE DCAHLI                                     <--- (4)
INCLUDE RAILIB(SRSDBRM)                            <--- (5)
INCLUDE CAFLIB(othercaf)                            <--- (6)
ENTRY      SRSDBRM
AMODE(31),RMODE(ANY)
NAME       SRSDBRMO                                <--- (7)
//

```

where

- (1) The RAILIB DD statement defines the Smart/RESTART system NCAL library which contains SRSDBRM, the Smart/RESTART interface routine intended for use with other DB2 attachment facilities. Ask your Smart/RESTART product administrator for the name of this dataset in your environment.
- (2) The CAFLIB DD statement defines the load module library which contains your locally developed DB2 attachment facility. Alternatively, the CAFLIB DD statement should reference the system library in which the vendor supplied attachment facility resides.
- (3) The SYSLMOD DD statement defines the target load module library into which the SRSDBRMO load module will be built.
- (4) The REPLACE control statement directs the linkage editor to *delete* the Smart/RRSAF external symbol DCAHLI from the DB2 interface routine so it can be replaced by a vendor supplied or locally developed DB2 attachment facility.
- (5) The first INCLUDE control statement directs the linkage editor to fetch the load module named SRSDBRM from the dataset defined by the RAILIB DD statement and incorporate it in the executable load module.
- (6) The second INCLUDE control statement directs the linkage editor to fetch the load module named 'othercaf' from the dataset defined by the CAFLIB DD statement and incorporate it in the executable load module. 'Othercaf' is the name of your site developed DB2 attachment package. Alternatively, 'othercaf' identifies the DB2 attachment facility supplied by a third party vendor.
- (7) SRSDBRMO is the name of the link edited load module.

Figure 22.2 *Link Editing a new DB2 Interface Routine*

Figure 22.3 illustrates a DB2/COBOL application named EXAMPLE which uses a locally developed attachment facility. The EXAMPLE load module includes the DB2 attachment package named 'othercaf'. Alternatively, if the COBOL DYNAM option is used, 'othercaf' and the rest of the locally developed attachment facility modules must reside in one of the libraries defined in the STEPLIB concatenation.

NOTE: *You cannot use more than one DB2 attachment facility within a single jobstep. Both your application and Smart/RESTART's own Resource Manager Interface routine must use the same DB2 attachment facility.*

```
//EXAMPLE JOB
//STEP1 EXEC PGM=EXAMPLE,PARMS='EXAMPLE PARMS'
//STEPLIB DD DSN=user.program.library,DISP=SHR <--- (1)
//          DD DSN=other.caf.loadlib,DISP=SHR <--- (2)
//          DD DSN=DCAvrn.DCALOAD,DISP=SHR <--- (3)
//          DD DSN=DSNxxx.SDSNLOAD,DISP=SHR
//RAINPUT DD * <--- (4)
//          RMI(SRSDBRMO) <--- (5)
//
```

where

- (1) The first dataset in the STEPLIB concatenation defines the application library which contains the EXAMPLE load module. The DB2 attachment package named 'othercaf' is link edited within the EXAMPLE load module. Alternatively, if the COBOL DYNAM option is used, 'othercaf' and the rest of the locally developed attachment facility modules must reside in one of the libraries defined in the STEPLIB concatenation.
- (2) The second dataset in the STEPLIB concatenation defines the system library which contains 'othercaf' and the rest of the locally developed attachment facility modules. It also contains SRSDBRMO, the Smart/RESTART interface routine intended for use with other DB2 attachment facilities (see Figure 22.2).
- (3) The third dataset in the STEPLIB concatenation defines the Smart/RESTART system library.
- (4) The RAINPUT DD statement supplies run-time parameters to Smart/RESTART.
- (5) Lastly, RMI(SRSDBRMO) identifies load module SRSDBRMO as the Resource Manager Interface routine Smart/RESTART should utilize at run-time to issue its own internal SQL requests.

Figure 22.3 *Restartable Jobstep which uses SRSDBRMO*

22.4 Smart/RESTART Options Which Require Smart/RRSAF

The following Smart/RESTART features require Smart/RRSAF. If a DB2 attachment package other than Smart/RRSAF is used, these features will not work and should not be used.

- Smart/RESTART Object Transparency is not supported without Smart/RRSAF.
- Smart/RESTART cannot automatically detect a resource unavailable like a -911 SQLCODE without Smart/RRSAF. Since Smart/RESTART cannot issue a ROLLBACK request automatically, your program must explicitly request backout of changes to working storage, sequential file position and 'protected' resources managed by RRS compliant resource managers like DB2, IMS and/or WebSphere MQ.
- Monitoring of application SQL activity is not supported without Smart/RRSAF.
- Tracing of application SQL activity is not supported without Smart/RRSAF.

Smart/RESTART for IMS and Mixed IMS / DB2 Applications

Smart/RESTART supports applications which access IMS and/or DB2 data in both DLIBATCH and BMP environments. Smart/RESTART coordinates the commit and rollback of IMS and DB2 resources with your application's working storage contents, VSAM updates and sequential file position. With the consistency of IMS, DB2 and application resources guaranteed, your application can resume execution cleanly from a unit-of-work boundary.

Smart/RESTART support for the mixed IMS/DB2 environment couples robust restart support with simplified application development and maintenance. Operational procedures are simplified as well. Smart/RESTART is designed to make restart as simple as possible by facilitating recovery from both application and system failures.

Smart/RESTART improves the integrity of your application processing and reduces your exposure to inconsistent data as compared to native IMS facilities. For example, should your application abend with an 0C7, Smart/RESTART ensures that IMS databases, DB2 data and your application's working storage contents and sequential file position are all restored to a mutually consistent state as of the last successfully completed unit-of-work.

23.1 Checkpoint Restart in the IMS DB2 Environment

When DB2 is accessed in the IMS environment, SQL requests are passed to DB2 through the IMS attachment facility. When your application requests a Checkpoint, Smart/RESTART records your application's working storage contents and sequential file position, records the ID of this most recent checkpoint in its DB2 table and then calls IMS Basic Checkpoint to commit the changes to both IMS and DB2 resources.

Smart/RESTART applications which access both IMS and DB2 data can be resubmitted for restart using the same JCL used for the initial run -- provided the JCL adheres to the conventions discussed Section 23.4. On restart, Smart/RESTART restores your application's working storage contents and sequential file position to a state consistent with DB2 and IMS data restored to the same unit-of-work boundary.

The following sections describe how to design, code, prepare and execute Smart/RESTART applications which access IMS and/or DB2 data in DLIBATCH and BMP environments. This chapter concludes with an illustrative COBOL program and the JCL to run it as a DLIBATCH job or IMS BMP.

23.2 Application Design and Coding

The same principles apply to restartable application development for the IMS environment (or mixed IMS / DB2 environment) as for any other environment. These principles include taking checkpoints at unit-of-work boundaries and directing your application's flow of control to resume execution at a unit-of-work boundary in the event of a restart. Coding IMS / DB2 applications should follow the rules and guidelines described in IBM's documentation for DB2 and IMS.

Perhaps the simplest approach is to call IMS to handle checkpoint and rollback of database data, use native COBOL verbs for sequential file I/O and let Smart/PRECOMPILER do the rest. The PRECOMPILER adds code to Restart-enable your application. In addition, you can explicitly request Smart/RESTART services via EXEC SRS statements or by calling the Smart/RESTART API.

NOTE: Smart/RESTART 'Object Transparency' is not supported in either DLIBATCH or BMP environments.

23.2.1 Checkpoint, Rollback and Restart

When run as a DLIBATCH job or BMP, Smart/RESTART provides 'coordinated' checkpoint and rollback services whose scope includes changes to both IMS and DB2 data as well as to your application's working storage contents, VSAM datasets and sequential file position. Since IMS is the 'coordinator' for both IMS and DB2 data, IMS checkpoint and rollback calls should be used rather than SQL COMMIT and ROLLBACK statements to handle database synchronization. Smart/PRECOMPILER automatically converts any IMS CHKP, ROLL or ROLB calls it encounters in your source program into appropriate calls to Smart/RESTART services. Alternatively, you can call these Smart/RESTART services directly to request coordinated checkpoint / rollback services.

Note that in DLIBATCH and BMP environments, SQL COMMIT and ROLLBACK statements are *not* supported. Nor are the IMS ROLS and SETS functions. (These latter two functions provide intermediate backout processing which DB2 does not support.)

In addition, the IMS extended restart service ('XRST') should not be invoked, nor should GSAM databases be used because Smart/RESTART restores working storage contents and repositions sequential files automatically. Nor should you code SYNC calls in the batch environment because IMS does not support them. Smart/PRECOMPILER can flag these statements, if they appear, and comment them out.

23.2.2 PCB Addressability

COBOL applications which access both IMS and DB2 resources should code a PROCEDURE DIVISION USING statement rather than the ENTRY DLITCBL USING construct more typical of IMS COBOL applications to address the list of program control blocks (PCBs) passed as parameters by IMS to your application program. This construction prevents the DB2 precompiler from inserting procedural statements prior to entry point DLITCBL. See the chapter on "Entry and Return Conventions" in IBM publication SH20-9026 for more information.

The IMS checkpoint function requires an IO-PCB. Your program's PCB address list must always reference an IO-PCB as the first PCB in the list. Be sure to specify CMPAT=YES on your application's PSBGEN statement.

NOTE: Even though calls to Smart/RESTART's CHECKPOINT and ROLLBACK services don't reference an IO-PCB, an IO-PCB must still be present in the PROCEDURE DIVISION USING list because IMS passes it to your program as a parameter. Smart/RESTART obtains the address of the IO-PCB directly from IMS for the IMS checkpoint call it issues internally.

23.2.3 Sequential File I/O

Smart/RESTART provides sequential file repositioning as one of its standard facilities. Thus, you needn't define sequential files as IMS GSAM databases in order to reposition them on restart. Instead, you can code standard COBOL file definitions and use native OPEN, CLOSE, READ and WRITE verbs to handle sequential file I/O.

23.3 Program Preparation for IMS / DB2 Applications

Program preparation for restartable IMS/DB2 applications requires the usual steps plus an optional (but recommended) step for Smart/PRECOMPILER. Installations that standardize on Smart/RESTART can revise their catalogued procedures to include this precompile step. This section describes the program preparation process in detail.

23.3.1 Run Smart/PRECOMPILER

If you want Smart/PRECOMPILER to restart-enable your application and conduct additional syntax and validity checks, add a jobstep to your program preparation JCL that runs Smart/PRECOMPILER. Smart/PRECOMPILER should run after the DB2 precompiler.

If you include Smart/PRECOMPILER in your program preparation jobstream, be sure to specify values for the precompiler's RM and CAF parameters. Code RM(IMS) if your application issues only DLI calls. Alternatively, specify RM (IMS,DB2) if your application accesses **both** IMS and DB2 data. In either case, code CAF(IMS) to identify IMS attach as the DB2 attachment facility.

23.3.2 Link Edit your Application

Your application must be link edited with the interface modules for both Smart/RESTART and IMS, unless the application was compiled with the COBOL DYNAM option (which requests dynamic linkage at execution time). Chapter 4.1.1 describes the STEPLIB concatenation required in this latter case.

The Smart/RESTART interface module is named RAIAPI. By link editing RAIAPI, you resolve any explicit calls you may have coded to the Smart/RESTART API as well as any calls introduced by Smart/PRECOMPILER. The IMS interface module is named DFSLI000. It provides the linkage for both SQL calls (through its alias DSNHLI) as well as IMS calls through its aliases CBLTDLI, PLITDLI, ASMTDLI, etc.

To link RAIAPI and DFSLI000 with your application program, you can code an explicit pair of linkage editor INCLUDE statements as illustrated in Figure 23.1 below:

```
//IMSDD DD DSN=IMSxxx.SRESLIB,DISP=SHR
//RAIDD DD DSN=DCAvrm.DCANCAL,DISP=SHR
//SYSLIN DD DSN=&OBJECT,DISP=(,PASS)
//      DD *
INCLUDE IMSDD(DFSLI000)
INCLUDE RAIDD(RAIAPI)
NAME    EXAMPLE(R)
//
```

Figure 23.1 *Link-Edit Jobstep*

Alternatively, you can rely on the linkage editor's automatic library call mechanism. In this case, be sure to concatenate the IMS library before the Smart/RESTART library, as illustrated Figure 23.2 below:

```
//SYSLIB DD DSN=IMSxxx.SRESLIB,DISP=SHR
//      DD DSN=DCAvrm.DCANCAL,DISP=SHR
//SYSLIN DD DSN=&OBJECT,DISP=(,PASS)
//      DD *
```

Figure 23.2 *Link-Edit Jobstep*

NOTE: If the linkage editor's SYSLIB concatenation is incorrect, the wrong DB2 attachment package (module DSNHLI) will be linked with your application.

23.3.3 Bind the Smart/RESTART DBRM into your DB2 application plan

Both IMS and mixed IMS/DB2 applications should bind the Smart/RESTART package or DBRM into their DB2 application plans as described in Chapter 3.6.

***NOTE:** Even if your application issues no SQL statements, it still requires a DB2 application plan because Smart/RESTART records its checkpoint synchronization information in a DB2 table. The BIND must include the SRSDBRM package or DBRM, even if no application packages or DBRMs are required. Moreover, DB2 must be active when your application runs in order for Smart/RESTART to operate properly.*

23.3.4 Converting IMS Extended Restart applications to Smart/RESTART

The following discussion describes how to convert an existing IMS COBOL application to operate with Smart/RESTART. The existing application is assumed to make use of IMS Extended Restart facilities. The converted application continues to call IMS to checkpoint or rollback database data. However, the converted application will use native COBOL verbs to perform I/O to repositionable, sequential files .

1. Convert IMS XRST calls to calls to the Smart/RESTART INITIALIZE service.

The IMS XRST call provides IMS with the addresses and lengths of the work areas that the application needs to save at checkpoint time in order to effect a successful restart. In addition, the IMS XRST call provides the application with the feedback required to determine whether the application is currently executing an initial or restart run.

The XRST call (and its associated restart determination logic) must be converted to a call to the Smart/RESTART INITIALIZE service. The same areas and lengths used in the XRST call should be supplied to the INITIALIZE service. The examination of the I/O area in the XRST call is replaced by a check of the Smart/RESTART field named FCA-STAT. During an initial run, FCA-STAT is blanks while during a restart run the area is set to "REST".

The success or failure of the call may be determined through a check of the COBOL RETURN-CODE register or by examining the FCA-RET-CODE area.

2. Convert IMS GSAM Database calls to native COBOL I/O Verbs.

When using IMS extended restart, all repositionable sequential files must be defined with GSAM DBDs and PCBs. During a XRST call, IMS reestablishes file position through the GSAM PCB feedback areas saved during the checkpoint call. Smart/RESTART can reposition standard sequential files without the use of GSAM.

- a. For each GSAM data base, define a COBOL FD that describes the file using the attributes from the DBDGEN and the JCL for the GSAM file.

- b. Convert each GSAM call in the application to a standard COBOL READ or WRITE statement. Follow standard COBOL rules to OPEN the file before the first READ or WRITE request.

3. Add a call to the Smart/RESTART TERMINATE service before each GOBACK or STOP RUN.

A call to the Smart/RESTART TERMINATE service must be inserted prior to each COBOL GOBACK or STOP RUN statement in the application. Smart/RECOM-PILER automatically inserts this statement during program preparation.

4. The PSB should be changed to remove the GSAM PCBs while retaining the I/O PCB and any other non-GSAM PCBs.

Since the GSAM definitions are no longer required, they should be removed and a new PSBGEN performed.

5. Convert IMS CHKPT/ROLB requests to calls to the Smart/RESTART CHECKPOINT service.

The Smart/RESTART CHECKPOINT service does not require the I/O PCB to be specified as a parameter. The CHECKPOINT service saves working storage areas and sequential file position and also releases IMS and DB2 locks. Smart/RESTART directly invokes the IMS basic checkpointing facility to perform the DBMS checkpointing function. Thus, IMS still functions as the commit coordinator for itself and DB2.

When requesting a rollback, Smart/RESTART restores working storage and file position. Next, it invokes IMS to drive the IMS/DB2 rollback process.

**6. The JCL must be modified for Smart/RESTART operation.
The following JCL changes may be required:**

- a. Add an SRSCHECK DD statement to define a checkpoint file. This is required when directing checkpoint data to a VSAM LDS (the default). This file is not required in an IMS/DB2 application when the application requests that all checkpoint data be written to a DB2 table.
- b. Add the RAINPUT DD * statement to allow the specification of run-time parameters.
- c. Remove any EXEC statement PARM fields relating to IMS restart (e.g. 'LAST' or the ID of the checkpoint from which to restart). Instead, these parameters are specified in the RAINPUT stream.
- d. Add the Smart/RESTART run-time load libraries to the STEPLIB concatenation for the job step.

7. Program Preparation.

- a. Both DLI and BMP environments are supported. Since IMS/DB2 support depends on the use of a DB2 checkpoint table, the IMS application must be

bound with a DB2 plan -- even if the application issues no SQL requests. The SRSDBRM package or DBRM supplied with Smart/RESTART must be included in the DB2 BIND. This module accesses DB2 through the DFSLI000 interface.

- b. When using Smart/PRECOMPILER, ensure that it runs after the DB2 precompiler.
- c. An I/O PCB must still be included to satisfy the requirements of IMS basic checkpoint which Smart/RESTART uses as part of its own checkpointing process.

8. Restrictions.

GSAM databases may only be accessed sequentially. Direct access via the IMS RSA is not supported.

23.4 Execution JCL for IMS and Mixed IMS / DB2 applications

Smart/RESTART applications which access IMS (or IMS and DB2) using database PCBs can run either as DLIBATCH jobs or as IMS BMPs.

The JCL for a DLIBATCH job is somewhat more complex than for a BMP because a DLI batch address space is independent of the IMS control region. Therefore, a great deal of IMS code is running along with your application. You must define an IMS Log file and provide DD statements for the datasets of any IMS databases your application will access. Lastly, the DB2 DLI Batch Support component has requirements of its own (which are discussed in Section 23.4.2 below).

This section discusses the JCL considerations common to both DLIBATCH and BMP environments, and then addresses the unique requirements of DLIBATCH jobs. Section 23.7 illustrates a sample DLIBATCH jobstream while Section 23.8 includes sample JCL for an IMS BMP.

23.4.1 STEPLIB Concatenation

The STEPLIB concatenation for your IMS/DB2 applications should include the datasets containing the IMS, DB2 and Smart/RESTART system load modules.

If your Smart/RESTART application was compiled with the COBOL DYNAM option, then the system must be able to load the module named DFSLI000 from the IMS library and the module RAIAPI from the Smart/RESTART load library. Be sure the IMS load library is defined *before* the Smart/RESTART load library in the STEPLIB concatenation.

23.4.2 DLIBATCH Considerations

In order to access IMS and DB2 in a DLIBATCH jobstep, you must run the DB2 DLI Batch Support program named DSNMTV01 as a front-end to your application. The DSNMTV01 program reads from an input file named DDITV02 and writes messages and diagnostic information to an optional DDOTV02 DD statement. The DB2 Application Programming Guide discusses these two files in detail in the section which describes how to process DLI batch applications.

DLIBATCH jobstreams should specify BKO=Y as a parameter of the DLIBATCH procedure and define the IMS log file as a disk dataset. Otherwise, IMS cannot dynamically backout uncommitted changes to DLI data. Backout occurs when either your application requests a rollback, the Smart/RESTARTabend routine does so or IMS itself issues a pseudo abend internally.

NOTE: *If BKO=N is specified or the IMS log is defined as a tape file, then the IMS batch backout utility must be submitted prior to restart in order to backout changes to DLI databases since the last commit point. Refer to the IMS Operating Procedures Manual for details.*

23.5 Smart/RESTART run-time parameters for IMS

Smart/RESTART run-time operation is governed by profile options. These may be supplied instream with the JCL (through the RAINPUT DD statement) or be defined as profile defaults when Smart/RESTART is installed. The following subsections discuss these options.

23.5.1 DB2 Attachment Facility Specification

The DB2 attachment facility in use is declared via the Smart/RESTART CAF parameter. IMS should be specified as the attachment package in both the IMS and mixed IMS / DB2 environment because IMS is responsible for coordinating recovery of DB2 and IMS resources. Anytime IMS will be accessed, the CAF operand should specify IMS as follows:

CAF(IMS)

NOTE: *This CAF parameter should correspond to the CAF parameter supplied to Smart/PRECOMPILER.*

23.5.2 Specify the Smart/RESTART Resource Manager Interface Module

Specify the Smart/RESTART resource manager interface module designed for the mixed IMS/DB2 environment. Its name is SRSDBRMI. Identify SRSDBRMI as the resource manager interface routine with the RMI keyword.

RMI(SRSDBRMI)

23.6 Sample Applications and Jobstreams

Smart/RESTART's DCASAMP dataset contains the COBOL source code for the two programs (IVPs) used to verify the product's installation. Member SRS8CIPT illustrates a 'precompiler transparent' application in which no explicit calls are made to Smart/RESTART. The SRS8CIPT program must be precompiled by Smart/PRECOM-PILER to be restart-enabled. In contrast, the SRS8CICB module is restart enabled via COBOL COPY text.

Section 23.7 shows the COBOL source code for SRS8CIPT along with illustrative JCL to run the application as both a DLIBATCH job and as an IMS BMP.

23.7 Sample DB2 / IMS COBOL Application

```
ID DIVISION.
PROGRAM-ID.    SRS8CIPT.
AUTHOR.        RAI.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-3090.
OBJECT-COMPUTER. IBM-3090.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT OUTPUT-FILE ASSIGN TO OUTFILE
    FILE STATUS IS OUTPUT-FILE-STATUS.

DATA DIVISION.
FILE SECTION.
FD  OUTPUT-FILE
    LABEL RECORDS STANDARD
    RECORDING MODE F
    BLOCK CONTAINS 0 RECORDS.
01  OUTPUT-RECORD  PIC  X(80).

WORKING-STORAGE SECTION.

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC SQL DECLARE CSR CURSOR FOR
        SELECT COLNO, NAME, TBNAME, TBCREATOR, COLTYPE
        FROM SYSIBM.SYSCOLUMNS
        WHERE TBNAME = 'SYSCOLUMNS'
        AND TBCREATOR = 'SYSIBM'
        ORDER BY COLNO
    END-EXEC.

*   Define SQL host variables
```

```

01 HOST-VARIABLES.
05 HV-COLNO          PIC S9(5) COMP.
05 FILLER            PIC X(05) VALUE SPACES.
05 HV-NAME           PIC X(18).
05 FILLER            PIC X(05) VALUE SPACES.
05 HV-TBNAME         PIC X(18).
05 FILLER            PIC X(05) VALUE SPACES.
05 HV-TBCREATOR      PIC X(8).
05 FILLER            PIC X(05) VALUE SPACES.
05 HV-COLTYPE        PIC X(8).
05 FILLER            PIC X(4)  VALUE SPACES.

*   Define the IMS segment

01 IMS-SEGMENT.
05 IMS-KEY            PIC 9(6).
05 IMS-DATA.
10 IMS-COLNO         PIC S9(5) COMP.
10 FILLER            PIC X(01) VALUE SPACES.
10 IMS-RUN-TYPE      PIC X(01) VALUE SPACES.
10 FILLER            PIC X(03) VALUE SPACES.
10 IMS-NAME          PIC X(18).
10 FILLER            PIC X(05) VALUE SPACES.
10 IMS-TBNAME        PIC X(18).
10 FILLER            PIC X(05) VALUE SPACES.
10 IMS-PROGRAM-NAME  PIC X(8).
10 FILLER            PIC X(05) VALUE SPACES.
10 IMS-COLTYPE       PIC X(8).
10 FILLER            PIC X(1)  VALUE SPACES.

01 IMS-SSA.
05 SEGMENT-NAME      PIC X(8)  VALUE 'SRSSEG '.
05 FILLER            PIC X      VALUE ' '.

*   Define this program's vital working storage.
*   Group critical fields within a single structure
*   whose contents will be saved at checkpoint time.
*   The values of these fields will be restored at restart time.

01 VITAL-STORAGE.
05 UOW-COUNT         PIC S9(4) COMP VALUE 0.
05 FETCH-COUNT       PIC S9(4) COMP VALUE 0.
05 IMS-KEY-BASE.
10 IMS-KEY-TIME      PIC S9(8) COMP VALUE +0.
10 IMS-SEGMENT-COUNTER PIC S9(4) COMP VALUE +0.

01 NON-CRITICAL-FIELDS.
05 ERROR-FLAG        PIC X(1) VALUE SPACE.
88 SQL-ERROR          VALUE 'S'.
88 IMS-ERROR          VALUE 'I'.
88 PROCESSING-ERROR   VALUES ARE 'S' 'I'.
05 QUERY-FLAG        PIC X(1) VALUE SPACE.
88 END-OF-QUERY       VALUE 'E'.
05 RUN-TYPE          PIC X(1) VALUE SPACE.
88 NORMAL-RUN         VALUE 'N'.
88 RESTART-RUN        VALUE 'R'.
05 CHARACTER-FIELD    PIC X(04) VALUE SPACES.
05 NUMERIC-FIELD      REDEFINES
CHARACTER-FIELD       PIC S9(4).
05 PGM-RETURN-CODE    PIC S9(4) COMP VALUE 0.
05 DISPLAY-SQLCODE    PIC -9(9).
05 OUTPUT-FILE-STATUS PIC 9(2)  VALUE ZEROS.
05 START-TIME         PIC 9(8).
05 PROGRAM-NAME       PIC X(8)  VALUE 'SRS8CRIPT'.

```

```

05  IMS-ISRT                      PIC  X(4)  VALUE 'ISRT'.
05  IMS-CHKP                      PIC  X(4)  VALUE 'CHKP'.
05  CHKP-ID                      PIC  X(8)  VALUE SPACES.

01  PGM-START-TIME.
05  DATE-CURRENT                 PIC 99/99/99.
05  TIME-CURRENT                 PIC 99/99/99PP.
05  DATE-INITIAL                 PIC 99/99/99.
05  TIME-INITIAL                 PIC 99/99/99PP.

```

LINKAGE SECTION.

```

01  IO-PCB.
05  IO-PCB-TERMNAME             PIC X(8).
05  FILLER                      PIC XX.
05  IO-PCB-STATUS               PIC XX.
05  IO-PCB-PREFIX.
10  FILLER                      PIC X.
10  IO-PCB-DATE                 PIC S9(5) COMP-3.
10  IO-PCB-TIME                 PIC S9(7) COMP-3.
10  FILLER                      PIC X(4).

01  DB-PCB.
05  DB-PCBNAME                  PIC X(8).
05  FILLER                      PIC XX.
05  DB-PCB-STATUS               PIC XX.
05  DB-PCB-PREFIX.
10  FILLER                      PIC X.
10  DB-PCB-DATE                 PIC S9(5) COMP-3.
10  DB-PCB-TIME                 PIC S9(7) COMP-3.
10  FILLER                      PIC X(4).

```

PROCEDURE DIVISION USING IO-PCB DB-PCB.

```

*   In Smart/RESTART's transparent mode of operation,
*   your application can distinguish between the initial and
*   restart environments on the basis of the value of the
*   COBOL RETURN-CODE special register. Since the value of
*   RETURN-CODE is volatile (may be lost) your program should
*   check this value immediately.

```

```

IF RETURN-CODE = +2001 THEN
    MOVE 'R' TO RUN-TYPE
ELSE
    IF RETURN-CODE = ZERO
        MOVE 'N' TO RUN-TYPE
    ELSE
        DISPLAY 'SRS8CIPT - Initialization error detected'
        GOBACK.

```

```

PERFORM A100-INITIALIZE THRU A100-EXIT.

```

```

PERFORM B100-UNIT-OF-WORK THRU B100-EXIT
    UNTIL UOW-COUNT > 9
    OR    PROCESSING-ERROR.

```

```

DISPLAY 'SRS8CIPT - Processing completed'.
MOVE PGM-RETURN-CODE TO RETURN-CODE.
GOBACK.

```

```

A100-INITIALIZE.
    DISPLAY SPACE.
    DISPLAY 'SRS8CIPT - Sample COBOL IMS/DB2 application'.

```

```

IF RESTART-RUN THEN
    DISPLAY 'SRS8CIPT - Detected the RESTART environment'
ELSE
    DISPLAY 'SRS8CIPT - Original run of the program'
    ACCEPT DATE-INITIAL FROM DATE
    ACCEPT TIME-INITIAL FROM TIME
    ACCEPT START-TIME FROM TIME
    MOVE START-TIME TO IMS-KEY-TIME.

ACCEPT DATE-CURRENT FROM DATE.
ACCEPT TIME-CURRENT FROM TIME.

DISPLAY 'SRS8CIPT - Current Run Date = ' DATE-CURRENT
        ' : Run Time = ' TIME-CURRENT.

IF RESTART-RUN THEN
    DISPLAY 'SRS8CIPT - Initial Run Date = ' DATE-INITIAL
        ' : Run Time = ' TIME-INITIAL.

OPEN OUTPUT OUTPUT-FILE.
IF OUTPUT-FILE-STATUS > 0 THEN
    DISPLAY 'OUTPUT ERROR'
    MOVE 'A' TO ERROR-FLAG
    MOVE 8 TO PGM-RETURN-CODE.

A100-EXIT.
EXIT.

/
*   Each unit of work fetches each row of the SQL query result.
*   For each row fetched from the SQL query result
*   insert the corresponding row into an IMS database.

B100-UNIT-OF-WORK.
ADD 1 TO UOW-COUNT.

DISPLAY 'SRS8CIPT - Commencing new logical unit of work '
        UOW-COUNT.

EXEC SQL OPEN CSR END-EXEC.
IF SQLCODE NOT = 0 THEN
    PERFORM B400-SQL-ERROR THRU B400-EXIT
    GO TO B100-EXIT.
MOVE SPACE TO QUERY-FLAG.

*   Fetch each row of the SQL query result and insert the data
*   from that row into an IMS database. One SQL result row and
*   one IMS segment represents our individual transaction,
*   several of which are bunched together in this batch
*   application to comprise the unit of work.

PERFORM B200-PROCESS-ROW THRU B200-EXIT
    UNTIL END-OF-QUERY
    OR PROCESSING-ERROR.

IF PROCESSING-ERROR GO TO B100-EXIT.

EXEC SQL CLOSE CSR END-EXEC.
IF SQLCODE NOT = 0 THEN
    PERFORM B400-SQL-ERROR THRU B400-EXIT
    GO TO B100-EXIT.

*   Commit the unit of work via an IMS basic or symbolic
*   checkpoint call. Smart/RESTART ignores the application

```

```

*      supplied checkpoint ID.

      CALL 'CBLTDLI' USING IMS-CHKP
                          IO-PCB
                          CHKP-ID.

      IF IO-PCB-STATUS NOT EQUAL TO SPACES THEN
        PERFORM B600-IMS-ERROR THRU B600-EXIT
        GO TO B100-EXIT.

B100-EXIT.
EXIT.

B200-PROCESS-ROW.
EXEC SQL FETCH CSR INTO
      :HV-COLNO,
      :HV-NAME,
      :HV-TBNAME,
      :HV-TBCREATOR,
      :HV-COLTYPE
END-EXEC.

      IF SQLCODE = 100 THEN
        MOVE 'E' TO QUERY-FLAG
        GO TO B200-EXIT.

      IF SQLCODE < 0 THEN
        PERFORM B400-SQL-ERROR THRU B400-EXIT
        GO TO B200-EXIT.

      ADD 1 TO FETCH-COUNT.
      ADD 1 TO IMS-SEGMENT-COUNTER.

*      Write the host variable structure containing the SQL result
*      row to a sequential output file. This sequential output
*      file will be properly repositioned on restart, without
*      defining it as a GSAM database.

      MOVE IMS-KEY-BASE TO IMS-KEY.
      MOVE HV-COLNO     TO IMS-COLNO.
      MOVE HV-NAME      TO IMS-NAME.
      MOVE HV-TBNAME    TO IMS-TBNAME.
      MOVE PROGRAM-NAME TO IMS-PROGRAM-NAME.
      MOVE RUN-TYPE     TO IMS-RUN-TYPE.
      MOVE HV-COLTYPE   TO IMS-COLTYPE.

      WRITE OUTPUT-RECORD FROM IMS-SEGMENT.

*      Next, insert the host variable structure containing the SQL
*      result row into an IMS database.

      CALL 'CBLTDLI' USING IMS-ISRT
                          DB-PCB
                          IMS-SEGMENT
                          IMS-SSA.

      IF DB-PCB-STATUS EQUAL TO 'II' THEN
        DISPLAY 'SRS8CIPT - Duplicate record encountered'
        DISPLAY 'SRS8CIPT - Delete/Define & reload the database'
        .

      IF DB-PCB-STATUS NOT EQUAL TO SPACES THEN
        PERFORM B500-IMS-ERROR THRU B500-EXIT

```



```

        GO TO B200-EXIT.

*      Is it time to deliberately abend the original run
*      of the program?

        IF NORMAL-RUN AND FETCH-COUNT = 120
            DISPLAY 'SRS8CIPT - A deliberate SOC7 abend after 120'
              ' records'

*      This statement will produce a deliberate SOC7 abend

        ADD 1 TO NUMERIC-FIELD.
B200-EXIT.
        EXIT.

B400-SQL-ERROR.
        DISPLAY 'SRS8CIPT - ERROR ON SQL CALL'.
        DISPLAY 'SRS8CIPT - SQL COMMUNICATIONS AREA'
        MOVE SQLCODE TO DISPLAY-SQLCODE.
        DISPLAY 'SQLCODE =' DISPLAY-SQLCODE.
        DISPLAY 'SQLCA =' SQLCA.
        MOVE 'S' TO ERROR-FLAG.
        MOVE 12 TO PGM-RETURN-CODE.
B400-EXIT.
        EXIT.

B500-IMS-ERROR.
        DISPLAY 'SRS8CIPT - ERROR ON DLI CALL REFERENCING DB-PCB'.
        DISPLAY 'SRS8CIPT - DB-PCB STATUS CODE = ' DB-PCB-STATUS.
        MOVE 'I' TO ERROR-FLAG.
        MOVE 12 TO PGM-RETURN-CODE.
B500-EXIT.
        EXIT.

B600-IMS-ERROR.
        DISPLAY 'SRS8CIPT - ERROR ON DLI CALL REFERENCING IO-PCB'.
        DISPLAY 'SRS8CIPT - IO-PCB STATUS CODE = ' IO-PCB-STATUS.
        MOVE 'I' TO ERROR-FLAG.
        MOVE 12 TO PGM-RETURN-CODE.
B600-EXIT.
        EXIT.

```

Figure 23.3 *Sample DB2 / IMS COBOL Application*

23.8 Sample DLIBATCH JCL

```
//          JOB
//*-----

/*
/*      Invoke the standard DLIBATCH cataloged procedure
/*      from the IMS procedure library
/*
/*      JCL parameter and DD statement overrides to
/*      operate in a Smart/RESTART mixed environment
/*
/*-----
//SRS8CIPT EXEC DLIBATCH,
//          MBR=DSNMTV01,      <---- IBM front end to set up IMS attach
//          PSB=SRSSAMPU,
//          DBRC=N,            <--- DBRC not active
//          LOGT=SYSDA,
//          BKO=Y,            <--- Dyn batch backout works off IMS disk
log
//          IRLM=N,
//          REGION=4M
//G.IMS      DD DSN=DCAvrms.DCAPSB,DISP=OLD
//          DD DSN=DCAvrms.DCADBD,DISP=OLD
//G.IEFRDER DD DSN=DCAvrms.DCAIMSL,DISP=OLD,VOL=
/*
/*      Additional DD statements added to the DLIBATCH procedure
/*
//STEPLIB DD DISP=SHR,DSN=IMS.RESLIB
//          DD DISP=SHR,DSN=DCAvrms.DCALOAD
//          DD DISP=SHR,DSN=DSN.SDSNEXIT
//          DD DISP=SHR,DSN=DSN.SDSNLOAD
//          DD DISP=SHR,DSN=CEE.SCEERUN
/*-----
/*
/*      DD statement DDITV02 supplies IMS attachment parameters
/*      to the DB2 DLI batch support component.
/*      Note: the connection name is defaulted to the jobname
/*
/*-----
//DDITV02 DD *                      /* IMS attachment parameters
//          DSN7,SYSL,DSNMIN10,,A,-,,,SRS8CIPT,SRS8CIPT
/*-----
/*
/*      The DB2 DLI batch support component
/*      writes output to file DDOTV02
/*
/*-----
//DDOTV02 DD DSN=&TEMP,DISP=(NEW,PASS),SPACE=(TRK,(1,1)),
//          UNIT=SYSDA,DCB=(RECFM=VB,BLKSIZE=4096,LRECL=4092)
/*-----
/*
/*      IMS HISAM data base (underlying dataset is a VSAM KSDS)
/*
/*-----
//SRSDBI1 DD DISP=SHR,DSN=DCAvrms.SRS8CI.KSDSFILE
/*-----
/*
/*      IMS VSAM options
/*
/*-----
//DFSVSAMP DD *
```

```

VSRBF=2048,5
/**
/**      Application's sequential output file
/**
//OUTFILE DD DISP=OLD,DSN=DCAvrms.OUTFILE2
//SYSUDUMP DD DUMMY
//SYSABOUT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
/**-----
/**
/**      Smart/RESTART checkpoint dataset
/**
/**-----
//SRSCHECK DD DISP=SHR,DSN=DCAvrms.DCACKPTV
//SRSPRINT DD SYSOUT=* Smart/RESTART message file
/**
/**      Profile parameters specified explicitly through the RAINPUT
/**      file below override the corresponding site defined defaults
/**
//RAINPUT DD *
*
* Smart/RESTART related profile commands follow
*
CLEAR_CHKPT(BEFORE) CKPT_FREQ(1) MSGDD(SRSPRINT)
VERIFY_JID(NO) MSG_DISPLAY(DEBUG) ESTAE(ON)
REPOSITION(ONLY,(OUTFILE))
RMI(SRSDBRMI)

```

Figure 23.4 Sample DLIBATCH JCL

23.9 Sample JCL for an IMS BMP

```
//IMSBMP JOB ...
//*------
//*          Invoke standard IMSBATCH cataloged procedure
//*          from the IMS procedure library
//*-----
//STEP1      EXEC IMSBATCH,
//            MBR=SRS8CIPT,                <--- (1)
//            PSB=SRSSAMPU                 <--- (2)
//G.STEPLIB DD DSN=APPL.PGM.LOAD,DISP=SHR
//            DD DSN=DCAvrm.DCALOAD,DISP=SHR
//            DD DSN=DSNxxx.SDSNLOAD,DISP=SHR
//            DD DSN=IMSxxx.RESLIB,DISP=SHR
//*
//SRSCHECK DD DISP=SHR,DSN=DCAvrm.DCACKPTV
//OUTFILE DD DSN=DCAvrm.OUTFILE1,DISP=SHR
//SYSABOUT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//RAINPUT DD *
//            RMI(SRSDBRMI)                <--- (3)
//
```

where

- (1) SRS8CIPT identifies the name of the application load module
- (2) SRSSAMPU identifies the name of the PSB to use with the SRS8CIPT application
- (3) The RMI(SRSDBRMI) parameter identifies the Smart/RESTART Resource Manager Interface routine to use in this IMS BMP.

Figure 23.5 Sample JCL for an IMS BMP

Chapter 24

Smart/RESTART Support for

WebSphere MQ

This chapter documents Smart/RESTART support for WebSphere MQ for z/OS. The first section addresses the requirements and restrictions associated with restartable applications which access MQ resources in ‘standalone mode’. The next section describes the run-time requirements and illustrates the JCL needed for restartable WebSphere MQ applications. The last section describes how applications can explicitly manage their connection to WebSphere MQ rather than let Smart/RESTART connect and disconnect implicitly.

24.1 Application Restrictions and Requirements

In WebSphere MQ ‘standalone mode’, Smart/RESTART provides only single phase COMMIT support for MQ resources. This means Smart/RESTART can keep your application’s working storage contents, VSAM data and sequential file position in synch with any resources managed by WebSphere MQ for z/OS. Restartable applications that access DB2, IMS and/or other RRS compliant resources along with MQ resources within a single unit-of-work must run Smart/RESTART in RRS mode as described in Chapter 20.

Restartability also requires that any recoverable messages GET or PUT by your application must participate in the unit-of-work managed by WebSphere MQ. Specifically:

GET messages should specify MQGMO_SYNCPOINT in their Get-Message Options structure.

PUT messages should specify MQPMO_SYNCPOINT in their Put-Message Options structure. In addition, their Message Descriptor structure should specify MQPER_PERSISTENT as the persistence attribute.

In addition, the typical restartable WebSphere MQ application should neither explicitly connect nor explicitly disconnect from the local queue manager. However, see the section titled Application Managed CONNECT and DISCONNECT later in this chapter. At initialization time, Smart/RESTART implicitly issues an MQCONN call to connect to the local queue manager. Smart/RESTART also disconnects implicitly from the local queue manager when your application terminates (either normally or abnormally).

Once Smart/RESTART is initialized, it makes the connection handle (representing a connection to the queue manager) available in one of the following FCA variables (depending on the language in which your application is coded). Use this connection handle on all subsequent message queuing calls.

COBOL: (member SRSCFCA of DCASAMP)

```
05     FCA-MQ-CONNECT-HANDLE     PIC   S9(9) COMP.
```

PL/1: (member SRSPFCA of DCASAMP)

```
2     FCA_MQ_CONNECT_HAND     FIXED BIN(31),
```

Assembler Language: (member SRS@FCA of DCASAMP)

```
FCAMQCHN     DS     F
```

Restartable WebSphere MQ applications must contain the requisite RAI API service requests in order to function correctly. For example, your applications must invoke Smart/ RESTART to initialize and terminate the environment. This can be accomplished by explicitly coding calls to the INITIALIZE and TERMINATE services. Alternatively, you can let Smart/PRECOMPILER embed the necessary INITIALIZE and TERMINATE calls to restart enable your application.

In contrast, you should always call the Smart/RESTART CHECKPOINT service or SRRCMIT **manually** when the unit-of-work completes successfully. Similarly, call Smart/RESTART's ROLLBACK service or SRRBACK when the unit-of-work needs to be backed out. **Never call MQCMIT or MQBACK directly.** Instead, call the Smart/RESTART CHECKPOINT and ROLLBACK services or the SRRCMIT and SRRBACK routines to respectively commit or rollback WebSphere MQ resources and keep them in synch with your application's working storage contents, VSAM data and sequential file position.

NOTE: *Smart/PRECOMPILER does not automatically convert WebSphere MQ calls to Smart/RESTART calls.*

Member SRS8DBMQ of the DCASAMP library provides an annotated example of a restartable WebSphere MQ application written in COBOL. Member SRSBDBMQ provides sample JCL with which to build the SRS8DBMQ application. Finally, member SRSRDBMQ provides JCL to execute the SRS8DBMQ application.

```
//RAINPUT DD *  
RMI(SRSDMQS)                <--- (1)  
QMGR(MQS1)                  <--- (2)  
QNAME(SMART.RESTART.CHECKPOINT.QUEUE) <--- (3)
```

where

- (1) The RMI operand should specify a value of 'SRSDMQS'. This is the name of the Smart/RESTART resource manager interface routine designed for WebSphere MQ for z/OS.
- (2) The QMGR operand identifies the queue manager to which both Smart/RESTART and your application will connect. Any queue manager defined here overrides the default queue manager name specified when Smart/RESTART was installed.
- (3) The QNAME operand identifies the permanent queue in which Smart/RESTART maintains its synchronizing information. Any queue name defined here overrides the default queue name specified when Smart/RESTART was installed.

Figure 24.1 *Run-time parameters supplied to Smart/RESTART*

24.2 Linking the WebSphere MQ interface routine

Modules which call WebSphere MQ services must link-edit a stub program that provides access to WebSphere MQ. Section 24.2.1 describes and illustrates how to link the MQ stub intended for the RRS environment while Section 24.2.2 discusses how to link the CSQBSTUB routine intended for applications which access WebSphere MQ resources in 'standalone mode'.

NOTE: *If your program calls all its subprograms dynamically (e.g. a COBOL module compiled with the DYNAM option), you can skip this Section entirely. These considerations are addressed in Chapter 4 which discusses execution JCL.*

24.2.1 Linking the MQ interface routine for the RRS Environment

Figure 24.2 illustrates a link edit jobstep for the restartable WebSphere MQ application program named EXAMPLE that accesses RRS compliant resources *in addition to* those of WebSphere MQ for z/OS. The linkage editor is directed to explicitly include the interface routines required by Smart/RESTART and WebSphere MQ (in an RRS environment).

```
//LINK      EXEC PGM=IEWL
//SYSLIB    DD DSN=automatic.call.library,DISP=SHR
//RAILIB    DD DSN=DCAvrml.DCANCAL,DISP=SHR          <--- (1)
//MQLIB     DD DSN=csq.SCSQLOAD,DISP=SHR            <--- (2)
//SYSLIN    DD DSN=&OBJSET,DISP=(OLD,DELETE)         <--- (3)
//          DD *
            INCLUDE RAILIB(RAIAPI)                  <--- (4)
            INCLUDE MQLIB(CSQBRSTB)                 <--- (5)
            INCLUDE RAILIB(SRS4SRR)                 <--- (6)
            NAME      EXAMPLE(R)
//
```

where

- (1) The RAILIB DD statement defines the Smart/RESTART system library which contains the Smart/RESTART interface routine. Ask your Smart/RESTART product administrator for the name of this dataset in your environment.
- (2) The MQLIB DD statement defines the WebSphere MQ system library which contains the WebSphere MQ batch stub program named CSQBRSTB. Ask your WebSphere MQ administrator for the name of this dataset in your environment.
- (3) The SYSLIN file identifies the object module named EXAMPLE which was produced by a compiler or the system assembler.
- (4) The first INCLUDE control statement directs the linkage editor to fetch the load module named RAIAPI from the dataset defined by the RAILIB DD statement and incorporate it in the executable load module.
- (5) The second INCLUDE control statement directs the linkage editor to fetch the load module named CSQBRSTB from the dataset defined by the MQLIB DD statement and incorporate it in the executable load module. CSQBRSTB is the name of the batch stub program supplied with WebSphere MQ for z/OS that is intended for use when RRS compliant resources in addition to MQ are accessed.
- (6) Applications which update resources managed by multiple RRS compliant products (such as DB2, WebSphere MQ and IMS) within a single unit-of-work should specify a third INCLUDE control statement to direct the linkage editor to fetch the load module named SRS4SRR from the dataset defined by the RAILIB DD statement and incorporate it in the executable load module. See Section 3.5.1 and Chapter 20 of this publication for details.

Figure 24.2 Smart/RESTART Link-Edit Jobstep

24.2.2 Linking the MQ interface for the Standalone MQ Environment

Figure 24.3 illustrates a link edit jobstep for the restartable WebSphere MQ application program named EXAMPLE that accesses WebSphere MQ resources in standalone mode. The linkage editor is directed to explicitly include the interface routines required by Smart/RESTART and WebSphere MQ (in standalone mode).

```
//LINK      EXEC PGM=IEWL
//SYSLIB    DD DSN=automatic.call.library,DISP=SHR
//RAILIB    DD DSN=DCAvrn.DCANCAL,DISP=SHR          <--- (1)
//MQLIB     DD DSN=csq.SCSQLOAD,DISP=SHR           <--- (2)
//SYSLIN    DD DSN=&OBJSET,DISP=(OLD,DELETE)        <--- (3)
//          DD *
            INCLUDE RAILIB(RAIAPI)                  <--- (4)
            INCLUDE MQLIB(CSQBSTUB)                 <--- (5)
            NAME      EXAMPLE(R)
//
```

where

- (1) The RAILIB DD statement defines the Smart/RESTART system library which contains the Smart/RESTART interface routine. Ask your Smart/RESTART product administrator for the name of this dataset in your environment.
- (2) The MQLIB DD statement defines the WebSphere MQ system library which contains the WebSphere MQ batch stub program named CSQBSTUB. Ask your WebSphere MQ administrator for the name of this dataset in your environment.
- (3) The SYSLIN file identifies the object module named EXAMPLE which was produced by a compiler or the system assembler.
- (4) The first INCLUDE control statement directs the linkage editor to fetch the load module named RAIAPI from the dataset defined by the RAILIB DD statement and incorporate it in the executable load module.
- (5) The second INCLUDE control statement directs the linkage editor to fetch the load module named CSQBSTUB from the dataset defined by the MQLIB DD statement and incorporate it in the executable load module. CSQBSTUB is the name of the batch stub program supplied with WebSphere MQ for z/OS that is intended for applications which access MQ resources in standalone mode.

Figure 24.3 *Smart/RESTART Link-Edit Jobstep*

24.3 Run-time Requirements

The run-time requirements for restartable applications that access WebSphere MQ resources are illustrated in Figure 24.1. This figure shows an annotated example of the run-time parameters passed to Smart/RESTART (through the RAINPUT file).

24.4 Application Managed CONNECT and DISCONNECT

Typically, Smart/RESTART CONNECTs to WebSphere MQ during its initialization and issues an WebSphere MQ DISCONNECT during Smart/RESTART termination. This all occurs automatically. This section deals with those situations in which the application needs to *explicitly* manage the connection to WebSphere MQ itself and the application supplies Smart/RESTART with the connection handle it acquires. In this way, the application can issue requests for WebSphere MQ services *prior* to calling the Smart/RESTART INITIALIZE service in the application main module.

Applications can manage the WebSphere MQ connection themselves by proceeding as previously described in this chapter – with the following differences:

1. Specify QMGR(UNKNOWN) as a parameter in the RAINPUT file. This informs Smart/RESTART that the application will manage the WebSphere MQ connection itself. In this case, the application must issue an MQCONN request *before* it calls the Smart/RESTART INITIALIZE service.
2. The application must copy the connection handle returned by the MQCONN service into the field named FCA-MQ-CONNECT-HANDLE -- before it calls the Smart/RESTART INITIALIZE service. (For COBOL applications this field is defined within the copy member named SRSCFCA of the DCASAMP library).
3. When the application explicitly manages the WebSphere MQ connection, it must also take responsibility for disconnecting because Smart/RESTART will not disconnect at termination. In these cases, the application should issue an MQDISC request after the application issues its final call to the Smart/RESTART TERMINATE service.

Smart/MONITOR

30.1 Overview

This chapter introduces Smart/MONITOR and illustrates those features that perform collection, reporting, analysis and prediction. Smart/MONITOR helps you measure and tune your applications with respect to their SQL activity, procedural logic and file I/O. In addition, Smart/PREDICTOR estimates completion times and issues early warnings when a job is in danger of running too long. These warnings allow you to take remedial action to help meet production schedules.

Smart/MONITOR captures performance data (CPU and elapsed times) on SQL requests, application logic and repositionable file I/O. In addition, Smart/MONITOR maintains I/O counts by file name, execution counts by SQL statement and entry / exit counts by module name.

You can obtain summaries of SQL, file I/O and CPU activity in real-time and produce a variety of print reports for off-line analysis. Once collected, performance data can be massaged, summarized and reported very flexibly. Smart/MONITOR includes a sophisticated analysis component which makes performance tuning recommendations.

In addition to this chapter, the Smart/MONITOR documentation includes the following:

- Chapter 31 describes how to customize Smart/MONITOR operation to suit your preferences and tailor the list of restartable jobs it displays.
- Chapter 32 describes the Smart/MONITOR displays and how to navigate among them.
- Chapter 33 describes the Smart/MONITOR Primary Commands
- Chapter 34 describes and illustrates how you can browse and edit the storage associated with a restartable application -- in real-time, during program execution.
- Chapter 35 describes Smart/PREDICTOR, a Smart/MONITOR facility which graphically displays the progress and projected completion time of a batch job under observation.

30.1 How to establish a Smart/MONITOR on-line session

Smart/MONITOR requires several ISPF dialog libraries for its operation. These can be allocated via the following command:

```
EXEC 'hlq.DCAEXEC(DCAALLOC)' EX
```

where hlq represents the high level qualifier(s) of the Smart/RESTART REXX exec library. Once Smart/MONITOR's dialog libraries are allocated, first time users should issue a SRSMONP command from ISPF Option 6 as follows to copy and initialize their Smart/MONITOR profile:

```
SRSMONP
```

Once Smart/MONITOR's dialog libraries are allocated and the user's Smart/MONITOR profile is initialized, Smart/MONITOR itself can be invoked by issuing the following command from within ISPF:

```
SRSMON
```

Multiple Smart/MONITOR sessions can be established by repeatedly issuing the SRSMON command. In addition, multiple TSO users can establish concurrent monitor sessions to track the activity of restartable jobs.

NOTE: *The Smart Jobstream Series Installation and Customization Guide describes how to allocate the ISPF libraries that Smart/MONITOR requires.*

30.2 Operating Levels

Smart/MONITOR supports two modes of operation, referred to as **Basic** and **Extended**. They describe the sources from which Smart/MONITOR accesses data.

Basic Mode

In Basic Mode, data is obtained from one of the checkpoint synchronization structures defined when Smart/RESTART is installed. Sites typically define a **single** checkpoint structure to be shared by all Smart/RESTART applications. Alternatively, a site can create **multiple** checkpoint structures in order to improve performance and/or to logically segregate applications.

When a site operates with multiple Smart/RESTART checkpoint structures, the Smart/MONITOR CONFIG panel (described in Chapter 31) lets you direct observation to a particular checkpoint structure. Smart/MONITOR always displays the name of the **current** checkpoint structure above the scrollable list of jobs.

Basic mode lets you examine a jobstep's I/O activity and working storage as of the most recent checkpoint. You can even **SUSPEND** the execution of the jobstep while you examine file and/or working storage data.

Extended Mode

Smart/MONITOR's Extended Mode of operation adds the following capabilities:

Direct Memory Control: Working storage and I/O buffers for any module managed by Smart/RESTART can be viewed and altered in real-time.

Smart/RESTART Monitoring: can be activated to obtain detailed statistics on repositionable I/O as well as requests for Smart/RESTART services.

SQL Monitoring: SQL monitoring can be activated to display detailed information on an application's SQL activity.

30.3 Smart/MONITOR Scrollable Job Summary

The scrollable Job Summary panel illustrated in Figure 30.1 displays your restartable applications in sequence by jobname, start-date and start-time.

```
Smart/MONITOR vv.rr.mm      Smart/RESTART Job Queues      Row 1 to 7 of 7
Command ==>                Scroll ==> PAGE
_____Filtering is enabled. 11/13/2009 15:50 RAI021 SRM >Panel 1 of 4>
Checkpoint Synchronization DB2 Table Qualifier: DCA1010      DB2 SSID: D9G
_____SORT St-Date/D, St-Time/D
  RAI7      ALL      ALL ALL      ALL      ALL      ALL      ALL      ALL
Cmd Jobname StepName LPAR JesJobID JesJStat SyncStat ASID St-Date  St-Time
RAI7J8CV SRS8CV  Z7L2 JOB04119 UNKNOWN  RUNNING  N/A  2009-11-13 15.50.06
RAI7J8CM SRMIVP  Z7L1 JOB04116 ACTIVE   SUSPEND  00AF 2009-11-13 15.46.13
RAI7SMP3 SRSSAMP3 Z7L1 JOB04115 INACTIVE COMPLETE N/A  2009-11-13 15.45.01
RAI7J8CM SRMIVP  Z7L1 JOB04100 INACTIVE U2005   N/A  2009-11-13 13.47.40
RAI78CPT SRS8CPT  Z7L1 JOB04098 INACTIVE COMPLETE N/A  2009-11-13 13.46.36
RAI7298R P109298B Z7L1 JOB03049 INACTIVE SOC7   N/A  2009-10-26 14.29.35
RAI7J8CV SRS8CV  Z7L2 JOB03001 UNKNOWN  SOC7     N/A  2009-10-13 13.51.14
***** Bottom of data *****
```

Figure 30.1

You can restrict the list of jobs displayed on the scrollable Job Summary on the basis of such criteria as jobname, job status, start date and start time. You can tab directly to the underlined fields on the Job Summary panel and key in the desired value(s). Alternatively, Chapter 31 describes how you can specify job selection criteria via the CONFIG command.

Once the jobs of interest are displayed, you can scroll the Job Summary up, down, left and right via the ISPF scroll PF keys. You can also *drill down* within a particular job to display job related performance data. For example, you can display detail information on file I/O, checkpoint activity and SQL statistics. Chapter 32 describes and illustrates these

additional displays. In addition, HELP is always available -- by pressing PF1 -- as you navigate through the various monitor panels.

The following operands control one or more actively running Smart/RESTART jobs with a single command. They allow the <job_name> operand to be masked via the SQL wild characters _ and %:

FORce JOBNAME=<job_name>	Immediately cancel the processing of restartable jobs whose job name is like <job_name>
RESume JOBNAME=<job_name>	Resume processing of restartable jobs whose job name is like <job_name>
SUSpend JOBNAME=<job_name>	Suspend processing of restartable jobs whose job name is like <job_name>
QUIesce JOBNAME=<job_name>	Gracefully cancel at a Unit-of-Work boundary restartable jobs whose job name is like <job_name>
MODIFY <job_name>,<command> F <job_name>,<command>	Issue any valid Smart/RESTART run-time command against one or more restartable jobs using the standard MVS MODIFY interface

30.4 Job specific Row Commands

You can enter a variety of row selection commands on Smart/MONITOR's scrollable Job Summary panel. The row commands described in this section can be entered in the selection field immediately to the *left* of the job of interest. For example, Figure 30.2 illustrates how to request a display of SQL statistics for the job named RAI7J8CM.

```

Smart/MONITOR vv.rr.mm      Smart/RESTART Job Queues      Row 1 to 7 of 7
Command ==>              Scroll ==> PAGE
Filtering is enabled. 11/13/2009 15:50 RAI021 SRM >Panel 1 of 4>
Checkpoint Synchronization DB2 Table Qualifier: DCA1010      DB2 SSID: D9G
                                                                    SORT St-Date/D, St-Time/D
RAI7      ALL      ALL ALL      ALL      ALL      ALL      ALL      ALL
Cmd Jobname StepName LPAR JesJobID JesJStat SyncStat ASID St-Date St-Time
RAI7J8CV SRS8CV Z7L2 JOB04119 UNKNOWN RUNNING N/A 2009-11-13 15.50.06
SQL RAI7J8CM SRMIVP Z7L1 JOB04116 ACTIVE SUSPEND 00AF 2009-11-13 15.46.13
RAI7SMP3 SRSSAMP3 Z7L1 JOB04115 INACTIVE COMPLETE N/A 2009-11-13 15.45.01
RAI7J8CM SRMIVP Z7L1 JOB04100 INACTIVE U2005 N/A 2009-11-13 13.47.40
RAI78CPT SRS8CPT Z7L1 JOB04098 INACTIVE COMPLETE N/A 2009-11-13 13.46.36
RAI7298R P109298B Z7L1 JOB03049 INACTIVE SOC7 N/A 2009-10-26 14.29.35
RAI7J8CV SRS8CV Z7L2 JOB03001 UNKNOWN SOC7 N/A 2009-10-13 13.51.14
***** Bottom of data *****

```

Figure 30.2

An invalid command will display the Smart/MONITOR menu illustrated in Figure 30.3. From this menu, you can select a row command by entering the command, or placing the cursor on your choice and pressing ENTER to retrieve the command.

```

JobName=RAI7320R JesJobID=JOB04525 JobStep=P109320A SystemID=Z7L1 RunID=3
Command Input ==>

Smart/RESTART Job Control      Start Date Time : 2009-11-18 14.33.09
CAN . Phys CKPT next commit and abend Ckpt Date Time : 2009-11-18 14.33.27
FOR . Force immediate application abend Checkpoint ID : 0000037
QUI . Quiesce job at next paced CKPT DB2 Sync Table Job Status: COMPLETE
RES . Resume Smart/RESTART job JES Job status . . . . : INACTIVE
SUS . Suspend Smart/RESTART job RAI Server Address Space : RAI0

Active Smart/RESTART job only Requires access to checkpoint medium
AFS . Application File Summary FIL . Repositionable File I/O Summary
APS . Application Program Storage GOL . List application files
DET . Display Job/Jobstep Detail JOB . Display Job/Jobstep Detail
PRE . Smart/PREDICTOR STO . Application program Storage
PRM . Display run-time profile
PRO . Modify Smart/RESTART profile Resource Manager Synchronization Data
REC . Smart/RESTART Recovery Facility DEL . Delete Job Entry (JE)
RFS . Repositionable File I/O summary DRD . Delete JE and CKPT dataset (DA)
STA . Smart/RESTART JOB statistics ARC . Archive Job Entry(JE)
SQL . SQL Access Summary ARD . Archive JE and delete CKPT DA

HEL . Smart/MONITOR Online Reference EXI . Exit

```

Figure 30.3

The menu displayed in Figure 30.3 supports the following row commands (which are described in alphabetical order):

AFS	This command displays all currently open files. This includes both repositionable files as well as other files opened by the current MVS task ¹ .
APS	The APS command allows you to display and edit the storage areas that Smart/RESTART saves at checkpoint time. Furthermore, this command allows you to display and edit the complete program storage areas of a restartable COBOL program ¹ .
ARC	The ARC command allows an authorized user, whose TSO ID has access to the DB2_ARCHIVE Resource Access Definitions Facility (RADF) profile, to <i>archive</i> a DB2 synchronization row that corresponds to the currently selected job. See Chapter 9 of the <i>Smart Jobstream Series Installation and Customization Guide (publication SJS-001)</i> for a detailed description of RADF. Once the row is archived, a restart of the corresponding jobstep is no longer possible. Jobsteps that require a restart should not have their synchronization DB2 rows archived.
ARD	<i>Archive</i> a DB2 synchronization row that corresponds to the currently selected job and delete the corresponding checkpoint data set. See the description of the ARC command (above) for details associated with the <i>Archive</i> part of this multi-action command.
CANCEL	The CANCEL command directs Smart/RESTART to gracefully halt an application at a unit-of-work boundary. Smart/RESTART honors the application's next checkpoint request as a physical checkpoint event (thereby ignoring any checkpoint pacing criteria that may be active). Smart/RESTART abends the application once the physical checkpoint is complete. This enables the jobstep to resume execution at a more propitious time -- with no wasted processing ^{1,3} .
DELeTe	The DELeTe command allows an authorized user, whose TSO ID has access to the DB2_DELETE Resource Access Definitions Facility (RADF) profile, to delete information about a particular job. See Chapter 9 of the <i>Smart Jobstream Series Installation and Customization Guide (publication SJS-001)</i> for a detailed description of RADF. Currently active jobs cannot be deleted. Jobs whose run status is ABEND should not be deleted since these jobs require the displayed information in order to restart.
DETailS	Displays jobstep details of a currently active Smart/RESTART job on a scrollable panel. These details include jobstep information, application activity counts, checkpoint pacing criteria and checkpoint information ¹ .

DRD	Delete Smart/RESTART jobs from the queue along with their corresponding checkpoint datasets.
EXIt	The EXIt command terminates Smart/MONITOR and returns to ISPF.
FILEs	The FILE command displays I/O activity for any repositionable files managed by Smart/RESTART ² .
FORCE	The FORCE command directs Smart/RESTART to immediately abend the application. Smart/RESTART issues the user abend regardless of whether processing is currently at a point of logical consistency. Any processing performed during the current unit-of-work is discarded ^{1,3} .
GOL	The GOL command displays the <u>G</u> lobal <u>O</u> pen <u>L</u> ist which shows all files in an <i>open</i> state as of the <i>last checkpoint</i> . This includes both repositionable files as well as other files opened by the current MVS task ² .
JOBstep	Smart/MONITOR displays jobstep information on the panel titled 'Jobstep Detail Information' ² .
PREDictor	The PREDICTOR command activates Smart/PREDICTOR for the selected job to <i>graphically portray the application's progress</i> . Chapter 35 describes Smart/PREDICTOR in detail ¹ .
PRM	The PRM command displays the current Smart/RESTART run-time profile ¹ .
PROfile	The PROFILE command lets you dynamically change the Smart/RESTART profile of the selected job. The Smart/RESTART Reference describes the various profile parameters in detail ¹ .
QUIESCE	The QUIESCE command lets you gracefully halt an application at a unit-of-work boundary. Smart/RESTART abends the application once the next physical checkpoint is complete. This enables the jobstep to resume execution at a more propitious time -- with no wasted processing ^{1,3} .
RECOvery	The RECOVERY command invokes the Abend Recovery dialog with which you can analyze a batch job failure and select a recovery strategy. Abend Recovery Facilities are described in Part V of this manual ¹ .

RESume	The RESUME command directs Smart/RESTART to resume execution of a suspended application ¹ .
RFS	The RFS command displays I/O activity for any currently active repositionable file managed by Smart/RESTART ¹ .
SQL	The SQL command displays an application's SQL activity. This includes SQL request counts by statement instance, statement type and issuing package or DBRM. In addition, SQL request rates are calculated per commit and per interval of time ¹ .
STAtistics	The STAtistics command summarizes an application's call to Smart/RESTART arranged by service type. In addition, Smart/RESTART service rates are calculated on a unit-of-work basis as well as per interval of time ¹ .
STOrage	The STORAGE command allows you to display and edit the storage areas that Smart/RESTART saves at checkpoint time ² .
SUSpend	The SUSpend command temporarily halts the execution of a Smart/RESTART batch job for a specified period of time. Smart/RESTART resumes application execution once this time interval expires. You can navigate through various Smart/MONITOR panels while a job is SUSPended. The RESume row command immediately resumes job execution ¹ .

(¹) the corresponding command can be issued against an active Smart/RESTART job only.

(²) the corresponding command retrieves data from a checkpoint medium and presents information as of the last checkpoint. These commands are only valid when BULK_CKPT is set to DB2, BDAM or VSAM. However, the VSAM checkpoint files cannot be shared between active Smart/RESTART jobs and Smart/MONITOR ISPF sessions, and as the result, these command cannot be used to retrieve the checkpoint information of an actively running Smart/RESTART job whose checkpoint medium is VSAM. Smart/MONITOR will substitute these commands with the corresponding commands denoted by (¹).

(³) the corresponding **command** is routed to the target Smart/RESTART job via the RAI Server Address space to which this Smart/RESTART job is connected, using the standard z/OS MODIFY interface:

MODIFY <rai_server_address_space_name>,SRS <jobname>,<command>

If the target Smart/RESTART job is running on another z/OS system, Smart/MONITOR routes the command to the RAI Server instance running on the

same z/OS system as the target Smart/RESTART job. Smart/MONITOR uses the standard z/OS ROUTE interface as follows:

```
ROUTE <z/OS_system_name>,MODIFY  
<rai_server_address_space_name>,SRS <jobname>,<command>
```

The RAI Server address space serves as a communications *bridge* between Smart/MONITOR and the Smart/RESTART job. As such, there is no need to establish cross memory communications between Smart/MONITOR and the target Smart/RESTART job in order to issue the **command**. The Smart/RESTART XMEM parameter governs whether or not cross memory communications should be established between Smart/RESTART and Smart/MONITOR. See Chapter 14 of the *Smart/RESTART Reference Manual* (publication SJS-005) for a detailed description of the XMEM parameter.

The **command** can be issued by authorized users only. The Smart/MONITOR user must have access to the RADF (Resource Access Definitions Facility) profile associated with the corresponding **command**. See Chapter 9 of the *Smart Jobstream Series Installation and Customization Guide* (publication SJS-001) for a detailed description of the Smart/MONITOR Resource Access Definitions Facility.

Smart/MONITOR Setup

This chapter describes how to restrict the list of jobs appearing in the scrollable job summary on the basis of jobname, start time, current status and a variety of other criteria. This chapter also describes how to customize Smart/MONITOR operation to suit your preferences.

31.1 The CONFIG Command

Enter the CONFIG command to display the Smart/MONITOR Profile panel illustrated in Figure 31.1.

```
Smart/MONITOR Profile
Command ==>
SRM051I Specify SRM profile parms and press ENTER or press END/RETURN to exit

Initial Display View
  Date Format . . . . . I      (I-ISO, U-USA, E-EUR or J-JIS)
  Time Format . . . . . I      (I-ISO, U-USA, E-EUR or J-JIS)
  Color control . . . . . N    (Select Y to change color setup)

Miscellaneous Section
  DB2 environment . . . . . N  (Y/N - modify DB2 env defaults)
  Multiple DB2 SSID's . . . . N (Y/N - define multiple DB2 subsystems)
  AUTO MODE . . . . . N       (Y/N - modify AUTO MODE setup)
  Message control . . . . . N  (Y/N - modify debug control)
```

Figure 31.1

The Profile panel includes two groups of parameters as follows:

- The set of parameters labeled 'Initial Display View' let you specify which jobs Smart/MONITOR should display on the scrollable job summary. (Described in Section 31.1.1.)
- Section 31.1.3 describes several miscellaneous parameters that govern Smart/MONITOR operation.

31.1.1 Initial Display View

You specify which jobs Smart/MONITOR should display on the scrollable job summary through the following parameters. Each parameter value you specify operates in concert with the other job selection criteria.

Date Format	Specifies the format in which Smart/MONITOR should display dates.
Time Format	Specifies the format in which Smart/MONITOR should display time values.
Colors	You can alter the colors of fields displayed by Smart/MONITOR by entering 'Y' in the Colors selection field.

31.1.2 Miscellaneous Section

DB2 Environment	Specify ‘Y’ to view and change the Smart/MONITOR settings that pertain to the DB2 environment. (Discussed in Section 31.2.3)
Multiple DB2 SSIDs	Specify ‘Y’ to configure Smart/MONITOR support for multiple, concurrently accessed DB2 subsystems. (Discussed in Section 31.2.2)
AUTOMODE	Enter ‘Y’ to display a window on which the number of update <i>intervals</i> and the <i>time</i> duration of each interval can be specified. (Discussed in Section 31.2.3). Also see the description of the AUTomode primary command in Chapter 33.
Message control	Enter ‘Y’ to display the Smart/MONITOR debug control panel. (Discussed in Section 31.2.3)

31.1.3 Specify DB2 - Related Parameters

The DB2 Environment command displays the popup panel illustrated in Figure 31.2.

```
----- DB2 Related Parameters -----
Command ==>

Smart/RESTART Checkpoint Synchronization Table
  Owner ID . . . . . DCAvrM
  DB2 Table Name . . DCAvrM.CKPT_SYNCH

Row Retention Period
  Expiration . . . . . 0    (Days)
  Action upon expiration A    (Archive/Delete)
  Confirm Action . . . . . Y    (Y/N - Archive or Delete)

DB2 environment parameters
  DB2 Subsystem . . . D9G    (with which to connect    )
  DB2 plan name . . . DCAvrMC (for Smart/MONITOR session)
  RLX SQL load module RLXSQL  (associated with appl plan)
  COMMIT request . . . Y      (Y/N - after every display)
```

Figure 31.2 **DB2 - Related Parameters**

31.1.4 Identify Checkpoint Synchronization Structure

The parameters described in this section let you identify the checkpoint structure in which synchronizing information about restartable jobs is stored.

Owner ID	Specifies the authorization ID of the owner of the Smart/RESTART checkpoint synchronization structure.
DB2 Table Name	Specifies the name of the Smart/RESTART checkpoint structure.
Row retention period	Expiration establishes aging criteria (in days) by which a job is deemed obsolete. When an authorized user^{1,2} starts a session, Smart/MONITOR displays those jobs old enough to satisfy the row retention period <i>expiration</i> criteria. The authorized user^{1,2} can delete or archive these obsolete rows or proceed directly to the Job Summary panel. Action upon expiration defines Smart/MONITOR action upon each obsolete job: • D - an authorized Smart/MONITOR user¹ can delete these obsolete rows. This is the default. • A - an authorized Smart/MONITOR user² can archive these obsolete rows
Confirm Delete Request	Confirm Action governs whether Smart/MONITOR should display a confirmation panel before archiving or deleting the information Smart/RESTART maintains about a job. This confirmation option applies both to automatic action (driven by the row retention period) as well as to the DELETE and ARCHIVE primary commands described in Chapter 33.

¹ A user whose TSO ID has access to the DB2_DELETE Resource Access Definitions Facility (RADF) profile.

² A user whose TSO ID has access to the DB2_ARCHIVE Resource Access Definitions Facility (RADF) profile.

NOTE: See Chapter 9 of the *Smart Jobstream Series Installation and Customization Guide* (publication SJS-001) for a detailed description of RADF.

31.1.5 Smart/MONITOR DB2 environment setup parameters

DB2 Subsystem	Specifies the DB2 subsystem ID or data sharing group name to which Smart/MONITOR should connect.
DB2 plan name	Specifies the DB2 plan name used by Smart/MONITOR.
RLX SQL load module	Specifies the name of the RLX SQL load module which should execute in conjunction with Smart/MONITOR's DB2 application plan.
Commit request	Governs whether Smart/MONITOR should issue a COMMIT after each SQL request.

31.2 CONFIG Command to configure Smart/MONITOR

The CONFIG primary command can be entered on any Smart/MONITOR panel. The syntax of the CONFIG command is as follows:

CONFIG {option}

where {option} specifies one of the following values:

*OWNERID=*DB2_table_owner_ID
*JOBNAME=*job_name
*STATUS=*job_status
COLORs
AUTomode
DB2Environment
MESsage

Specify only one of the options enumerated above on any single invocation of the CONFIG command. The OWNERID, JOBNAME and STATUS subcommands all require a value while the COLORs, AUTomode, DB2Environment and MESsage subcommands all cause an option window to be displayed.

31.2.1 AUTOMODE Setup

The AUTOMODE command specifies the *default* frequency and duration with which Smart/MONITOR automatically updates its displays. See the description of the AUTomode primary command in Chapter 35 for additional information.

```
----- AUTOMODE Setup -----

Specify AUTO MODE Attributes
Interval ==> 5      (Updates number in AUTO mode)
Time      ==> 10    (Time per information update, seconds)

Line command syntax
AUTomode {{<interval>},{<time>}}
```

Figure 31.3

31.2.2 Support for Monitoring Multiple DB2 Subsystems

Smart/MONITOR allows you to monitor restartable applications that connect to different DB2 subsystems and/or data sharing groups. However, the display of restartable jobs during any one Smart/MONITOR session are restricted to a single DB2 subsystem or data sharing group.

```
EDIT          SSID=dsn,DSN=db2hlq.SDSNLOAD,PGM=r1xSQLDriver   Columns 00001 00072
Command ==>                                         Scroll ==> PAGE
***** ***** Top of Data *****
000001 SSID=DSN1,DSN=DSN710.SDSNLOAD,PGM=NULL
000002 SSID=DSN2,DSN=DSN710.SDSNLOAD,PGM=NULL
000003 SSID=DSN3,DSN=DSN710.SDSNLOAD,PGM=NULL
***** ***** Bottom of Data *****
```

Figure 31.4

Each line defines setup parameters for a particular DB2 subsystem ID or data sharing group name. The keyword parameters and their values should be specified as follows:

SSID	specifies the DB2 subsystem ID or the name of the DB2 data sharing group to which Smart/MONITOR should connect.
DSN	specifies the name of the DB2 system library containing the executable code associated with the DB2 subsystem name or data sharing group.

PGM	Specifies the name of the RLX SQL load module which should execute in conjunction with Smart/MONITOR's DB2 application plan. The name should always be NULL.
-----	--

31.2.3 Debugging and Message Control

The MESSAGE command displays the Debugging and Message Control panel illustrated in Figure 31.5. Do not change these settings except as directed by RAI technical support.

```

---- Debugging and message control panel ---

Smart/MONITOR  ==> N (Y/N Trace)
  Debug level  ==> 0 (A/C/E/F/I/L/O/R)
Smart/RESTART  ==> N (Y/N Interface Msgs)
Operating System ==> N (Y/N Messages)

```

Figure 31.5

31.3 Smart/MONITOR Security

The following RACF resource profiles are implemented to provide more granular access to the resources controlled by Smart/MONITOR:

RACF profile name	Checkpoint synch DB2 table rows		Access application resources via SRS		Update the SRS run time profile	Read CKPT file	Access SRS control blocks (internal)
	Rea d	Delete	Read	Write			
RAI.SRM.ALTER	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RAI.SRM.CONTROL	Yes	Yes	Yes	No	Yes	Yes	No
RAI.SRM.UPDATE	Yes	No	Yes	Yes	Yes	Yes	No
RAI.SRM.READ	Yes	No	Yes	No	No	Yes	No
RAI.SRM.OPER	Yes	No	No	No	No	No	No

Chapter 4 of the *RAI Server Installation and Operations Guide* (publication RSV-001) describes how to control access to authorized Smart/MONITOR functions through the RAI Authorization Facility.

Smart/MONITOR Displays

Smart/MONITOR presents a hierarchy of panels that display job information in progressively greater detail. Successive levels can be accessed through primary commands, row selection commands and pop-up menus. The sections which follow discuss what can be observed, what can be altered and how to interpret the data.

32.1 Job Summary Display

Navigation begins with the scrollable Job Summary panel illustrated in Figure 32.1. The Job Summary displays your restartable applications in sequence by jobname, start-date and start-time. Chapters 30 and 31 describe how to restrict the list of jobs displayed on the Job Summary.

To request a specific row command, position the cursor to the selection field to the left of any jobname. For example, to request job detail information for the job named EXAMPLE, key in 'JOB' in the Cmd field to the left of the EXAMPLE row, as illustrated in Figure 32.1.

Smart/MONITOR vv.rr.mm		Smart/RESTART Job Queues					Row 1 to 17 of 19	
Command ==>							Scroll ==> PAGE	
		09/03/2010 11:43 RAI021 SRM >Panel 1 of 4>						
Checkpoint Synchronization DB2 Table Qualifier: DCA1015							DB2 SSID: DSNA	
							SORT St-Date/D, St-Time/D	
	ALL	ALL	ALL	ALL	ALL	ALL	ALL	ALL
Cmd	Jobname	StepName	LPAR	JesJobID	JesJStat	SyncStat	ASID	St-Date
								St-Time
	SJ15J8CM	SRMIVP	Z7L1	JOB00662	ACTIVE	RUNNING	002C	2010-09-03 11.43.00
	SJ158CPT	SRS8CPT	Z7L2	JOB00502	INACTIVE	S0C7	N/A	2010-08-30 14.34.00
	SJ15RZCB	STEP1SRS	Z7L1	JOB00382	INACTIVE	COMPLETE	N/A	2010-08-24 15.23.00
	SJ15J8CV	SRS8CV	Z7L1	JOB00324	INACTIVE	COMPLETE	N/A	2010-08-24 14.57.00
	SJ15JJJ3	SRSSAMP3	Z7L1	JOB00317	INACTIVE	COMPLETE	N/A	2010-08-24 14.56.00
	SJ15JJU2	SRS8CICB	Z7L1	JOB00307	INACTIVE	COMPLETE	N/A	2010-08-24 14.50.00
	SJ15JJU1	SRS8CIPT	Z7L1	JOB00302	INACTIVE	COMPLETE	N/A	2010-08-24 14.49.00
	SJ158CMJ	SRMIVP	Z7L2	JOB00249	INACTIVE	COMPLETE	N/A	2010-08-24 13.32.00

Figure 32.1

32.2 Jobstep Detail

The panel in Figure 32.2 presents detail on the job named EXAMPLE. Press the END PF key to return to the Job Summary display.

```
ckpt                      Smart/RESTART JobStep Detail
Command ==>

Restart run of job EXAMPLE number 1885
  Start Date   . . . . 04/15/2003   Smart/RESTART JOBID . EXAMPLE
  Start Time   . . . . 14.23.29     Program name   . . . . SRMSMPA
  Step/Procstep name SRMSAMPR       MVS system ID   . . . . RAI1
  Step Number   . . . . 0001        Resource Manager . . DB2
  Step Status   . . . . Abended      DB2 Plan Name   . . . . SRMSAMPD

Activity Counts
  Checkpoint ID . . . 26             (Absolute cross RESTART checkpoint ID)
  RSAM Files    . . . 3             (Transparent Files)
  API Files     . . . 0             (Manual API Files)
  Total Files   . . . 3             (Files managed by Smart/RESTART)
  Active modules . . 1             (Modules active at checkpoint time)
  Saved modules . . 1             (Modules resident at CKPT time)

Checkpoint Information
  Checkpoint DSN . . DCA.CHECKPT.DATASET
  Volume Serial . . GT0002
  Journal DSN . . .
  Program name   . . SRMSMPB         (Requesting checkpoint)
  Restart number . . 0               (Restart sequence number)
  Checkpoint id . . 26              (Absolute cross RESTART checkpoint ID)
  Depth . . . . . 2                 (Discrete checkpoints maintained)
  Size . . . . . 42                 (Number of blocks per checkpoint)
  Size Used . . . . 11              (Number of blocks used)
  Current RRN . . . . 1             (of the current checkpoint set)
  RRN Used . . . . 43               (of the highest checkpoint set)
  RRN Limit . . . . 84              (Last possible block)
  Actual Written . . 11             (Non-Bypassed CKPT writes)
```

Figure 32.2

The checkpoint information about the current job includes the following:

Program name	the name of the Smart/RESTART managed module which requested the checkpoint.
Restart number	On an initial run, the restart number is 0. A restart run contains the sequence number of the current restart run.
Resource Manager	The Resource Manager value identifies the resource manager accessed by the restartable application.
Depth	Indicates how many discrete checkpoints Smart/RESTART should maintain.

Size	Specifies the number of blocks allocated for each checkpoint.
RRN	Identifies the relative record number at which the set of records associated with the current checkpoint begins.

The JOB command directs Smart/MONITOR to retrieve the jobstep detail information from the checkpoint medium and reflects the status of the restartable batch job as of the last checkpoint. To obtain the most current jobstep detail for the target active restartable batch job, enter the DET command and press ENTER.

You can enter the ? row command to display the scrollable Smart/MONITOR menu, illustrated in Figure 32.3.

Smart/MONITOR Commands	
Command ==>	
Smart/RESTART job operations control	Access checkpoint medium
CAN . Phys CKPT next commit and abend	FIL . Repositionable File I/O Summary
FOR . Force immediate application abend	GOL . List application files
QUI . Quiesce job at next paced CKPT	JOB . Display Job/Jobstep Detail
RES . Resume Smart/RESTART job	STO . Application program Storage
SUS . Suspend Smart/RESTART job	
Active Smart/RESTART job only	DB2 Synchronization row control
AFS . Application File Summary	DEL . Delete Job entry from the table
APS . Application Program Storage	DRD . Delete Job entry and CKPT DA
DET . Display Job/Jobstep Detail	
PRE . Smart/PREDICTOR	
PRM . Display run-time profile	
PRO . Modify Smart/RESTART profile	
REC . Smart/RESTART Recovery Facility	
RFS . Repositionable File I/O summary	
STA . Smart/RESTART JOB statistics	HEL . Smart/MONITOR Online Reference
SQL . SQL Access Summary	EXI . Exit

Figure 32.3

32.3 Repositionable File I/O

The FIL and RFS row commands produce the displays illustrated in the figures below. The RFS command retrieves data directly from the currently active restartable batch job, whereas the FIL command obtains information from the checkpoint medium.

Figure 32.4 illustrates the Repositionable File I/O Summary on which I/O counts are listed by DDNAME and file type. (**Note:** Statistics for non-repositionable files do not appear in the File I/O Summary.)

Figures 32.5 and 32.6 illustrate how you can scroll the File I/O Summary up, down, left and right via the scroll PF keys. Scroll the File I/O Summary to the right to display a description of each file in terms of its record format, record length, BLKSIZE and current relative block position. Smart/MONITOR also displays the name of the module which opened the file.

```
Smart/RESTART - Repositionable File I/O Summary ROW 1 TO 3 OF 3
Command ==> Scroll ==> PAGE
SRM061I B - Browse, E - Edit most recent record.
Restart run of job EXAMPLE number 1885
  Start Date . . . . 04/15/2003    Smart/RESTART JOBID . EXAMPLE
  Start Time . . . . 14.23.29     Program name . . . . SRMSMPA
  Step/Procstep name SRMSAMPR     MVS system ID . . . . RAI1
  Step Number . . . . 0001        Resource Manager . . DB2
  Step Status . . . . Abended     DB2 Plan Name . . . . SRMSAMPD

S File   File   Records   Records   Total   Percent   Blocks   Total Blocks
  Name   Type   Processed Remaining Records Processed Processed

. INFILE INPUT      1295      6205      7500      17.26      64      375
. OUTFILE OUTPUT     1295      -          -          -          -          -
. OUTFILE2 OUTPUT    1295      -          -          -          -          -
```

Figure 32.4

```
Smart/RESTART - Repositionable File I/O Summary ROW 1 TO 3 OF 3
Command ==> Scroll ==> PAGE
SRM061I B - Browse, E - Edit most recent record.
Restart run of job EXAMPLE number 1885
  Start Date . . . . 04/15/2003    Smart/RESTART JOBID . EXAMPLE
  Start Time . . . . 14.23.29     Program name . . . . SRMSMPA
  Step/Procstep name SRMSAMPR     MVS system ID . . . . RAI1
  Step Number . . . . 0001        Resource Manager . . DB2
  Step Status . . . . Abended     DB2 Plan Name . . . . SRMSAMPD

S File   File   Block   Record   Record   Open by   DSN
  Name   Type   Size   Length   Format   Program

. INFILE INPUT    1600      80 FB    SRMSMPA RAIXN.DCAvrn.SRMINFIL
. OUTFILE OUTPUT  1600      80 FB    SRMSMPA RAIXN.DCAvrn.SRMOUTF1
. OUTFILE2 OUTPUT 1600      80 FB    SRMSMPA RAIXN.DCAvrn.SRMOUTF2
```

Figure 32.5

Scroll right once more to bring additional file information (such as volume serial number and device type) into view.

Enter 'B' or 'E' respectively, to the right of the File Name in order to Browse or Edit the file's current buffer. Chapter 34 describes the Smart/MONITOR Browse and Edit commands in detail.

```

Smart/RESTART - Repositionable File I/O Summary  ROW 1 TO 3 OF 3
Command ==>                                Scroll ==> PAGE
SRM061I B - Browse, E - Edit most recent record.
Restart run of job EXAMPLE number 1885
  Start Date . . . . 04/15/2003    Smart/RESTART JOBID . EXAMPLE
  Start Time . . . . 14.23.29      Program name . . . . SRMSMPA
  Step/Procstep name SRMSAMPR      MVS system ID . . . . RAI1
  Step Number . . . . 0001         Resource Manager . . DB2
  Step Status . . . . Abended      DB2 Plan Name . . . . SRMSAMPD

S File      File  DSN                                Volume Device
Name        Type                                Serial Type

. INFILE    INPUT RAIXN.DCAvrn.SRMINFIL             GT0005 DASD
. OUTFILE   OUTPUT RAIXN.DCAvrn.SRMOUTF1            GT0002 DASD
. OUTFILE2  OUTPUT RAIXN.DCAvrn.SRMOUTF2            GT0002 DASD

```

Figure 32.6

32.4 SQL Summary

The SQL Row command produces the display illustrated in Figure 32.7. The SQL monitoring facility must be active within the restartable batch job before meaningful statistics can be displayed. If SQL monitoring is not active, then Smart/MONITOR starts it dynamically. In this case, all counts reflect only the period of time since SQL monitoring becomes active.

Enter an 'S' to the left of a particular DBRM or package name to request detailed SQL monitoring data.

```

Smart/RESTART SQL Access Summary      ROW 1 TO 6 OF 6
Command ==>                          Scroll ==> PAGE

Job/Jobstep Information
Job Name      ==> EXAMPLE
Job Number    ==> 6556                (Assigned by Job Entry Subsystem  )
Step/Procstep ==> SRMIVP
Job status    ==> Suspended
Plan Name     ==> SRMSAMP            (DB2 Application Plan           )
DBRM Name     ==> SRSDBRM
Statement type ==> COMMIT            (Current SQL statement type     )
LUW number    ==> 400

S DBRM      Stmt Stmt      Total
Name        #   Type      Amount

. EXAMPLE2  182  INSERT.....    1
. EXAMPLE1  424  COMMIT.....  400
. EXAMPLE2  246  UPDATE.....   15
s EXAMPLE1  201  OPEN.....   399
. EXAMPLE2  149  FETCH.....  1995
. EXAMPLE1  170  CLOSE.....   399

```

Figure 32.7

32.5 SQL Statistics Detail

Figure 32.8 displays a count for each SQL statement type within the selected package or DBRM. For each statement type, the SQL Statistics Detail panel also shows the average number of SQL calls per commit as well as per second.

```

Smart/RESTART SQL Monitor

Command ==>

DBRM Name - EXAMPLE1  Current SQL statement type - CLOSE  Status - Running

Statement Type      Total      per      per
                   Amount      Checkpoint  Second

OPEN.....          829          39          2.4
FETCH.....         4145         199         12.3
CLOSE.....          829          39          2.4
PREPARE.....         0           0           0.0
EXECUTE.....         0           0           0.0
EXECUTE IMMEDIATE.... 0           0           0.0
DESCRIBE.....         0           0           0.0
EXPLAIN.....         0           0           0.0
SELECT.....         0           0           0.0
INSERT.....         0           0           0.0
DELETE.....         0           0           0.0
UPDATE.....         0           0           0.0
ROLLBACK.....        0           0           0.0
COMMIT.....         0           0           0.0
OTHER.....          0           0           0.0

```

Figure 32.8

32.6 Smart/RESTART Statistics

The STATistics Row command produces the display illustrated in Figure 30.9. The Smart/RESTART monitoring facility must be active within the restartable batch job before meaningful statistics can be displayed. If monitoring is not active, Smart/MONITOR starts Smart/RESTART monitoring dynamically. In this case, the activity counts displayed reflect the period of time since Smart/RESTART monitoring becomes active.

```
Smart/RESTART Performance Information

Command ==>

Job/Jobstep Information                                Total time in Commit function, sec
Job Name      ==> EXAMPLE                               Elapsed      ==> 54.29
Job Number    ==> 6556                                   TCB          ==> 8789.06
Step/Procstep ==> STEP1                                SRB          ==> 6.66
Job status    ==> Suspended
Program       ==> SRMSAMP
CSECT Name    ==> SRMSMPC      (Module in control      )
Function      ==> Checkpoint  (Last I/O requested function )

Function      Total calls      Calls per      Calls per
Name          Name              Checkpoint     Second

Read..... 1995              133           25.9
Write..... 3990             266           51.8
Open..... 3                 0             0.0
Close..... 0                 0             0.0
Checkpoint... 15             1             0.1
Initialize... 4789           319           62.1
Terminate... 4787           319           62.1
Allocate... 0                0             0.0
Abend..... 0                0             0.0
```

Figure 32.9

Smart/MONITOR Primary Commands

This chapter documents the Smart/MONITOR Primary Commands. These commands can be entered from the main panel only. Alternatively, you can enter a question mark (?) from the command line to display the scrollable help panel illustrated in Figure 33.1. This help panel identifies the primary commands supported by Smart/MONITOR and documents the syntax of each command.

The remaining sections of this chapter describe the Smart/MONITOR Primary Commands in alphabetical order.

Smart/MONITOR Primary Commands	
Command ==>	More: +
Display mode control	
ARCHive	Display archived Smart/RESTART job entries
AUTomode {interval}{,time}	Automated operations request
DFORMAT {ISO USA EUR JIS}	Modify Date format
TFORMAT {ISO USA EUR JIS}	Modify Time format
SORT columnName {Desc},...	Sort by various columns
Configuration Profile control	
CONFIG	Primary Configuration menu
CONFIG OWNERID=ownerID	Checkpoint synchronization DB2 table Owner ID
CONFIG JOBNAME=jobName	Select Smart/RESTART jobs by name
CONFIG COLORS	Color Attributes control
CONFIG AUTomode	AUTO MODE Setup
CONFIG DB2Environment	Update DB2 environment parameters
CONFIG MCAF	Update DB2 SSIDs and system libraries
CONFIG MESSAGE	Debugging and Message Control
Modify one or more actively running Smart/RESTART jobs with a single command. The operand jobName below is maskable via the standard SQL wild characters	
FORce JOBNAME=jobName	Cancel processing for a job immediately
RESume JOBNAME=jobName	Resume processing for a suspended job
SUSpend JOBNAME=jobName	Suspend processing of a job for one hour
QUIesce JOBNAME=jobName	Gracefully cancel a job at a UnitOfWork boundary

```

MODIFY jobName,command      Issue any valid Smart/RESTART run-time profile
F jobName,command          command via the standard MVS MODIFY interface

Debuging control
ABOUT                     Show Smart/MONITOR control variables
DEBUG ON|OFF              Turn debug messages ON/OFF
TRACE ON|OFF              Turn Smart/MONITOR REXX source trace ON/OFF
DUMp VAR varName{,HEX}    DUMP Smart/MONITOR REXX variables
DUMp BDAm ckptDataSet     DUMP Smart/RESTART checkpoint BDAM data set

Smart/RESTART checkpoint synchronization DB2 table delete row control
DELeTe columnName=columnValue

Note:   Scroll Job Summary ( PF7 - UP, PF8 - DOWN, PF10 - LEFT, PF11 - RIGHT),
        Specify Row, Line command, or press END/RETURN to exit

```

Figure 33.1

33.1 ARCHIVE Command

ARChive

The *ARChive* command directs Smart/MONITOR to display archived Smart/RESTART job entries as shown in Figure 33.2 below:

```

Smart/MONITOR vv.rr.mm      Smart/RESTART Job Queues (ARCHIVE) Row 1 to 4 of 4
Command ==>>>                                Scroll ==>>> PAGE
                                     11/12/2009 16:24 RAI021 SRM >Panel 1 of 5>
Checkpoint Synchronization DB2 Table Qualifier: DCA1010          DB2 SSID: D9G
                                     SORT St-Date/D, St-Time/D

```

	ALL	ALL	ALL	ALL	ALL	Archived	
Cmd	Jobname	StepName	LPAR	JesJobID	SyncStat	LPAR User ID	yyymmdd hh:mm:ss
	RAI78CMJ	SRMIVP	Z7L1	JOB00920	U2005	Z7L1 RAI021	090922 11:43:18
	RAI78CM2	SRMIVP	Z7L1	JOB00648	U2005	Z7L1 RAI021	090915 15:24:26
	RAI78CPT	SRS8CPT	Z7L1	JOB09725	S0C7	Z7L1 RAI021	090827 15:32:02
	RAI78CMJ	SRMIVP	Z7L1	JOB09649	U2005	Z7L1 RAI021	090827 11:22:05

Figure 33.2

Archived job entries are created by the *ARC* row command. The *ARC* row command must be issued for each Smart/RESTART job you wish to archive, as described in Section 30.4.

The archived job entries correspond to the Smart/RESTART DB2 synchronization rows whose *CKPT_SKEY_RRN* column is set to -2. You can issue the following SQL query to select the complete set of the archived Smart/RESTART jobs using SPUFI or a similar DB2 query tool: `SELECT * FROM DCA?dcavrm?.CKPT_SYNCH WHERE CKPT_SKEY_RRN = -2;`

NOTE: Once a Smart/RESTART synchronization row is archived, a restart of a corresponding jobstep is no longer possible. Jobsteps that require a restart should not have their synchronization DB2 rows archived.

Press PF3, or issue the END command to exit the archived Smart/RESTART Job Queues list, and to return back to the original display format.

33.2 AUTOMODE Command

`AUTOMODE(enter_limit,seconds)`

The **AUTomode** command establishes an automatic refresh period for Smart/MONITOR displays. The AUTomode command accepts two parameters. The first parameter denotes the number of ENTERs Smart/MONITOR should simulate. The second parameter specifies the duration (in seconds) between each simulated ENTER.

Smart/MONITOR simulates ENTER after each interval elapses. Smart/MONITOR continues to refresh the display until the number of simulated ENTERs specified by *enter_limit* is reached. Chapter 31 describes how to set **default** values for AUTOMODE.

NOTE: The AUTOMODE command is particularly useful in conjunction with the graphical display produced by Smart/PREDICTOR.

33.3 CONFIG Command

See the description of the CONFIG command in Chapter 31.

33.4 DEBUG Command

DEBUg(ON|OFF)

The DEBUG command displays a trace message each time a new Smart/MONITOR component is entered. This command is used only for debugging. Its operands are ON and OFF.

33.5 DFORMAT Command

DFormat(ISO|USA|EUR|JIS)

The DFORMAT command defines the date format Smart/MONITOR should use to display and interpret date values. The supported formats are ISO, USA, EUR and JIS.

33.6 DUMP Command

DUMp

The DUMP command is used for debugging. The DUMP command has three formats, distinguished by three *mutually exclusive* parameters:

VARIables *mask*

The **VARIABLE** parameter displays the names and values of all REXX variables referenced by the current Smart/MONITOR session. You can specify a mask to subset the list of variables displayed to those whose names match the mask value.

BDAM *dsname*

The second command format, specified via the **BDAM** parameter, displays the names and values of all variables maintained in the BDAM checkpoint file as of the **last** checkpoint. The **dsname** parameter specifies the name of the checkpoint file to be examined. Specify the checkpoint dataset name within quotes. The BDAM parameter is only valid when the BULK_CKPT profile option is set to BDAM. The DUMP BDAM command is not supported when BULK_CKPT specifies either DB2 or TAPE.

33.7 FULL / HALF Commands

FULL/HALF

The FULL and HALF commands control the display of application storage and file buffers. Issue the HALF command to *split* the screen in order to display application storage and file buffers simultaneously. Alternatively, issue the FULL command to restrict the display to a full screen of either file buffer contents or application storage.

33.8 SORT Command

SORT *column_name order*

The SORT command defines the order in which Smart/MONITOR displays the list of restartable jobs on the scrollable Job Summary panel. The *column_name* parameter identifies which column (such as JOBNAME or ST-DATE) should be used as the sort key. You must spell *column_name* exactly as it appears in the heading of the Job Summary panel. The *order* parameter specifies whether the list should be sorted in ascending or descending order.

33.9 TRACE Command

TRAcE ON|OFF

The TRACE command activates or deactivates a detailed trace that is useful in debugging Smart/MONITOR functions. The TRACE command should only be used at the direction of RAI technical support.

33.10 TFORMAT Command

TFOrmat(ISO|USA|EUR|JIS)

The TFORMAT command defines the time format Smart/MONITOR should use to display and interpret time values. The supported formats are ISO, USA, EUR and JIS.

33.11 Controlling Multiple Jobs with a Single Command

The following operands control one or more actively running Smart/RESTART jobs with a single command. They allow the <job_name> operand to be masked via the SQL wild characters _ and %:

FORce JOBNAME=<job_name>	Immediately cancel the processing of restartable jobs whose job name is like <job_name>
RESume JOBNAME=<job_name>	Resume processing of restartable jobs whose job name is like <job_name>
SUSpend JOBNAME=<job_name>	Suspend processing of restartable jobs whose job name is like <job_name>
QUIesce JOBNAME=<job_name>	Gracefully cancel at a Unit-of-Work boundary restartable jobs whose job name is like <job_name>
MODIFY <job_name>,<command> F <job_name>,<command>	Issue any valid Smart/RESTART run-time command against one or more restartable jobs using the standard MVS MODIFY interface

Smart/MONITOR Browse and Edit

Smart/MONITOR provides facilities with which to browse and/or edit the storage associated with a restartable application. Authorized users can view and change both working storage and the contents of file buffers through Smart/MONITOR's cross memory facilities. Once within either browse or edit, you can toggle between the two modes by issuing the EDIT and BROWSE primary commands. You can also toggle between character and hexadecimal representations of the data as described in the section entitled Browse and Edit commands.

34.1 View and Change the Contents of File Buffers

Figure 34.1 illustrates the Repositionable File I/O Summary. You can enter 'B' to the left of the file name in order to **browse** the associated dataset. Alternatively, enter 'E' to **edit** the file's current buffer.

```

                                Smart/RESTART - Repositionable File I/O Summary  ROW 1 TO 3 OF 3
Command ==>                                Scroll ==> PAGE
SRM061I B - Browse, E - Edit most recent record.
Restart run of job EXAMPLE number 1885
  Start Date . . . . 04/15/2003      Smart/RESTART JOBID . EXAMPLE
  Start Time . . . . 14.23.29       Program name . . . . SRMSMPA
  Step/Procstep name SRMSAMPR      MVS system ID . . . . RAI1
  Step Number . . . . 0001         Resource Manager . . DB2
  Step Status . . . . Abended       DB2 Plan Name . . . . SRMSAMPD

S File      File  DSN                                Volume Device
Name        Type                                     Serial Type

B INFILE    INPUT  RAI.DCA.SRMINFIL                  GT0005 DASD
. OUTFILE   INPUT  RAI.DCA.SRMOUTF1                  GT0005 DASD
. OUTFILE2  INPUT  RAI.DCA.SRMOUTF2                  GT0005 DASD

```

Figure 34.1

The Edit facility lets you type over the data within the buffer and then issue the SAVE command to externalize the results.

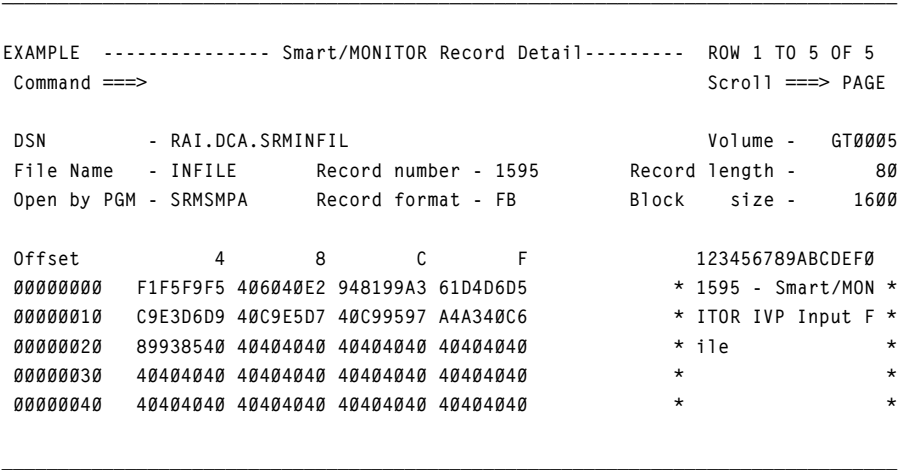


Figure 34.2

34.2 Application program Data Storage

Figure 34.3 illustrates how to request a display of application storage associated with the job named EXAMPLE. Simply enter the STOrage or APS row selection command, immediately to the *left* of the job of interest. The STOrage command directs Smart/MONITOR to retrieve data from the checkpoint medium when the APS command directs Smart/MONITOR to fetch information directly from the active batch job.

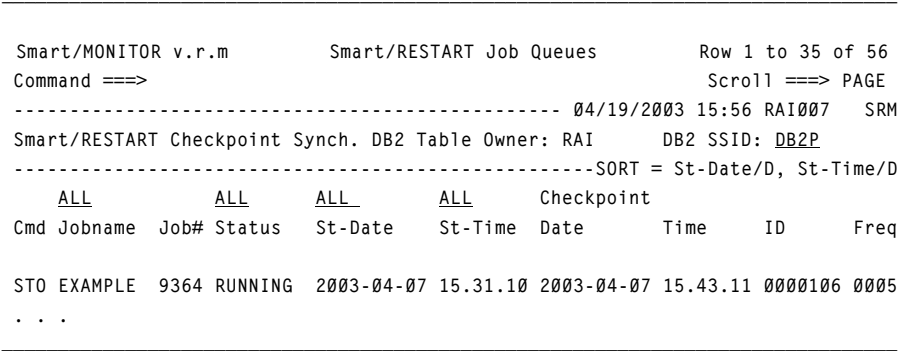


Figure 34.3

Smart/MONITOR displays the storage summary illustrated in Figure 34.4. This panel summarizes the storage defined to Smart/RESTART within each subprogram that is currently active within the run unit.

```

                                Application Program Storage Monitor          Row 1 to 3 of 3
Command ==>                                Scroll ==> PAGE
SRM043I S - Edit/Browse storage from memory, C - Browse storage from CKPT file
Restart run of job EXAMPLE number 1885
  Start Date . . . . 04/15/2003      Smart/RESTART JOBID . EXAMPLE
  Start Time . . . . 14.23.29        Program name . . . . SRMSMPA
  Step/Procstep name SRMSAMPR        MVS system ID . . . . RAI1
  Step Number . . . . 0001           Resource Manager . . DB2
  Step Status . . . . Abended         DB2 Plan Name . . . . SRMSAMPD

S Pair CSECT   Storage  Storage  Storage Size of   Files #   RBA of this
  Num  Name    Address  Length  Status  TGT Saved QSAM  VSAM  area in CKPT DA

S   1 SRMSMPA  09615028   156976 PRN           0    4    0           28
.   1 SRMSMPB  097831C0   153696 PRN           0    0    0          157048
.   2 SRMSMPB  097A8A40   156232 PRN           0    0    0          310744
.   1 SRMSMPC  097CF028   310216 PRN           0    0    0          467004

```

Figure 34.4

You can issue the 'S' and 'C' commands to drill down further into specific storage areas. The 'S' commands displays storage as it currently exists while the 'C' command displays the contents of a storage area as it existed at the last checkpoint.

The APS command permits you to edit and browse the complete program storage area (COBOL programs only) along with those storage areas defined to Smart/RESTART (i.e. the areas saved at checkpoint time and restored at restart time). The STO command allows you to edit and browse only those areas defined to Smart/RESTART.

NOTE: *You should suspend any job whose storage you edit and resume it once editing is complete.*

Figure 34.5 illustrates the Smart/MONITOR Storage Dump panel. Simply type over the target data to change its contents. You can then issue the SAVE command to actually update the storage.

```

----- Smart/RESTART - Storage Dump from Memory-----
Command ==> edit char_                               Scroll ==> PAGE

Job/Jobstep Information
  Job Name           ==> RAI7SRMR
  Job Number         ==> 1885           (Assigned by Job Entry Subsystem )
  Step / Procstep    ==> SRMSAMPR
  Program            ==> SRMSMPA       (Application Program name       )

Address  Offset          4          8          C          F          123456789ABSDEF0
***** Top of Data *****
09615028 00000000      6EE2D9E2 40C2C5C7 C9D54C00 00000000 * >SRS BEGIN< *
09615038 00000010      C9D5C9E3 6DE2E3E8 D3C54DC5 D5C45D40 * INIT_STYLE(END) *
09615048 00000020      40404040 40404040 40404040 40404040 * *
09615058 00000030      40404040 40404040 40404040 40404040 * *
09615068 00000040      40404040 40404040 40404040 40404040 * *
09615078 00000050      40404040 40404040 40404040 40404040 * *
09615088 00000060      D9C5E2E3 0000C4C3 D9C1F9F9 F940C9D5 * REST DCRA999 IN *
09615098 00000070      C9E3C9C1 D3C9E9C5 40400000 F0F0F0F0 * ITIALIZE 0000 *
096150A8 00000080      00000000 00000000 E2D9D4E2 D4D7C140 * SRMSMPA *
096150B8 00000090      F2000000 D7000000 00000000 00000000 * 2 P *
096150C8 000000A0      00000000 00000000 C1C2C5D5 C4404040 * ABEND *

```

Figure 34.5

34.3 Browse and Edit commands

Smart/MONITOR's Browse and Edit facilities support the following Primary commands:

A	<i>hexaddress</i>	Reset the storage display to start at the address specified by <i>hexaddress</i> .
F	<i>string</i>	Locate the value <i>string</i> . If found, reposition the display to begin with the line which contains <i>string</i> .
F	<i>string</i> FIRST	Locate the first occurrence of the value <i>string</i> and if found reposition the display to begin with the line which contains <i>string</i> . The search begins from offset 0 of the currently displayed storage area.
F	<i>string</i> LAST	Locate the last occurrence of the value <i>string</i> and if found reposition the display to begin with the line which contains <i>string</i> . The search begins from the end of the currently displayed storage area.
UP	<i>linecount</i>	Scroll the display backwards the number of lines specified by the <i>linecount</i> parameter.
DOWN	<i>linecount</i>	Scroll the display forwards the number of lines specified by the <i>linecount</i> parameter.

EDIT CHAR HEX	Governs which portion of the display is modifiable and which portion is protected. For example, in character mode, the character data is highlighted and modifiable while the hexadecimal data is protected. Simply type over the modifiable portion of the display in order to change data. You can toggle between Char and Hex modes.
BROWSE	Switch to Browse mode from Edit mode. Browse allows you to view but not alter data.
SAVE	Signals Smart/MONITOR to externalize changes to either application storage or the contents of file buffers.

Smart/PREDICTOR

35.1 Overview

Smart/PREDICTOR analyzes file I/O and CPU utilization rates to project an application's completion time and graphically displays its progress. These completion time estimates are based on the following factors:

Input Files: The rate at which an application processes its repositionable input files is one of the critical factors used to project a completion time. For projections to be accurate, these input files must be read until end-of-file.

Steady State Execution: Accurate projections require that jobs run at relatively steady rates.

Operating Environment: The OS/390 System Resource Manager (SRM) apportions service units to a job based on service objectives defined for that job. Smart/PREDICTOR assumes a job's service objectives do not change during execution.

35.2 How to Obtain Completion Time Predictions

Smart/PREDICTOR can project completion times only for those applications for which prediction is enabled. To activate Smart/PREDICTOR for a particular jobstep, specify PREDICT(ON) in the application's RAINPUT stream. Since Smart/PREDICTOR overhead is insignificant, RAI recommends that PREDICT be set ON at the start of all jobs you may wish to forecast.

35.3 Smart/PREDICTOR Displays

Figure 35.1 illustrates how to request a completion time estimate for the job named EXAMPLE by keying in the ‘PRE’ row command to the left the jobname.

S/MONITOR ----- Smart/RESTART Job Queues ----- Row 1 to 35 of 56									
Command ==>					Scroll ==> PAGE				
----- 02/09/07 15:56 RAI007 SRM									
Smart/RESTART Checkpoint Synch. DB2 Table Owner: RAI					DB2 SSID: DB2P				
-----SORT = St-Date/D, St-Time/D									
ALL	ALL	ALL	ALL	Checkpoint					
Cmd	Jobname	Job#	Status	St-Date	St-Time	Date	Time	ID	Freq
PRE	EXAMPLE	9364	RUNNING	2002-09-07	15.31.10	2002-09-07	15.43.11	0000106	0005
...	RAI8XXXX	7234	S222	2002-09-07	12.12.20	2002-09-07	13.03.11	0000006	0005
...	RAI8SRMR	6556	COMPLETE	2002-09-07	30.57.17	2002-09-07	12.04.08	0000026	0100
...	RAI7VSTR	6644	S0C7	2002-08-24	16.25.38	2002-08-24	16.25.41	0000003	0001
...	RAI021	4269	RUNNING	2002-08-24	12.39.22	2002-08-24	12.39.22	0000000	0001
...	RAI021	4269	S13E	2002-08-24	12.35.02	2002-08-24	12.35.29	0000003	0001

Figure 35.1

The Prediction panel in Figure 35.2 graphically displays the progress of the application and forecasts the jobstep completion time.

```
----- Smart/PREDICTOR - Run Time Estimator -----
Command ==>

Job Information                                Input I/O counts          Blocks  Records
Name      ==> RAI7SRMR                      Read since first Run      250     4990
Number    ==> 3593                          Read for this Run         33      665
Status     ==> Running                       In 1 Open Input Files     375

Timing statistics                            Input I/O rates
Since 1st Run ==> 00:03:26                  Seconds per read          00.824   00.040
So far      ==> 00:00:27                    Reads per second          1.2     24.4
Remaining   ==> 00:01:40
Estimated EOJ ==> -
Projected EOJ ==> 02:28 pm. Sep 11th, 2002
Projected Late ==> 00:00:00
Suspend Time ==> 00:00:00

0          25          50          75          100
|||||>.....|.....|.....|
|||||>.....|.....|.....|
0          25          50          75          100

21% completed for this run - 67% completed since initial run
```

Figure 35.2

The Prediction panel displays the following execution statistics:

Input I/O counts:

Read since first run (Blocks): The number of input blocks processed since the application was initially submitted. For a restarted job, this count may be higher than that reported in **Blocks read since start of run**.

Read since first run (Records): The number of input records read since the application was initially submitted. For a restarted job, this count may be higher than that reported in **Records read since start of run**.

Read since start of run (Blocks): The number of input blocks processed during this particular run.

Read since start of run (Records): The number of input records read during this particular run.

In *nnn* Open Input Files: The estimated number of input blocks detected in all open input RSAM files, where *nnn* contains the number of all open input RSAM files -- i.e. the number of repositionable input files that are currently open.

Input I/O rates:

Seconds per read (Blocks): The I/O service time per input block expressed in blocks read per millisecond.

Seconds per read (Records): The I/O service time per input record expressed in records read per millisecond.

Reads per second (Blocks): The number of input blocks read per second during the most recent analysis interval.

Reads per second (Records): The number of input blocks read per second during the most recent analysis interval.

Timing Statistics:

Since 1st Run: The estimated amount of time it took the application to read all the blocks as indicated in the field "Blocks read since start of run." The value of this field is derived using the following formula:

$$\text{Since_1st_Run} = \text{Blocks_read_since_initial_run} * \text{time_to_read_one_block}$$

where

$$\text{time_to_read_one_block} = \text{time_so_far} / \text{Blocks_read_since_start_of_run}$$

So far:	The total run-time since Smart/PREDICTOR was activated by the Smart/PREDICTOR ISPF dialog, or explicitly by the Smart/RESTART command PREDICT(ON). See Chapter 13 of the Smart/RESTART Reference Manual for detailed information on Smart/RESTART command PREDICT.
Remaining:	The estimated amount of time the application will run before it completes successfully.
Target EOJ:	The estimated date and time the application is <i>expected</i> to complete successfully. The value of this field is calculated as the sum of the values specified in the Smart/RESTART command EST_RUN_TIME and the time identified by the field Since 1st Run . See Chapter 11 of the Smart/RESTART Reference Manual for detailed information on Smart/RESTART command EST_RUN_TIME.
Projected EOJ:	The estimated date and time the application will complete successfully. The value of this field is calculated as the sum of the current time and the time as estimated for the field Remaining .
Projected Late:	The estimated additional time required for the application to complete successfully. The value of this field is derived using the formula

$$\text{Projected_Late} = (\text{so_far} + \text{remaining}) - \text{est_run_time}$$

where `est_run_time` contains the value specified in Smart/RESTART command `EST_RUN_TIME`. Please note that the value for **Projected Late** is calculated only when `EST_RUN_TIME` contains a non-zero value.

Suspend Time:	The accumulated amount time that the application was non-dispatchable. See Chapter 13 of the Smart/RESTART Reference Manual for detailed information on Smart/RESTART command SUSPEND.
----------------------	--

NOTE: *Smart/PREDICTOR displays a message at the top of the Prediction panel to alert the observer to significant events such as a job being suspended.*

Smart/RESTART Support for Assembler Language Applications

This chapter describes and illustrates Smart/RESTART support for modules coded in Assembler Language. Restartable application development in Assembler Language is the same as compiled language development in the following respects:

- the same principles apply -- such as issuing checkpoints at unit-of-work boundaries
- Sequential file repositioning is supported transparently. Your modules can issue standard OPEN, GET, PUT and CLOSE macros provided the DCB specifies QSAM locate mode.
- Both main routines and subprograms can be coded in assembler language.

In contrast, there is no Smart/PRECOMPILER support for assembler language so all requests for Smart/RESTART services (except for repositionable file I/O) must be coded explicitly.

The sample assembler-language application supplied with Smart/RESTART consists of two modules -- a main routine named SRS8A3M (illustrated in Figure 40.1) and a subroutine named SRS8A3S (shown in Figure 40.2). These modules, along with the JCL to build and execute the sample application, are delivered in machine readable form in the following members of the DCASAMP library:

SRS8A3M	source of the Assembler Language main module
SRS8A3S	source of the Assembler Language subroutine
SRSBASM3	JCL to build the SRS8ASM3 sample application
SRSRASM3	JCL to run the SRS8ASM3 sample application
SRS8ASM2	source of the Smart/RESTART Assembler / IMS application
SRSBASM2	JCL to build the SRS8ASM2 sample application
SRSRASM2	JCL to run the SRS8ASM2 sample application

In addition, the DCASAMP dataset includes the following macros to help you develop restartable applications in assembler language:

SRS@API expands the Smart/RESTART service name literals
 SRS@FCS maps the File Control Specification (not used with transparent I/O)
 SRS@EDA maps the feedback area for the RAI_ENVCHECK service
 SRS@EWA maps the Smart/RESTART exit work area
 SRS@FCA maps the Smart/RESTART Function Communications Area

NOTE: The SRS8A3M and SRS8A3S source modules make use of the structured IF THEN ELSE ENDIF macros that reside within the DCAMACS dataset.

40.1 An Assembler Language Main Module

The Assembler Language main module SRS8A3M appears in Figure 40.1 below. It calls the Smart/RESTART **INITIALIZE** service to identify those work areas which should be saved at checkpoint time and restored in the event of restart. Module SRS8A3M calls the Smart/RESTART **TERMINATE** service to indicate it is about to exit.

Since the DCB defined in module SRS8A3M specifies QSAM locate mode the module can perform repositionable I/O by issuing standard OPEN, CLOSE, PUT and GET macros. Smart/RESTART's Repositionable Sequential Access Method (RSAM) transparently supports files defined and accessed in this manner.

```

      TITLE 'SRS8A3M - SMART/RESTART SAMPLE MAIN ASSEMBLER ROUTINE'
*-----
*
*   This sample Smart/RESTART assembler main routine reads
*   records and invokes a subprogram to write each record and
*   increment a record counter. The main routine explicitly
*   calls the checkpoint service for every 100 records
*   processed.
*
*   Shortly after entry, the Smart/RESTART INITIALIZE service
*   is explicitly called to create the Smart/RESTART
*   environment. Immediately before exiting, the TERMINATE
*   service is explicitly called to remove the Smart/RESTART
*   environment.
*
*   In contrast, repositionable I/O is performed transparently
*   using standard OPEN, GET, PUT and CLOSE macros.
*
*   Note: this routine uses static areas and is non-reusable.
*-----
      COPY #ASMEV          STRUCTURED MACROS / SYSPARM OPTIONS
      YREGS
SRS8A3M CSECT ,
SRS8A3M AMODE 24           MODULE'S ADDRESSING MODE
SRS8A3M RMODE 24          MODULE'S RESIDENCE MODE
      USING *,R15
      STM  R14,R12,12(R13)  STORE REGS IN MVS PROVIDED SAVEAREA
*
      ST   R13,SAVEAREA+4   STORE A(ATTACH SA) AS PREVIOUS SA
      LR   R12,R13          R12 --> PREV SA
      LA   R13,SAVEAREA     R13 --> SRS8A3M SAVEAREA (SA1)
      ST   R13,8(R12)       STORE CURR SA AS 'NEXT' IN PREV SA
*
      DROP R15

```



```

        LA      R12,0(R15)          R12 --> CSECT ADDRESS (NO HIGH BYTE)
        USING  SRS8A3M,R12          ESTABLISH ADDRESSABILITY
*
        BAL     R14,INITAPI          INITIALIZE SMART/RESTART ENVIRONMENT
        IF (LTR,R15,R15,NZ)          SUCCESSFUL ?
        B       EPILOG              NO, EXIT IMMEDIATELY
        ENDIF
*
        LA      R10,INFILE           R10 --> INFILE DCB
        OPEN   ((R10),(INPUT))       OPEN QSAM DCB - GET LOCATE MODE
        IF (LTR,R15,R15,NZ)          WAS SMART RESTART OPEN SUCCESSFUL ?
        WTO 'OPEN FOR INPUT FILE WAS UNSUCCESSFUL',ROUTCDE=(11)
        B       EPILOG              EXIT IMMEDIATELY
        ENDIF
*
        XR      R9,R9                CLEAR LREC COUNTER FOR COMMIT CALL
*
        READREC DS      0H
        LA      R9,1(R9)             ADVANCE COMMIT TRIGGER COUNTER
        GET     (R10)                GET READ BUFFER ADDRESS FROM SRS
*
*      INVOKE SUBROUTINE TO WRITE RECORD JUST READ (POINTED TO BY R1)
*
        L       R15,SRS8A3S          R15 --> SUBROUTINE EPA
        BALR    R14,R15              INVOKE SRS8A3S TO WRITE THE RECORD
        IF (LTR,R15,R15,NZ)          SUCCESSFUL ?
        LR      R2,R15               NO, HOLD RC ACROSS WTO
        WTO     'SRS8A3S FAILED'     ISSUE ERROR MESSAGE
        LR      R15,R2               RESTORE RC
        B       EPILOG              EXIT IMMEDIATELY
        ELSE
        ST      R0,OUTCOUNT         SAVE CURRENT COUNTER FOR DISPLAY
        ENDIF
*
*      CAUSE A DELIBERATE ABEND AFTER 268 RECORDS ON THE INITIAL RUN
*      NOTE: THE TEST_ABEND PROFILE COMMAND MAY BE USED INSTEAD
*
        IF (CH,R9,EQ,H268),AND,      268 RECS YET?                                X
            (TM,LCLFLAG,RESTART,NO) AND NOT A RESTART ?
        DC      XL2'00C1'            ABEND AND ROLLBACK FILE POS + STOR
        ENDIF
*
        IF (CH,R9,EQ,H100)            100 RECORDS PROCESSED (ONE LUW)?
        BAL     R14,CKPTAPI           YES, ISSUE CHECKPOINT API REQUEST
        IF (LTR,R15,R15,NZ)          SUCCESSFUL ?
        B       EPILOG              NO, EXIT IMMEDIATELY
        ENDIF
        XR      R9,R9                RESET READ COUNT FOR NEW LUW
        ENDIF
*
        B       READREC              READ THE NEXT RECORD
*
*
        EOD    DS      0H              END OF DATA EXIT ROUTINE
        WTO     'SRS8A3M  END OF FILE ENCOUNTERED'
*
        BAL     R14,TERMAPI           REMOVE THE SMART/RESTART ENVIRONMENT
*
        EPILOG DS      0H
        L       R13,4(R13)            R13 --> ATTACH SAVEAREA
        L       R14,12(R13)           RESTORE R14 FROM ATTACH SA
        LM      R0,R12,20(R13)        RESTORE ALL REGS FROM ATTACH SA
        BSM     R0,R14                RETURN TO INITIATOR
*

```

```

*-----
*
*   ISSUE A SMART/RESTART INITIALIZE API REQUEST TO DEFINE
*   THE WORK AREAS TO SAVE AT COMMIT TIME.
*
*   BUILD AN INITIALIZE PARAMETER LIST AND ISSUE THE API REQUEST.
*   ASSUME USE OF INIT-STYLE(LENGTH).  THIS INITIALIZE CALL DEFINES
*   THE STORAGE WITHIN THE SRS8A3M MODULE TO SAVE.
*
*   SEE THE INITIALIZE CALL IN MODULE SRS8A3S THAT DEFINES THE
*   WORK AREAS WITHIN THAT SUBPROGRAM THAT ARE SAVED AT COMMIT TIME
*-----
INITAPI  ST      R14,SAVER14
*
*       LA      R11,WFCA           R11 --> WROK FCA FOR API REQUESTS
*       USING   FCA,R11           STACK AND OVERLAY FCA MAP
*       XC      WFCA,WFCA         CLEAR WORK FCA
*
*       MVC     FCACSNAM,CSNAME    PLACE CSECT NAME IN FCA
*       MVI     FCARAMST,FCARAMSL  Pass vital areas by address & length
*
*       LA      R2,PARMLIST        R2 --> COMMOM API PARMLIST
*       XC      PARMLIST,PARMLIST  CLEAR THE PARAMETER AREA
*
*       LA      R1,RAI_INITIALIZE  R1 --> INITIALIZE FUNCTION NAME
*       ST      R1,0(R2)           STORE 'INITIALIZE' AS FIRST PARM
*       LA      R1,WFCA            R1 --> PTR TO FCA IN PARM LIST
*       ST      R1,4(R2)           STORE PTR TO FCA IN PARM LIST
*       LA      R1,AREA1           R1 --> PTR TO AREA1 IN PARM LIST
*       ST      R1,8(R2)           STORE PTR TO AREA1 START
*       LA      R1,AREA1LEN        R1 --> PTR TO AREA1 LENGTH
*       ST      R1,12(R2)          STORE PTR TO AREA1 LENGTH
*       LA      R1,AREA2           R1 --> PTR TO AREA1 IN PARM LIST
*       ST      R1,16(R2)          STORE PTR TO AREA2 START
*       LA      R1,AREA2LEN        R1 --> PTR TO AREA2 LENGTH
*       ST      R1,20(R2)          STORE PTR TO AREA2 LENGTH
*       OI      20(R2),X'80'       MARK LEN AS LAST PARM
*       LA      R1,0(R2)           R1 --> INITIALIZE API PARM LIST
*
*       L       R15,RAIAPI         R15 --> RAIAPI APPL API STUB
*       BALR    R14,R15            CALL RAIAPI
*
*
*       IF (OC,FCARETC,FCARETC,NZ) INITIALIZE REQUEST SUCCESSFUL ?
*       WTO     'INITIALIZE FAILED',ROUTCDE=(11)
*       LA      R15,8              NO, INDICATE ERROR
*       B       INITAPIX           EXIT IMMEDIATELY
*       ENDIF
*
*
*       IF (CLC,FCASTAT,EQ,$REST) IS THIS A RESTART ?
*       WTO     'RESTART DETECT',ROUTCDE=(11)
*       OI      LCLFLAG,RESTART    YES, INDICATE RESTART IN PROGRESS
*       ENDIF
*
*       MVC     AREA2,AREA2LIT     INITIALIZE SAVE AREA 2
*       XR      R15,R15            INDICATE SUCCESS
*
*       INITAPIX L      R14,SAVER14
*       BR      R14               RETURN TO MAINLINE
*-----
*
*   ISSUE TERMINATE API REQUEST
*
*   BUILD TERMINATE PARAMETER LIST AND ISSUE API REQUEST
*-----
TERMAPI  ST      R14,SAVER14
*
*       LA      R11,WFCA           R11 --> WORK FCA FOR API REQUESTS
*       USING   FCA,R11           STACK AND OVERLAY FCA MAP

```

```

*      XC      WFCA,WFCA          CLEAR FCA USED BY TERMINATE SERVICE
*
*      MVC      FCACSNAM,CSNAME    PLACE CSECT NAME IN FCA
*
*      LA      R2,PARMLIST          R2 --> COMMOM API PARMLIST
*      XC      PARMLIST,PARMLIST    CLEAR THE PARAMETER AREA
*
*      LA      R1,RAI_TERMINATE     R1 --> TERMINATE STRING
*      ST      R1,0(R2)             STORE 'TERMINATE' AS FIRST PARM
*      LA      R1,WFCA              R1 --> PTR TO FCA IN PARM LIST
*      ST      R1,4(R2)             STORE PTR TO FCA IN PARM LIST
*      OI      4(R2),X'80'          MARK FCA AS LAST PARM
*      LA      R1,0(R2)             R1 --> TERMINATE API PARM LIST
*
*      L        R15,RAIAPI          R15 --> RAIAPI APPL API STUB
*      BALR     R14,R15             CALL RAIAPI
*
*
*      IF (OC,FCARETC,FCARETC,NZ)   TERMINATE REQUEST SUCCESSFUL ?
*      WTO      'TERMINATE FAILED',ROUTCDE=(11)
*      LA      R15,8                NO, INDICATE ERROR
*      B        TERMAPIX            EXIT IMMEDIATELY
*      ENDIF
*
*      XR      R15,R15              INDICATE SUCCESS
*
*      TERMAPIX L      R14,SAVER14
*      BR      R14                  RETURN TO MAINLINE
*
* -----
*
*      ISSUE CHECKPOINT API REQUEST
*
*      BUILD THE CHECKPOINT PARAMETER LIST AND ISSUE API REQUEST
*
*      NOTE: SQL MODULES THAT USE Smart/RRSAF MAY REPLACE THE
*      CALL TO THE CHECKPOINT SERVICE WITH A STANDARD SQL COMMIT
*      STATEMENT.
*
* -----
*
*      CKPTAPI ST      R14,SAVER14
*
*      LA      R11,WFCA              R11 --> COMMON FCA FOR API REQUESTS
*
*      XC      WFCA,WFCA          CLEAR INITIALIZE FCA
*      MVC      FCACSNAM,CSNAME    PLACE CSECT NAME IN FCA
*
*      LA      R2,PARMLIST          R2 --> COMMOM API PARMLIST
*      XC      PARMLIST,PARMLIST    CLEAR THE PARAMETER AREA
*
*      LA      R1,RAI_CHECKPOINT     R1 --> CHECKPOINT STRING
*      ST      R1,0(R2)             STORE 'CHECKPOINT' AS FIRST PARM
*      LA      R1,WFCA              R1 --> PTR TO FCA IN PARM LIST
*      ST      R1,4(R2)             STORE PTR TO FCA IN PARM LIST
*      OI      4(R2),X'80'          MARK FCA AS LAST PARM
*      LA      R1,0(R2)             R1 --> CHECKPOINT API PARM LIST
*
*      L        R15,RAIAPI          R15 --> RAIAPI APPL API STUB
*      BALR     R14,R15             CALL RAIAPI
*
*
*      IF (OC,FCARETC,FCARETC,NZ)   CHECKPOINT REQUEST SUCCESSFUL ?
*      WTO      'CKPT API FAILED',ROUTCDE=(11)
*      LA      R15,8                NO, INDICATE ERROR
*      B        CKPTAPIX            EXIT IMMEDIATELY
*      ENDIF
*
*
*      XR      R15,R15              INDICATE SUCCESS
*
*      CKPTAPIX L      R14,SAVER14
*      BR      R14                  RETURN TO MAINLINE

```

```

*-----
*
*          DATA DECLARATIONS
*
*-----
*
*          DS      0F          ALIGN
INFILE   DCB      DDNAME=INFILE,MACRF=(GL),DSORG=PS,EODAD=EOD,LRECL=80
RAIAPI   DC        VL4(RAIAPI)
SRS8A3S  DC        VL4(SRS8A3S)
CSNAME   DC        CL8'SRS8A3M'          CSECT NAME
*
*          DS      0D
AREA1    DC        20CL10'SAVED AREA1 '      WORKING STORAGE AREA 1
AREA1SZ  EQU      *-AREA1
*          DS      0H
AREA1LEN DC        AL4(AREA1SZ)
*
H100     DC        H'+100'
H268     DC        H'+268'
*
$REST    DC        CL4'REST'
AREA2LEN DC        AL4(AREA2SZ)
*
AREA2LIT DC        20CL10'SAVED AREA2 '      WORKING STORAGE AREA 2
*
*          SRS@API          SMART RESTART SERVICES NAMES
*
*
*          DS      0F
WFCA     DS        (FCALEN)CL1          WORK FUNCTION COMMUNICATIONS AREA
SAVEAREA DS        18F
SAVER14  DS        F
*
PARMLIST DS        6F          $INITIALIZE PARMLIST AREA
*
LCLFLAG  DS        CL1
RESTART  EQU      B'10000000'
*
*
*          DS      0D
AREA2    DS        CL200          WORKING STORAGE AREA 2
CVDWORK  DS        D            CONVERT AREA
OUTCOUNT DS        F            OUTPUT COUNT FROM SUBROUTINE
RECNUM   DS        CL5          DISPLAY COUNT
AREA2SZ  EQU      *-AREA2
*
*          DSECT
*
*          SRS@FCA          SRS FUNCTION COMMUNICATIONS AREA
*
*          END    SRS8A3M

```

Figure 40.1 *Main module SRS8A3M*

40.2 An Assembler Language Sub-Module

The assembler language subroutine module named SRS8A3S appears in Figure 40.2. Like the main module, SRS8A3S calls the Smart/RESTART **INITIALIZE** service to identify those work areas which should be saved at checkpoint time and restored in the event of restart. Similarly, the subroutine calls the Smart/RESTART **TERMINATE** service to indicate it is about to exit.

```

-----
TITLE 'SRS8A3S - Smart/RESTART SAMPLE ASSEMBLER SUBROUTINE'
*-----
*
*      This sample Smart/RESTART assembler subroutine is invoked
*      by main module SRS8A3S to write a record and
*      increment a record counter.
*
*      Shortly after entry, the Smart/RESTART INITIALIZE service
*      is explicitly called to identify those subroutine work areas
*      that should be saved at checkpoint time and restored
*      at restart time.
*
*      Immediately before exiting, this subroutine calls the
*      TERMINATE service explicitly.
*
*      In contrast, repositionable I/O is performed transparently
*      using standard OPEN, GET, PUT and CLOSE macros.
*
*      This routine uses static areas and is non-reusable.
*-----
COPY  #ASMENV          STRUCTURED MACROS / SYSPARM OPTIONS
YREGS
SRS8A3S CSECT ,
SRS8A3S AMODE 24          MODULE'S ADDRESSING MODE
SRS8A3S RMODE 24          MODULE'S RESIDENCE MODE
      USING *,R15
      STM  R14,R12,12(R13)  STORE REGS IN SRS8A3S SAVEAREA
*
      ST   R13,SAVEAREA+4   STORE A(SRS8A3S SA) AS PREVIOUS SA
      LR   R12,R13          R12 --> PREV SA
      LA   R13,SAVEAREA     R13 --> SRS8A3S SAVEAREA (SA1)
      ST   R13,8(R12)       STORE CURR SA AS 'NEXT' IN PREV SA
*
      DROP R15
      LR   R12,R15          R12 --> PROGRAM BASE
      USING SRS8A3S,R12     ESTABLISH ADDRESSABILITY
*
      LR   R9,R1            R9 --> RECORD JUST READ BY MAIN PGM
*
      BAL  R14,INITAPI       ISSUE INITIALIZE CALL
      IF (LTR,R15,R15,NZ)    SUCCESSFUL ?
      B    EPILOG           NO, EXIT IMMEDIATELY
      ENDIF
*
      LA   R10,OUTFILE       R10 --> OUTFILE DCB
      IF (TM,LCLFLAG,OPEN,NO) WAS FILE OPENED ALREADY ?
      OPEN ((R10),(OUTPUT)) NO, OPEN QSAM DCB - PUT LOC
      IF (LTR,R15,R15,NZ)    SMART RESTART OPEN SUCCESSFUL ?
      WTO 'OPEN FOR OUTPUT FILE WAS UNSUCCESSFUL',ROUTCDE=(11)
      B    EPILOG           EXIT IMMEDIATELY
      ELSE
      OI   LCLFLAG,OPEN
      ENDIF
      ENDIF
*
      PUT  (R10)             WRITE OUTPUT RECORD FROM INPUT BUF

```

```

*
*      MVC  0(00,R1),0(R9)      COPY SRS8A3S INPUT BUF TO OUTPUT BUF
*      L    R1,RECCOUNT          R1 = CURRENT OUTPUT RECORD COUNT
*      LA   R1,1(R1)             INCREMENT RECORD COUNT
*      ST   R1,RECCOUNT          STORE IN AREA SAVED DURING COMMIT
*
*      BAL  R14,TERMAPI          ISSUE TERMINATE PRIOR TO EXIT
*
*      EPILOG DS  0H
*      L    R0,RECCOUNT          RETURN RECORD COUNT TO CALLER
*      L    R13,4(R13)           R13 --> SRS8A3S SAVEAREA
*      L    R14,12(R13)          RESTORE R14 FROM SRS8A3S SA
*      LM   R1,R12,24(R13)       RESTORE ALL REGS FROM SRS8A3S SA
*      BSM  R0,R14              RETURN TO SRS8A3S
*-----
*
*      ISSUE INITIALIZE API REQUEST
*
*      BUILD AN INITIALIZE PARAMETER LIST AND CALL RAIAPI
*      ASSUME USE OF INIT-STYLE(LENGTH)
*-----
INITAPI ST   R14,SAVER14
*
*      LA   R11,WFCA              R11 --> COMMON FCA FOR API REQUESTS
*      USING FCA,R11              STACK AND OVERLAY FCA MAP
*      XC   WFCA,WFCA              CLEAR INITIALIZE FCA
*
*      MVC  FCACSNAM,CSNAME        PLACE CSECT NAME IN FCA
*      MVI  FCARAMST,FCARAMSL      Pass vital areas by address & length
*
*      LA   R2,PARMLIST            R2 --> COMMON API PARMLIST
*      XC   PARMLIST,PARMLIST      CLEAR THE PARAMETER AREA
*
*      LA   R1,RAI_INITIALIZE      R1 --> INITIALIZE STRING
*      ST   R1,0(R2)               STORE 'INITIALIZE' AS FIRST PARM
*      LA   R1,WFCA                R1 --> PTR TO FCA IN PARM LIST
*      ST   R1,4(R2)               STORE PTR TO FCA IN PARM LIST
*      LA   R1,AREA1               R1 --> PTR TO AREA1 IN PARM LIST
*      ST   R1,8(R2)               STORE PTR TO AREA1 START
*      LA   R1,AREA1LEN            R1 --> PTR TO AREA1 LENGTH
*      ST   R1,12(R2)              STORE PTR TO AREA1 LENGTH
*      OI   12(R2),X'80'           MARK LEN AS LAST PARM
*      LA   R1,0(R2)               R1 --> INITIALIZE API PARM LIST
*
*      L    R15,RAIAPI             R15 --> RAIAPI APPL API STUB
*      BALR R14,R15                CALL RAIAPI
*
*      IF (CLC,FCARETC,NE,BZEROS) INITIALIZE REQUEST SUCCESSFUL ?
*      WTO  'INITIALIZE FAILED',ROUTCODE=(11)
*      LA   R15,8                   NO, INDICATE ERROR
*      B    INITAPIX                 EXIT IMMEDIATELY
*      ENDIF
*
*      MVC  AREA1,AREA1LIT         INITIALIZE SAVE AREA 1
*
*      XR   R15,R15                INDICATE SUCCESS
*
*      INITAPIX L  R14,SAVER14
*      BR    R14                   RETURN TO MAINLINE
*-----
*
*      ISSUE TERMINATE API REQUEST
*
*      BUILD TERMINATE PARAMETER LIST AND ISSUE API REQUEST
*-----
TERMAPI ST   R14,SAVER14
*
*      LA   R11,WFCA              R11 --> COMMON FCA FOR API REQUESTS
*      USING FCA,R11              STACK AND OVERLAY FCA MAP
*      XC   WFCA,WFCA              CLEAR TERMINATE FCA
*
*      MVC  FCACSNAM,CSNAME        PLACE CSECT NAME IN FCA

```

```

*
      LA    R2,PARMLIST          R2 --> COMMON API PARMLIST
      XC    PARMLIST,PARMLIST    CLEAR THE PARAMETER AREA
*
      LA    R1,RAI_TERMINATE     R1 --> TERMINATE STRING
      ST    R1,0(R2)             STORE 'TERMINATE' AS FIRST PARM
      LA    R1,WFCA              R1 --> PTR TO FCA IN PARM LIST
      ST    R1,4(R2)             STORE PTR TO FCA IN PARM LIST
      OI    4(R2),X'80'          MARK FCA AS LAST PARM
      LA    R1,0(R2)             R1 --> TERMINATE API PARM LIST
*
      L     R15,RAIAPI           R15 --> RAI API APPL API STUB
      BALR  R14,R15              CALL RAI API
*
      IF (CLC,FCARETC,NE,BZEROS) INITIALIZE REQUEST SUCCESSFUL ?
      WTO   'TERMINATE FAILED',ROUTCDE=(11)
      LA    R15,8                NO, INDICATE ERROR
      B     TERMAPIX             EXIT IMMEDIATELY
      ENDIF
*
      XR    R15,R15              INDICATE SUCCESS
*
      TERMAPIX L R14,SAVER14
      BR    R14                  RETURN TO MAINLINE
*-----*
*
*      DATA DECLARATIONS
*-----*
      DS    0F                  ALIGN
      OUTFILE DCB DDNAME=OUTFILE,MACRF=(PL),DSORG=PS,LRECL=80
      BZEROS DC F'0'
      RAI API DC VL4(RAI API)
      CSNAME DC CL8'SRS8A3S'
*
      AREA1LIT DC 20CL10'WORKAREA 1 ' AREA TO SAVE AT COMMIT
*
      SRS@API                     EXPAND SMART RESTART SERVICES NAMES
*
*
      AREA1 DS CL200
      RECCOUNT DS F
      AREA1SZ EQU *-AREA1
      DS    0H
      AREA1LEN DC AL4(AREA1SZ)
*
      LCLFLAG DS CL1
      OPEN EQU B'10000000'
*
      DS    0F
      WFCA DS (FCALEN)CL1
      SAVEAREA DS 18F
      SAVER14 DS F
*
      PARMLIST DS 6F              AREA TO BUILD PARAMETER LIST
*
      SRS@FCA                     SRS FUNCTION COMMUNICATIONS AREA
*
      END    SRS8A3S

```

Figure 40.2 Subroutine SRS8A3S

40.3 Assembly / Link Edit JCL

Figure 40.3 illustrates the JCL in member SRSBASM3 of the DCASAMP dataset with which to assemble modules SRS8A3M and SRS8A3S and link edit them together.

```
//Jobname JOB ...
/*-----
/*
/*      Please be sure to edit this member with CAPS ON
/*      =====
/*
/*      This job prepares sample multi-CSECT assembler application
/*      SRSBASM3 to verify Smart/RESTART operations:
/*
/*      R E P L A C E  ?raihlq?
/*                  with the high level qualifier of the Smart Jobstream
/*                  Series target library data sets.
/*
/*      SET RAIHLQ=?raihlq?
/*
/*ASMLKED  PROC RAIHLQ=,CSECT=,
/*          UNIT=SYSDA          UNIT NAME
/*
/*-----
/*      Step 1: Assembly step
/*-----
//ASM      EXEC PGM=ASMA90, COND=(4,LT),
//          PARM=(DECK,TERM,BATCH,NOOBJ,NORENT,NOTEST,'XREF(SHORT)',
//          'SYSPARM(1011110110110)')
//SYSLIB   DD DISP=SHR,DSN=SYS1.MACLIB
//          DD DISP=SHR,DSN=&RAIHLQ..DCAMACS
//          DD DISP=SHR,DSN=&RAIHLQ..DCASAMP
//SYSUT1   DD DSN=&SYSUT1,UNIT=VIO,SPACE=(1700,(600,100))
//SYSUT2   DD DSN=&SYSUT2,UNIT=VIO,SPACE=(1700,(300,50))
//SYSUT3   DD DSN=&SYSUT3,UNIT=VIO,SPACE=(1700,(300,50))
//SYSTEM   DD SYSOUT=*,DCB=BLKSIZE=1089
//SYSPRINT DD SYSOUT=*,DCB=BLKSIZE=1089
//SYPUNCH  DD DSN=&OBJSET,UNIT=&UNIT,SPACE=(80,(500,100)),
//          DISP=(NEW,PASS)
//SYSIN    DD DSN=&RAIHLQ..DCASAMP(&CSECT),DISP=SHR
/*-----
/*      Step 2: Link edit (NCAL) should complete with RC=4
/*-----
//NCAL     EXEC PGM=IEWL,REGION=1024K,COND=(4,LT,ASM),
//          PARM=(NCAL,LIST,XREF,REUS,REFR,RENT,NOTEST)
//SYSLMOD  DD DSN=&RAIHLQ..DCANCAL(&CSECT),DISP=SHR
//SYSUT1   DD DSN=&SYSUT1,UNIT=&UNIT,SPACE=(1024,(50,20))
//SYSPRINT DD SYSOUT=*,DCB=(RECFM=FB,LRECL=121,BLKSIZE=1210)
//SYSLIN   DD DSN=&OBJSET,DISP=(OLD,DELETE)
/*-----
/*      Step 3: Link edit (FULL) should complete with RC=8 for
/*      for CSECT SRS8A3M and with RC=4 for CSECT SRS8A3S
/*-----
//LINKEDIT EXEC PGM=IEWL,REGION=2536K,COND=(4,LT,NCAL),
//          PARM='LIST,XREF,LET,NOTEST,SIZE=(512000,64000),NOREUS,NORENT,NOREFR'
//SYSLIB   DD DSN=&RAIHLQ..DCANCAL,DISP=SHR
//SYSUT1   DD DSN=&SYSUT1,UNIT=&UNIT,SPACE=(1024,(50,20))
//SYSPRINT DD SYSOUT=*,DCB=(RECFM=FB,LRECL=121,BLKSIZE=1210)
//SYSLMOD  DD DSN=&RAIHLQ..DCALOAD,DISP=SHR
//SYSLIN   DD DUMMY
//          PEND
/*-----
/*      Assembly CSECT SRS8A3M and build load module SRS8ASM3
/*-----
//SRS8A3M  EXEC ASMLKED,RAIHLQ=&RAIHLQ,CSECT=SRS8A3M,UNIT=SYSDA
//LINKEDIT.SYSLIN DD *
//          INCLUDE SYSLIB(SRS8A3M)
//          INCLUDE SYSLIB(SRS8A3S)
```



```

ENTRY    SRS8A3M
MODE     AMODE(24),RMODE(24)
NAME     SRS8ASM3(R)
/*
/*-----
/*      Assembly CSECT SRS8A3S and build load module SRS8ASM3
/*-----
//SRS8A3S EXEC ASMLKED,RAIHLQ=&RAIHLQ,CSECT=SRS8A3S,UNIT=SYSDA
//LINKEDIT.SYSLIN DD *
INCLUDE SYSLIB(SRS8A3M)
INCLUDE SYSLIB(SRS8A3S)
ENTRY    SRS8A3M
MODE     AMODE(24),RMODE(24)
NAME     SRS8ASM3(R)
/*
/*

```

Figure 40.3 JCL to build the Smart/RESTART Sample application SRS8ASM3

40.4 Execution JCL

Finally, Figure 40.4 illustrates the JCL in member SRSRASM3 of the DCASAMP dataset with which to run the SRS8ASM3 application.

```

/Jobname JOB ...
/*-----
/*
/*      This job executes sample assembler application SRS8ASM3. The
/*      main assembler program SRS8A3M reads records and invokes a sub-
/*      program to write records and increment a record counter. This
/*      module calls the Smart/RESTART call level interface (module
/*      RAIAP1) to explicitly initialize and terminate the Smart/RESTART
/*      environment.
/*
/*      In contrast, Repositionable I/Os are performed transparently
/*      using standard z/OS OPEN, GET, PUT and CLOSE macros. This
/*      routine uses static areas and is non-REUSABLE.
/*
/*      R E P L A C E  ?raihlq?
/*                  with the high level qualifier of the Smart Jobstream
/*                  Series target library data sets.
/*
// SET RAIHLQ=?raihlq?
/*-----
/*      Run sample assembler application SRS8ASM3
/*-----
//SRS8ASM3 EXEC PGM=SRS8ASM3,REGION=4M
//STEPLIB DD DISP=SHR,DSN=&RAIHLQ..DCALOAD
//SRSCHECK DD DISP=SHR,DSN=&RAIHLQ..DCACKPTV
//INFILE DD DISP=SHR,DSN=&RAIHLQ..DCASRMIN
//OUTFILE DD DISP=SHR,DSN=&RAIHLQ..OUTFILE2
//SYSPRINT DD SYSOUT=*
/*SYSABEND DD SYSOUT=*
/*SRSOFF DD DUMMY /* Uncomment this DD to disable Smart/RESTART
/*
//RAINPUT DD *
CLEAR_CKPT(BEFORE)
CKPT_SIZE(20) CKPT_FREQ(1) BULK_CKPT(VSAM)
MSG_DISPLAY(DEBUG)
RMI(NONE) REPOSITION(ONLY,(INFILE,OUTFILE))
/*
/*

```

Figure 40.4 JCL to run SRS8ASM3 sample assembler-language application

Checkpointing Media Alternatives

Smart/RESTART supports a range of checkpointing options which represent a trade-off between space conservation, media preference and restart flexibility. This chapter considers checkpointing options in detail.

Smart/RESTART supports four media for its checkpoint data. To enable Smart/RESTART to dynamically allocate a checkpoint dataset for both initial and restart runs specify (or default) the AUTO_JOBID parameter as

```
AUTO_JOBID( JOBNAME | SYSTEMANDJOBNAME ) .
```

This technique applies to every checkpoint medium except TAPE.

VSAM file

Smart/RESTART normally records its checkpointing information in a VSAM Linear Dataset (LDS). This is the checkpoint media option RAI recommends. In the DB2 environment, Smart/RESTART records supplementary information in a DB2 table to ensure that application and RRS compliant resources (such as DB2, MQ, and/or IMS) remain synchronized. Smart/RESTART writes most checkpoint data to the VSAM LDS. Just a small amount of synchronizing data is recorded in the DB2 table.

See the member SRSJCKPV of the DCACNTL library for JCL to define a Smart/RESTART checkpoint file as a VSAM LDS.

BDAM file

Smart/RESTART can also record its checkpointing information in a BDAM dataset. In the DB2 environment, Smart/RESTART records supplementary information in a DB2 table to ensure that application and DB2 resources remain synchronized. Smart/RESTART writes most checkpoint data to the BDAM file. Just a small amount of synchronizing data is recorded in the DB2 table.

TAPE file

Smart/RESTART can write all its checkpoint data to a sequential tape file. Checkpointing to tape is well-suited for those applications which may need to restart from *any* previous checkpoint. Using tape as the checkpointing media supports a potentially unlimited number of checkpoints, provided enough tape volumes are available.

DB2 Table

Smart/RESTART can record all checkpoint data (both synchronizing data as well as working storage contents and sequential file position) in a DB2 table. Using DB2 as the sole checkpoint media is feasible for applications which access RRS compliant resources such as DB2, MQ and IMS. DB2 checkpointing is simple to implement since there is no need to define a checkpoint DD statement in the JCL. However, the performance of DB2 checkpointing suffers in comparison to both the VSAM LDS and BDAM options. In addition, using DB2 as the sole checkpoint media supports restart from the **last** checkpoint only.

Overview

By default, application restart occurs from the **last** (most recent) checkpoint, although restart from a prior checkpoint is also possible. The RESTART_ID parameter (defined in the RAINPUT stream) can explicitly identify the checkpoint from which restart is requested.

NOTE: *The Smart/RESTART Reference Manual documents the RAINPUT parameters discussed in this chapter.*

The number of checkpoints Smart/RESTART maintains (from which you can request a restart) is governed by the CKPT_DEPTH parameter. However, restart from other than the last checkpoint requires manual intervention when the application also accesses DB2, IMS and/or MQ resources. When such protected resources are accessed, they must be restored to their state at the point in time which corresponds to the checkpoint selected for restart (as described in Chapter 50.3 - Point-in-Time Recovery with Smart/RESTART).

Checkpointing adds overhead in relation to such factors as checkpoint frequency, the number and size of storage areas to be saved and the number of repositionable files. The checkpointing process typically requires between 1 and 2 seconds per megabyte. The following formula can be used to estimate the elapsed time required to complete a checkpoint (using either VSAM or BDAM as the checkpoint medium):

$$ET = (((WS+4096)/4096) + (OF*2)) * .0040 + (.01/MR) + RT$$

where	WS	--> Aggregate Working Storage size in bytes
	OF	--> Number of repositionable files open at checkpoint time
	RT	--> Elapsed time to update one DB2 row (< .005)
	MR	--> CPU MIP rate (e.g. 2.3 = 2300000 instructions per sec)

41.1 Checkpointing to a VSAM LDS

Smart/RESTART can store bulk checkpoint data in a VSAM LDS (linear dataset). RAI recommends VSAM LDS checkpointing because it significantly outperforms the BDAM, DB2 and TAPE options.

Checkpoint overhead depends on checkpoint frequency, the amount of data saved and the checkpoint media. Checkpointing overhead can be reduced by limiting the amount of data saved. Smart/RESTART provides a number of techniques to identify the storage areas to save at checkpoint time.

The major advantage to using VSAM as the bulk checkpoint medium is you no longer need to be concerned with identifying and saving just critical storage areas. The VSAM LDS checkpointing option lets you save application storage in its entirety, regardless of its size, without a performance penalty. This relieves developers of the burden of determining which fields Smart/RESTART should save at checkpoint time and restore at restart time. For programs prepared with Smart/PRECOMPILER, this can eliminate the need to make any source code changes.

Unlike the other checkpoint media, VSAM LDS checkpointing writes out just the data changed since the last checkpoint. As such, an application could load a gigabyte file into a read only, in-storage table and *still* save all working storage at each checkpoint -- without degrading performance.

The one drawback to VSAM LDS checkpoint files is the lack of full Smart/MONITOR functionality because an active restartable application cannot share its checkpoint dataset with Smart/MONITOR. Full Smart/MONITOR functionality for an active job step requires either BDAM or DB2 as the checkpoint medium.

Smart/RESTART's default checkpoint medium is VSAM but this default can be changed at installation time. To identify the checkpoint medium in use, check with the Smart/RESTART administrator or look for the SRSI182I message issued at runtime as shown below.

```
SRSI182I - Primary checkpoint medium is VSAM
```

41.1.1 JCL Considerations

RAI recommends that the checkpoint dataset be pre-allocated as illustrated in Figure 41.1. By pre-allocating the checkpoint file, you can resubmit the JCL without change, in the event the application fails and needs to be restarted. Alternatively, Section 4.1.2.1 of Chapter 4 describes how Smart/RESTART can dynamically allocate a checkpoint file during initial and restart runs.

```
//ALLOCATE EXEC PGM=IEFBR14
//SRSCHECK DD DSN=checkpoint.dataset.name,
//          DISP=(NEW,CATLG,CATLG),
//          DCB=RECORG=LS,           (1)
//          SPACE=(TRK,60)           (2)
```

where

- (1) RAI recommends the checkpoint file be defined as a VSAM LDS (linear data set). You need not specify an LRECL or BLKSIZE.
- (2) A space allocation of 60 tracks for the checkpoint dataset should be ample for nearly all applications. See the CKPT_SIZE and CKPT_DEPTH parameters described in Chapter 14 of the Smart/RESTART Reference for further details.

Figure 41.1 *Pre-allocating a VSAM LDS Checkpoint File*

41.2 Checkpointing to a BDAM File

Smart/RESTART writes its checkpoint information to a BDAM dataset on disk when you specify **BULK_CKPT(BDAM)** in the RAINPUT stream or adopt BDAM checkpointing as the installation default. Smart/RESTART also records a small amount of synchronizing data in a DB2 table when an application also accesses RRS compliant resources such as DB2, MQ and IMS. Conversely, no synchronizing data is written to DB2 if the application does not access any RRS compliant resource managers.

Typically, Smart/RESTART rotates between a pair of checkpoint entries in a single checkpoint dataset (although more than two checkpoint entries can be maintained to support point-in-time recovery). In the default case, Smart/RESTART overlays the older of the two checkpoints with the current checkpoint.

41.2.1 JCL Considerations

Figure 41.2 illustrates how to pre-allocate a BDAM checkpoint file. By pre-allocating the checkpoint file, you can resubmit the JCL without change, in the event the application fails and needs to be restarted.

```
//ALLOCATE EXEC PGM=IEFBR14
//SRSCHECK DD DSN=checkpoint.dataset.name,
//          DISP=(NEW,CATLG,CATLG),
//          DCB=DSORG=DA,                <--- (1)
//          UNIT=SYSDA,VOL=SER=volser,
//          SPACE=(TRK,60)                <--- (2)
```

where

- (1) Define the BDAM file checkpoint file as a direct access dataset.
- (2) The default space allocation for a DASD checkpoint dataset is 60 tracks.

Figure 41.2 Pre-allocating a BDAM Checkpoint File

The space allocation for the BDAM file depends on two factors:

- The number of discrete checkpoints that Smart/RESTART should maintain
- the size of each checkpoint

These values are specified respectively through the CKPT_DEPTH and CKPT_SIZE parameters of the Smart/RESTART profile.

41.2.2 Checkpointing to a BDAM File -- Initial Run

Figure 41.3 illustrates the initial run of a Smart/RESTART application that makes use of a pre-allocated BDAM checkpoint file. The numbers in the right hand margin (within parentheses) correspond to the numbered annotations which follow the JCL.

```
//SRSSTEP EXEC PGM=EXAMPLE
//SRSCHECK DD DSN=check.point.dataset,DISP=OLD    <--- (1)
//RAINPUT DD *
BULK_CKPT(BDAM)                                   <--- (2)
//
```

where

- (1) The SRSCHECK DD statement defines the pre-allocated BDAM file into which Smart/RESTART writes checkpoint data. Since the checkpoint dataset already exists, the same JCL can be resubmitted should it be necessary to restart the application.
- (2) The BULK_CKPT parameter (specified in the RAINPUT stream) directs Smart/RESTART to write its checkpoint information to a BDAM dataset. Typically the BULK_CKPT parameter is defined during Smart/RESTART installation as a site default. Thus, BULK_CKPT need not be specified explicitly in each restartable jobstep.

Figure 41.3 *Initial run using pre-allocated BDAM Checkpoint File*

41.2.3 Checkpointing to a BDAM File -- Restart Run

Figure 41.4 illustrates the restart JCL for the job submitted initially in Figure 41.3 above. The initial and restart JCL are identical since the SRSCHECK DD statement references a pre-allocated checkpoint dataset.

```
//SRSSTEP EXEC PGM=EXAMPLE
//SRSCHECK DD DSN=check.point.dataset,DISP=OLD
//RAINPUT DD *
BULK_CKPT(BDAM)
//
```

Figure 41.4 *Restart run using pre-allocated BDAM Checkpoint File*

41.3 Checkpointing to DB2

You can direct Smart/RESTART to record the contents of application storage and sequential file position in the same DB2 table used to maintain synchronization information. This *bulk checkpoint data* is typically purged when the jobstep completes successfully. Alternatively, Smart/RESTART leaves this data intact when an application terminates abnormally -- to allow a restart run.

Smart/RESTART writes its checkpoint information to a DB2 table when you specify `BULK_CKPT(DB2)` in the RAINPUT stream or adopt DB2 checkpointing as the site default. DB2 checkpointing is suitable for restartable applications that access RRS compliant resources such as DB2, MQ and/or IMS.

NOTE 1: DB2 checkpointing supports restart from the *LAST checkpoint only*. Should your application need to restart from some previous checkpoint, you must use either VSAM LDS, TAPE or BDAM checkpointing instead.

NOTE 2: You should avoid canceling a Smart/RESTART job via the MVS Cancel command when bulk checkpoint data is stored in DB2. Instead, use either the Smart/RESTART QUIESCE or FORCE commands. This prevents the -906 SQLCODE that occurs during Smart/RESTART cleanup processing. Although this error does not compromise restartability, it reduces the amount of jobstep information accessible to Smart/MONITOR.

41.3.1 RAINPUT Parameters for DB2 Checkpointing

The following Smart/RESTART Profile parameters pertain to DB2 checkpointing. The Smart/RESTART Reference discusses each of these parameters in detail.

- `BULK_CKPT(DB2)`
- `DCR(option)`
- `DCR_AUX(option)`
- `JID(SSSSSS)` or `JOBID(SSSSSS)`
- `RESTART_ID(LAST)`
- `AUTO_JOBID(option)`
- `VERIFY_JID(option)` or `VERIFY_JOBID(option)`

41.3.2 Checkpointing to a DB2 Table -- Initial Run

Figure 41.5 illustrates the initial run of a Smart/RESTART application that writes all checkpoint data to a DB2 table. The numbers in the right hand margin (within parentheses) correspond to the numbered annotations which follow the JCL.

NOTE: When DB2 checkpointing is utilized, no checkpoint dataset need be defined. Therefore, the JCL in Figure 41.5 does not include an *SRSCHECK DD* statement that defines a checkpoint dataset.

```
//SRSSTEP EXEC PGM=EXAMPLE
//RAINPUT DD *
BULK_CKPT(DB2)           <--- (1)
JOBID(EXAMPLE_JID)       <--- (2)
//
```

where

- (1) The BULK_CKPT parameter (specified in the RAINPUT stream) directs Smart/RESTART to record all checkpoint information in a DB2 table. A bulk checkpointing default is defined during Smart/RESTART installation. Thus, BULK_CKPT(DB2) need not be specified explicitly, except to override the site default.
- (2) The JOBID parameter specifies a job identification string which relates the initial run of a batch job with any associated restart run(s). Smart/RESTART maintains the JID across multiple runs. This Job identifier is not released for reuse until the job completes successfully. A JOBID value **must be specified explicitly** when bulk checkpoint data is recorded within DB2.

Figure 41.5 Initial run using DB2 as the checkpoint medium

41.3.3 Checkpointing to a DB2 Table -- Restart Run

Figure 41.6 illustrates the restart JCL for the job submitted initially in Figure 41.5 above. The initial and restart JCL are identical. Note that checkpointing to DB2 supports restart from the **last checkpoint only**.

```
//SRSSTEP EXEC PGM=EXAMPLE
//RAINPUT DD *
BULK_CKPT(DB2)
JOBID(EXAMPLE_JID)
//
```

Figure 41.6 Restart run using DB2 as the checkpoint medium

41.4 Checkpointing to TAPE

Smart/RESTART writes its checkpoint information to a sequential tape file when you specify **BULK_CKPT(TAPE)** in the RAINPUT stream or adopt tape checkpointing as the site default. The TAPE option should be selected should your application need to restart from *any* previous checkpoint as well as the last. Tape checkpointing enables Smart/RESTART to retain an unlimited number of checkpoints from which to restart.

Smart/RESTART issues message SRS031I to describe each checkpoint. For example:

```
SRS031I - CHECKPOINT 00002008 is complete. CHECKPOINT ID is TAPE1A0020234
```

In this example, the value 00002008 identifies the checkpoint sequence number. Smart/RESTART maintains a unique sequence number across the initial run and any restart runs. In addition, message SRS031I identifies the checkpoint in terms of the tape volume on which it is recorded and the relative block number (on that volume) where the checkpoint data begins. The ID of a tape checkpoint is a twelve character string comprised of a six character volume name and six character relative block number. Thus, tape checkpoint 'TAPE1A020234' is located on tape volume TAPE1A starting at relative block number 020234.

Smart/RESTART issues message SRS804 when a restartable application using tape checkpointing terminates abnormally. Message SRS804 identifies the **last** successfully processed checkpoint in terms of its tape volume serial number and relative block number. This last checkpoint is typically the one most suitable for restart.

41.4.1 RAINPUT Parameters for Tape Checkpointing

The following Smart/RESTART Profile parameters pertain to tape checkpointing. The Smart/RESTART Reference discusses each of these parameters in detail.

- BULK_CKPT(TAPE)
- RESTART_ID(VVVVVVRRRRRR)

41.4.2 Checkpointing to Tape - Initial Run

Figure 41.7 illustrates the initial run of a Smart/RESTART application that writes all checkpoint data to a tape file. The numbers in the right hand margin (within parentheses) correspond to the numbered annotations which follow the JCL.

```
//SRSSTEP EXEC PGM=EXAMPLE
//SRSCHECK DD DSN=tape.checkpoint.dataset,DISP=(MOD,KEEP), <--- (1)
//          VOL=SER=(tapela,tapelb,tapelc,. . .) <--- (2)
//          LABEL=(1,SL),
//          UNIT=TAPE
//RAINPUT DD *
BULK_CKPT(TAPE) <--- (3)
//
```

where

- (1) The SRSCHECK DD statement defines a tape file in which Smart/RESTART records checkpoint data. Smart/RESTART appends new checkpoints to those already recorded on tape because MOD is specified as the dataset disposition. This allows the same JCL to be used for both initial and restart runs.
- (2) The VOL parameter defines several tape volumes for the checkpoint file.
- (3) The BULK_CKPT parameter (specified in the RAINPUT stream) directs Smart/RESTART to write all checkpoint information to a tape dataset. A bulk checkpointing default is defined during Smart/RESTART installation. Thus, BULK_CKPT(TAPE) need not be specified explicitly, except to override the site default.

Figure 41.7 Initial run using a Tape checkpoint file

41.4.3 Checkpointing to Tape -- Restart Run

Figure 41.8 illustrates the restart JCL for the job submitted initially in Figure 41.7. Suppose the initial run failed after completing 1000 checkpoints. However, we want to restart from checkpoint number 800, rather than the last checkpoint. In such a case, a `RESTART_ID` must be specified as a parameter in the `RAINPUT` stream.

NOTE: When checkpointing to tape, the value `LAST` is *invalid* as a `RESTART_ID`. The restart ID must be specified as a two part string of the form `VVVVVVTTTTTT` where `VVVVVV` specifies the tape volume serial number and `TTTTTT` is the relative block number on the tape volume where the checkpoint data begins.

```
//SRSSTEP EXEC PGM=EXAMPLE
//SRSCHECK DD DSN=tape.checkpoint.dataset,DISP=(MOD,KEEP),
//          VOL=SER=(tape1a,tape1b,tape1c,
//          LABEL=(1,SL),
//          UNIT=TAPE
//RAINPUT DD *
BULK_CKPT(TAPE)
RESTART_ID(TAPE1A020234)                                <- - - (1)
//
```

where

- (1) To restart from a *particular* checkpoint, specify the checkpoint's location as a twelve character `RESTART_ID`. In this example, the `RESTART_ID` requests restart from the checkpoint located on tape volume `TAPE1A`, starting at relative block number `020234`. This value was obtained from the `SRS031I` message associated with the checkpoint whose sequence number was 800.

Smart/RESTART uses this `RESTART_ID` to request that the appropriate tape be mounted. Next, Smart/RESTART positions to the relative block number specified by the second six characters of the `RESTART_ID`.

Figure 41.8 Restart run using a Tape checkpoint file

When restarting from the *last* checkpoint, working storage contents, sequential file position and RRS compliant resources managed by the likes of DB2, IMS, and/or WebSphere MQ are automatically in synch. When restarting from a prior checkpoint, Smart/RESTART automatically restores working storage contents and sequential file position to a state consistent with that checkpoint. However, DB2, IMS and/or MQ resources must be manually restored to a state consistent with that point in time (as described in Chapter 50.3 Point-in-Time Recovery with Smart/RESTART).

41.5 Tying It All Together

Smart/RESTART supports four checkpoint media, three types of Job IDs, two allocation schemes for checkpoint files and varying requirements for initial and restart runs. The table below attempts to tie all these combinations together and document the various options and requirements. The columns of the table represent the three different types of Job IDs while the rows pertain to the four different checkpoint media.

The three types of Job IDs derive from two different parameters. JOBID refers to an explicit Job ID specified via the JOBID parameter. Smart/RESTART can also generate a job identifier *automatically* as described in Chapter 14 of the Smart/RESTART Reference.

A Job ID based on the z/OS or OS/390 job name results from AUTO_JOBID(JOBNAME), a Job ID based on the job name and MVS system name results from AUTO_JOBID(SYSTEMANDJOBNAME) while a timestamp based Job ID results from an AUTO_JOBID(TIMESTAMP) specification.

	Timestamp	Job Name or SystemAndJobname	Explicit JOBID
VSAM	Allowed	Full Support (1)	Full Support
BDAM	Allowed	Full Support (1)	Full Support
TAPE	Allowed (2)	Allowed (2)	Allowed (2)
DB2	Not allowed	Allowed	Allowed

"Full Support" means the combination of Job ID and checkpoint media is allowed and **can also** be used with dynamic allocation of checkpoint files in a restart run

"Allowed" means the combination of Job ID and checkpoint media is allowed but can **not** be used to dynamically allocate a checkpoint file in a restart run. Remember that dynamic allocation of checkpoint files does not apply to DB2 tables.

- (1) Smart/RESTART allows the use of different job names on the JCL JOB statement for the initial run and restart run(s) provided an explicitly specified Job ID (JOBID) or timestamp based Job ID is used. However, when the Job ID derives from the AUTO_JOBID(JOBNAME|SYSTEMANDJOBNAME) parameter, the initial run and restart run must use the same job name if the checkpoint file is dynamically allocated. Statically allocated checkpoint files allow different job names even with AUTO_JOBID(JOBNAME|SYSTEMANDJOBNAME).
- (2) TAPE checkpoint datasets must be statically allocated via JCL – for both initial and restart runs.

Automatic Restart Facilities

42.1 Automatic Restart Overview

Typically, when a Smart/RESTART job abnormally terminates, the user must correct the cause of theabend (e.g. allocate more disk space) and *manually* resubmit the JCL in order to restart the application from the last checkpoint. Some conditions which cause a failure may be temporary -- such as SQLCODE -911 which denotes resource contention.

As an alternative, Smart/RESTART can *automatically* restart failing applications without manual intervention or re-submission of JCL. Thus, 'temporary' failures which are conducive to automatic restart (such as a -911 SQLCODE) may be declared 'retriable'. When such retriable failures occur, Smart/RESTART allows the application to resume execution in a completely automatic fashion.

Smart/RESTART's automatic restart facility restores working storage and sequential file position to their state as of the last checkpoint. Your application resumes execution at entry to the program (just like a manually submitted restart) -- not following a COMMIT or ROLLBACK request.

42.2 Automatic Restart Operation

Automatic restart is controlled through a pair of Smart/RESTART runtime parameters through which you list particular MVS abend and/or SQLCODE values that are eligible for automatic restart. These are typically defined at installation time but can be specified at run-time through the RAINPUT file.

The main operational differences between manual and automatic restart are as follows:

Manual restart is deferred until the JCL is physically resubmitted. JES assigns the resubmitted job a new job number and may initiate it in a different address space.

Automatic restart occurs immediately -- without manual intervention. The automatically restarted job retains its original JES job number and executes under the same initiator and MVS address space.

Despite the fact that the job runs in the same address space and is effectively viewed as the same job by the operating system, it behaves as if it were manually resubmitted for restart. Thus, all resources at the time of failure are released and all work processed after the last checkpoint is backed out.

Should the conditions causing the original failure persist, then any automatic restart will also fail. For example, a coding error that produces an S0C4 abend will fail repeatedly. Therefore, automatic restart from these types of errors is not appropriate. In contrast, a -911 SQLCODE is more conducive to automatic restart since the resource contention that produced the -911 may not persist.

Automatic restart from a -911 SQLCODE allows the application to reprocess a failing unit-of-work as follows:

Smart/RESTART backs out changes since the last checkpoint and abnormally terminates the application. Smart/RESTART then cleans-up jobstep resources and re-enters the application as if it was manually resubmitted for restart.

The restarted application resumes execution from the last successful checkpoint. As such, the failing unit-of-work is retried. If contention persists, the unit-of-work will fail once more. This sequence of failure, automatic restart and unit-of-work retry will repeat until the unit-of-work is successfully processed or the user specified retry count is exceeded.

NOTE: Every Smart/RESTART application is made aware of a restart condition and may take action to avoid any errors it previously encountered.

42.2.1 Restrictions

Smart/RESTART's automatic restart facilities will not work if any of the following conditions are true:

- An application runs in a DLIBATCH or IMS BMP environment
- The ESTAE attribute of the Smart/RESTART profile is set OFF
- Automatic restart is requested for an MVS abend that does not permit task retry (e.g. S222 and S13E)

42.3 Interactive Restart

Interactive restart affords a third alternative to manual and automatic restart. For each condition defined as eligible for restart, the decision on whether to restart can be deferred until the actual time of failure. When the defer option is chosen, Smart/RESTART suspends the job when it detects a pre-defined error condition. At that point, an authorized user can examine the job, its input buffers, working storage, etc through Smart/MONITOR. Once this analysis is complete, recovery/restart can proceed or the application can be terminated immediately. See the description of Smart/MONITOR in Part IV for further details.

42.4 Requesting Automatic Restart

Automatic restart services are governed through several options of the Smart/RESTART runtime profile. The following discussion briefly describes the function of these runtime options. Chapter 14 of the Smart/RESTART Reference documents their syntax and discusses individual operands and values in detail. Several examples in this chapter illustrate their use in various circumstances.

42.4.1 ABEND_TRAP parameter

The ABEND_TRAP parameter specifies the MVS abend codes that Smart/RESTART should attempt to intercept and subsequently restart. Interception entails either immediate or deferred recovery action. Only abend codes specified through the ABEND_TRAP directive are eligible for recovery processing.

Note that ABEND_TRAP lists just MVS abend codes -- not SQL error conditions. Use the SQL_TRAP command (described next) to specify SQL error conditions.

For example, the following ABEND_TRAP specification requests automatic restart whenever Smart/RESTART detects a SOC7 or U4010 abend.

```
ABEND_TRAP((SOC7,R),(U4010,R))
```

42.4.2 SQL_TRAP parameter

The SQL_TRAP parameter requests that Smart/RESTART automatically rollback all application processing and resource manager activity in a coordinated fashion to the previous point of consistency. When Smart/RESTART detects a condition defined through SQL_TRAP, it issues an internal user abend to initiate automatic restart procedures.

The following SQL_TRAP specification requests that Smart/RESTART perform automatic restart whenever an SQL -911 occurs. Smart/RESTART may retry 5 times before letting the application terminate abnormally.

```
SQL_TRAP(ON,911,AUTO,5)
```

NOTE: In addition to being described in Chapter 14 of the Smart/RESTART Reference, installation wide SQL_TRAP defaults can be defined as described in Chapter 5 of the Smart/RESTART Installation and Customization Guide.

42.4.3 AUTO_RECOVER command

The AUTO_RECOVER parameter can be used to override existing settings that may have been specified via the SQL_TRAP and ABEND_TRAP parameters. This may be useful when a restartable application is *already* executing and some global change to recovery behavior is desired.

For example, a **run-time** decision can be made to enable or disable all automatic restart specifications previously defined through the ABEND_TRAP parameter. Another use is to override the number of retries Smart/RESTART may attempt before abandoning efforts to retry the failed unit-of-work

The following AUTO_RECOVER specification directs Smart/RESTART to stop further reprocessing attempts after a total of 7 retries.

```
AUTO_RECOVER(ON,7)
```

Abend Recovery Facilities

43.1 Abend Recovery Overview

Smart/RESTART allows an application to resume execution from a point of consistency with respect to application processing and resource manager updates. The nature of the failure governs what must be done before the job can be restarted successfully. Remedial actions include increasing the space allocation for an output file that ran out of space, recompiling an application with a logic flaw or deleting input records that contain bad data. Once corrections are made, the application can be resubmitted for restart.

An attempt to process invalid data is a common cause of program failure. To recover, you can manually delete the invalid record from the input file and then resubmit the application for restart. Alternatively, Smart/RESTART provides several facilities that simplify and even automate abend recovery.

NOTE: All these facilities require that the ESTAE option of the Smart/RESTART profile be set ON.

You can direct Smart/RESTART to flag the current input record so it can be deleted or replaced during a subsequent restart. Smart/RESTART can even bypass the entire unit-of-work which contains the invalid record.

NOTE: The option to bypass an entire unit-of-work can only be used in conjunction with the External Commit facilities described in Chapter 44.

Smart/RESTART initiates these recovery actions (e.g. skip or edit an input record) in accordance with the abend conditions specified through the ABEND_TRAP parameter. Smart/RESTART does not recognize invalid data within an input record. Rather, Smart/RESTART assumes the record to be invalid because a user defined abend occurred while the application was processing this record.

Smart/MONITOR allows you to analyze the application (its working storage, file buffers, etc.) – on-line, at the point of failure. At that time, you can edit the input record or simply request that it be bypassed upon restart.

43.2 Defining Eligible Abend Codes

You can define what abend recovery Smart/RESTART should perform through the ABEND_TRAP parameter. ABEND_TRAP is described in Chapter 14 of the Smart/RESTART Reference. Each abend code defined by ABEND_TRAP specifies a corresponding restart option that indicates whether manual or automatic restart should be performed.

43.3 Selecting Abend Recovery Options

Automatic and Manual Abend Recovery

Smart/RESTART can perform abend recovery *automatically*, without operator intervention. Alternatively, you can direct abend recovery *manually* through the Smart/MONITOR dialogs. Abend recovery occurs in relation to the file specified by the ABEND_SKIP parameter.

Automatic actions include skipping the error record when a restart is submitted. When automatic action is requested, Smart/RESTART collects the information it needs to bypass one or more records during a subsequent restart. Automatic action requires that the ABEND_SKIP parameter request SKIP processing.

Alternatively, you can defer action until the point of abend (provided the ABEND_SKIP parameter specifies the EDIT option). The EDIT option suspends the application for up to one hour -- at the point of failure. This allows time for manual analysis and alteration of an input buffer's contents through Smart/MONITOR. You can also choose to skip the record or take no action. Smart/RESTART lets the abend proceed after which you can restart the application. Should the input buffer be modified, the application receives the *corrected* input buffer upon restart and avoids a recurrence of the abend.

43.4 Saving Images of Skipped or Replaced Records

Auxiliary Recovery Control Files

The ABEND_SKIP directive causes Smart/RESTART to perform SKIP or EDIT processing during abend recovery. You can also define an auxiliary file into which Smart/RESTART can write skipped or edited records.

NOTE: You can ignore DCB attributes when allocating these auxiliary files since Smart/RESTART obtains their characteristics from the files defined by the ABEND_SKIP parameter.

Space allocation for these auxiliary files must be sufficient to accommodate either a single record (when the ONLY option is specified in the ABEND_SKIP command) or as many records as might be read during a single unit-of-work (when the UOW option is specified).

43.5 Using Smart/MONITOR During Abend Recovery

Whenever **ABEND_SKIP** requests the **EDIT** option, Smart/RESTART suspends the application when it detects an abend defined through **ABEND_TRAP**. Your application remains suspended while it waits for a Smart/MONITOR session to connect to it. Once connected, an authorized Smart/MONITOR user can display and modify working storage and input file buffers to attempt to correct the data responsible for an abend. Smart/MONITOR is documented in Part IV of this manual. Here, we merely present an overview of the steps required to connect to the suspended application and communicate with it.

43.6 Conducting an Interactive Abend Recovery Session via Smart/MONITOR

When Smart/MONITOR displays its Abend Recovery Panel, you can edit an input record by entering 'Y' to the left of the related query. Editing can be conducted in character or hexadecimal format. Once editing is complete, issue the **SAVE** command to commit the changes.

The suspended application then resumes the abend process. Smart/RESTART automatically restarts the step (with the same job number, JES initiator and MVS address space) when the abend action specifies 'R'. Alternatively, specify 'S', to manually submit the restart job.

Of course, you are not obliged to edit storage or records. Instead, you can SKIP the record containing invalid data. In addition, when the external commit facility (described in Chapter 44) is used, the entire unit-of-work can be bypassed during the subsequent restart.

NOTE: The **ABEND_SKIP** directive must specify the **EDIT** option to enable Smart/MONITOR to interact with your application during abend recovery.

Externally Defined Checkpoints

44.1 External Checkpoint Overview

As an alternative to coding checkpoints *within* your application, Smart/RESTART also lets you define units of work and their associated checkpoints *external* to the program. Consequently, operating behavior can be altered without recompiling or relinking an application. Such externally defined units-of-work may be used in conjunction with the Abend Recovery and Automatic Restart facilities described in previous chapters.

Smart/RESTART supports external unit-of-work definition on the basis of the following criteria:

- The COMMIT_DB2 parameter controls how often Smart/RESTART should initiate the physical checkpoint process based on the number of SQL requests the application issues.
- The COMMIT_DB2_CHANGES parameter controls how often Smart/RESTART initiates the physical checkpoint process based on the number of SQL requests the application issues that change data (DELETE, INSERT, and/or UPDATE statements).
- The COMMIT_TIME parameter controls how often Smart/RESTART initiates the physical checkpoint process based on elapsed time. Authorized users may find it useful to issue this command from the MVS console or through Smart/MONITOR.
- The COMMIT_CURSOR parameter controls how often Smart/RESTART initiates the physical checkpoint process based on references to a named DB2 cursor with respect to count, content or elapsed time.
- The COMMIT_FILE parameter controls how often Smart/RESTART initiates the physical checkpoint process based on references to a named input or output driver file with respect to count, content or elapsed time.

For example, the `COMMIT_FILE` parameter directs Smart/RESTART to schedule a checkpoint based on the examination of an input or an output record. An externally defined unit-of-work may be based on the contents of one or more fields within a record or on record counts maintained by Smart/RESTART.

This technique assumes a main Read or Write process loop whose *driver* file contains records that determine when a checkpoint should occur. Smart/RESTART triggers a checkpoint based on record content or on a count of records read or written. In effect, the logic which detects the completion of a unit-of-work is relocated from within the application to Smart/RESTART.

While not recommended, it is possible to combine externally defined checkpoints with checkpoints coded within the application. In this hybrid case, the manipulation of checkpoint frequency via the `CKPT_PACE` parameter affects only checkpoints issued by the application unless the `COMMIT_FILE_PACING` parameter is set to `ON`.

The remainder of this chapter uses the `COMMIT_FILE` parameter to illustrate how to define an external unit-of-work.

44.2 The Input Driver File processing model

44.2.1 Processing Model attributes

The COMMIT_FILE parameter supports applications that adhere to the following guidelines.

- The contents or counts of records associated with a single input file govern when a checkpoint should occur. This input file is known as the *driver* file.
- The driver file is read via the Repositionable Sequential Access Method (RSAM).
- A central Read/Process loop must exist. Other input and output files may be accessed. They may also be repositionable. However, they do not define the boundaries of a unit-of-work.
- DB2 applications must maintain cursor position using the coding techniques described in Chapter 2 which maintain cursor position across checkpoints and can reestablish cursor position during restart. Your cursor declarations should specify the WITH HOLD attribute. They should also include a WHERE clause that can reposition the cursor on restart. Alternatively, DB2 cursor position can be reestablished via a REPOSITION_CURSOR command supplied in the RAINPUT stream. The REPOSITION_CURSOR_AT and REPOSITION_CURSOR_AFTER commands are documented in Chapter 12 of the *Smart/RESTART Reference* (Publication SJS-005).
- Checkpoints triggered by external checkpoint criteria are not effected by the checkpoint frequency definitions CKPT_PACE, CKPT_FREQ and CKPT_TIME unless the COMMIT_FILE_PACING parameter is set to ON.

44.2.2 Application Processing Scenario

As each application READ is issued, Smart/RESTART uses the Unit-of-Work definition (specified through the COMMIT_FILE directive) to determine whether the record just read constitutes a new logical unit-of-work. When Smart/RESTART detects a 'control break' (the completion of the previous unit-of-work and the commencement of a new one), Smart/RESTART calls its checkpoint service to commit working storage contents, sequential file position and resource manager activity. This checkpoint completes *before* the application regains control from its READ request. This requires no special attention except during restart.

On restart, the first READ the application issues should retrieve the record that triggered the checkpoint (the control break record). This ensures the processing associated with the control break record is re-executed since the processing associated with this record occurred *after* the checkpoint in the previous, failed run. Section 44.6 describes and illustrates a jobstream that uses the REPO_STYLE command to properly position the file for restart. Chapter 2 of this manual and Chapter 12 of the Smart/RESTART Reference Manual provide further details on the syntax and use of the REPO_STYLE directive.

44.3 Unit-of-Work Definition

44.3.1 Units-of-Work defined by record count

A logical unit-of-work may be defined as the number of records read and processed from an input (driver) file. Smart/RESTART can automatically issue a checkpoint whenever a pre-specified number of driver file records are read. The COMMIT_FILE parameter identifies the driver file by its DDNAME. The COMMIT_FILE command also specifies the number of records to be read before Smart/RESTART *implicitly* issues a checkpoint.

44.3.2 Units-of-Work defined by record content

The COMMIT_FILE command lets you define a unit-of-work in terms of up to six **control break** fields within the driver file record. A change in any one of these fields causes Smart/RESTART to issue a physical checkpoint.

The following example directs Smart/RESTART to issue a checkpoint whenever the account field changes in the driver file named **TRANFILE**. The 15 byte account field is located at position 25 in the input record.

```
COMMIT_FILE(CONTENT,TRANFILE,(25,15))
```

44.4 Application Feedback

Although not generally required for successful processing, the application may want to know when Smart/RESTART triggers a checkpoint based on an external commit definition. Smart/RESTART provides this feedback as follows:

When a READ triggers a successful checkpoint, Smart/RESTART sets the field labeled FCA-STATUS to the value 'EC00'. Should the checkpoint fail, the FCA-STATUS field is set to 'EC08'.

44.5 External Checkpointing Example

The following example illustrates the use of Smart/RESTART's external commit facility using the COMMIT_FILE parameter. An input transaction file (the Driver file) is sorted by account number. The main processing loop copies the input record to an output file while maintaining a running total of payments by account. The account payments total is displayed whenever the account number changes. The application deliberately abends when it reads record number 38. A subsequent restart will resume processing from the last checkpoint.

```
ID DIVISION.
PROGRAM-ID.      EXAMPLE.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT TRAN-FILE ASSIGN TO TRANFILE
        FILE STATUS IS TRAN-FILE-STATUS.
    SELECT OUTPUT-FILE ASSIGN TO OUTFILE
        FILE STATUS IS OUTPUT-FILE-STATUS.

DATA DIVISION.
FILE SECTION.
FD  TRAN-FILE
    LABEL RECORDS STANDARD
    RECORDING MODE F
    BLOCK CONTAINS 0 RECORDS.
01  TRAN-RECORD.
    05  REC-ID          PIC  X(5).
    05  FILLER          PIC  X(5).
    05  ACCOUNT         PIC  X(8).
    05  FILLER          PIC  X(2).
    05  TRAN-PAYMENT    PIC  X(7).
    05  FILLER          PIC  X(44).

FD  OUTPUT-FILE
    LABEL RECORDS STANDARD
    RECORDING MODE F
    BLOCK CONTAINS 0 RECORDS.
01  OUTPUT-RECORD      PIC  X(80).

WORKING-STORAGE SECTION.

01  VITAL-STORAGE.
    05  UOW-COUNT              PIC  S9(4) COMP VALUE 1.
    05  INPUT-COUNT            PIC  S9(4) COMP VALUE 0.
    05  CHARACTER-FIELD        PIC  X(04) VALUE SPACES.
    05  NUMERIC-FIELD          REDEFINES
        CHARACTER-FIELD        PIC  S9(4).
    05  ERROR-FLAG             PIC  X(1) VALUE SPACE.
        88  API-ERROR          VALUE 'A'.
        88  PROCESSING-ERROR   VALUES ARE 'A' 'C' 'I'.
    05  EOF-FLAG               PIC  X(1) VALUE SPACE.
        88  END-OF-FILE        VALUE 'E'.
    05  RUN-FLAG               PIC  X(1) VALUE SPACE.
        88  NORMAL-RUN         VALUE SPACE.
```

```

      88  RESTART-RUN          VALUE 'R'.
05  PGM-RETURN-CODE          PIC S9(4) COMP VALUE 0.
05  PAYMENTS-PER-ACCOUNT     PIC 9(7) VALUE 0.

PROCEDURE DIVISION.

    PERFORM A100-INITIALIZE THRU A100-EXIT.
    PERFORM B100-UNIT-OF-WORK THRU B100-EXIT
        UNTIL END-OF-FILE
        OR    PROCESSING-ERROR.

    DISPLAY 'EXAMPLE - PROCESSING COMPLETED'.
    GOBACK.

A100-INITIALIZE.
    IF RETURN-CODE = +2001 THEN
        DISPLAY 'EXAMPLE - DETECTED THE RESTART ENVIRONMENT'
        MOVE 'R' TO RUN-FLAG
    ELSE
        DISPLAY 'EXAMPLE IS EXECUTING AN INITIAL RUN'

    OPEN INPUT TRAN-FILE.
    IF TRAN-FILE-STATUS > 0 THEN
        DISPLAY 'INPUT ERROR'
        MOVE 'A' TO ERROR-FLAG.

    OPEN OUTPUT OUTPUT-FILE.
    IF OUTPUT-FILE-STATUS > 0 THEN
        DISPLAY 'OUTPUT ERROR'
        MOVE 'A' TO ERROR-FLAG.
A100-EXIT.
    EXIT.

B100-UNIT-OF-WORK.
    READ TRAN-FILE
        AT END MOVE 'E' TO EOF-FLAG
        GO TO B100-EXIT.
    IF FCA-EXT-COMMIT-OK THEN
        ADD 1 TO UOW-COUNT
        DISPLAY 'TOTAL PAYMENTS = ' PAYMENTS-PER-ACCOUNT
    ELSE
        IF FCA-EXT-COMMIT-ERR THEN
            DISPLAY 'CHECKPOINT FAILURE OCCURRED'
            MOVE 'C' TO PROCESSING-ERROR
        END-IF
    END-IF.

    ADD 1 TO INPUT-COUNT.
    MOVE TRAN-RECORD TO OUTPUT-RECORD.
    WRITE OUTPUT-RECORD.

    ADD TRAN-PAYMENT TO PAYMENTS-PER-ACCOUNT.

*   DELIBERATELY ABEND THE ORIGINAL RUN OF PROGRAM AT RECORD 38

    IF NORMAL-RUN AND INPUT-COUNT = 38
        DISPLAY 'EXAMPLE - DELIBERATE SOC7 ABEND AFTER READING 38
RECORDS'
        ADD 1 TO NUMERIC-FIELD.
B100-EXIT.
    EXIT.

```

Figure 44.1 *An External Commit facility Example*

44.5.1 Driver file contents

The transaction file (input driver file) contains the following 70 records.

00001	02040656	0003400	PART OF UOW 1	
00002	02040656	0003500	PART OF UOW 1	
00003	02040656	0010770	PART OF UOW 1	
00004	02040656	0003450	PART OF UOW 1	
00005	02040656	0068400	PART OF UOW 1	
00006	02040656	0000500	PART OF UOW 1	
00007	02040656	0003700	PART OF UOW 1	
00008	02040656	0074800	PART OF UOW 1	
00009	02040656	1006700	PART OF UOW 1	
00010	02040656	0789400	PART OF UOW 1	
00011	02041777	0067540	PART OF UOW 2	
00012	02041777	0054760	PART OF UOW 2	
00013	02041777	0074400	PART OF UOW 2	
00014	02041777	0844400	PART OF UOW 2	
00015	02041777	0034600	PART OF UOW 2	
00016	02041777	0863400	PART OF UOW 2	
00017	02041777	0443400	PART OF UOW 2	
00018	02041777	0077400	PART OF UOW 2	
00019	02041777	0024300	PART OF UOW 2	
00020	02041777	0045600	PART OF UOW 2	
00021	02051777	0067540	PART OF UOW 3	
00022	02051777	0054760	PART OF UOW 3	
00023	02051777	0074400	PART OF UOW 3	
00024	02051777	0844400	PART OF UOW 3	
00025	02051777	0034600	PART OF UOW 3	
00026	02051777	0863400	PART OF UOW 3	
00027	02051777	0443400	PART OF UOW 3	
00028	02051777	0077400	PART OF UOW 3	
00029	02051777	0024300	PART OF UOW 3	
00030	02051777	0045600	PART OF UOW 3	
00031	03005041	0067540	PART OF UOW 4	
00032	03005041	0054760	PART OF UOW 4	
00033	03005041	0074400	PART OF UOW 4	
00034	03005041	0844400	PART OF UOW 4	
00035	03005041	0034600	PART OF UOW 4	
00036	03005041	0863400	PART OF UOW 4	
00037	03005041	0443400	PART OF UOW 4	
00038	BAD data	xxxxxxx	PART OF UOW 4	S0C7 occurs on this record
00039	03005041	0024300	PART OF UOW 4	
00040	03005041	0045600	PART OF UOW 4	
00041	03550000	0067540	PART OF UOW 5	
00044	03550000	0054760	PART OF UOW 5	
00043	03550000	0074400	PART OF UOW 5	
00044	03550000	0844400	PART OF UOW 5	
00045	03550000	0034600	PART OF UOW 5	
00046	03550000	0863400	PART OF UOW 5	
00047	03550000	0443400	PART OF UOW 5	
00048	03550000	0077400	PART OF UOW 5	
00049	03550000	0024300	PART OF UOW 5	
00050	03550000	0045600	PART OF UOW 5	
00051	04089695	0067540	PART OF UOW 6	
00052	04089695	0054760	PART OF UOW 6	
00044	04089695	0074400	PART OF UOW 6	
00054	04089695	0844400	PART OF UOW 6	
00055	04089695	0034600	PART OF UOW 6	

00056	04089695	0863400	PART OF UOW 6
00057	04089695	0443400	PART OF UOW 6
00058	04089695	0077400	PART OF UOW 6
00059	04089695	0024300	PART OF UOW 6
00060	04089695	0045600	PART OF UOW 6
00061	04986786	0067540	PART OF UOW 7
00062	04986786	0054760	PART OF UOW 7
00063	04986786	0074400	PART OF UOW 7
00064	04986786	0844400	PART OF UOW 7
00065	04986786	0034600	PART OF UOW 7
00066	04986786	0863400	PART OF UOW 7
00067	04986786	0443400	PART OF UOW 7
00068	04986786	0077400	PART OF UOW 7
00069	04986786	0024300	PART OF UOW 7
00070	04986786	0045600	PART OF UOW 7

44.5.2 JCL for the Initial and Restart Runs

```

//STEP1    EXEC PGM=EXAMPLE
//SRSCHECK DD DSN=DCAvrM.CHECKPT,DISP=SHR
//TRANFILE DD DSN=input.driver.file,DISP=OLD          <==== (1)
//OUTFILE  DD DSN=output.file,DISP=OLD
//SYSOUT   DD SYSOUT=*
//SRSPRINT DD SYSOUT=*
//RAINPUT  DD *
            COMMIT_FILE(CONTENT,TRANFILE,(11,8))      <==== (2)
            REPO_STYLE(AT,ONLY,(TRANFILE))            <==== (3)
//

```

where

- (1) The TRANFILE DD statement defines the input 'driver' file
- (2) Smart/RESTART should trigger a checkpoint when the contents of the account field changes. The account field starts at position 11 within the TRANFILE input record for a length of 8.
- (3) In the event of restart, reposition the input driver file to the record which triggered the checkpoint. On restart, allow the application to re-read and fully process this 'control break' record.

Figure 44.1 *External COMMIT Example*

44.6 The Output Driver File processing model

The Output Driver File processing model is very similar to the Input Driver File processing model discussed in paragraphs 44.2 through 44.5 with the following notable exceptions:

- The contents or counts of records associated with a single output file govern when a checkpoint should occur. This output file is known as the *driver* file.
- A central Write/Process loop must exist. Other input and output files may be accessed which may also be repositionable. However, they do not define the boundaries of a unit-of-work.
- As each application WRITE is issued, Smart/RESTART uses the Unit-of-Work definition (specified through the COMMIT_FILE directive) to determine whether the record just written constitutes a *new* logical unit-of-work. When Smart/RESTART detects a ‘control break’ (the completion of the previous unit-of-work and the commencement of a new one), Smart/RESTART calls its checkpoint service to commit working storage contents, sequential file position and resource manager activity. This checkpoint completes *before* the application regains control from its WRITE request.
- The REPO_STYLE command applies to input files only. Therefore, do not specify the REPO_STYLE command when the driver file is an output file.
- A logical unit-of-work may be defined as the number of records written to an output (driver) file. Smart/RESTART can automatically issue a checkpoint whenever a pre-specified number of driver file records are written.
- JCL for the Initial and Restart runs are the same as shown in the Figure 44.1 above, and vary only by the RAINPUT contents:

```
. . .  
//RAINPUT DD *  
    COMMIT_FILE(CONTENT,OUTFILE,(11,8))  
//
```

where the OUTFILE DD statement defines the output ‘driver’ file.

Advanced Smart/RESTART Topics

This chapter addresses several advanced Smart/RESTART topics, each in its own major section. For example, Section 50.1 describes how to identify, multiple discontinuous storage areas to Smart/RESTART while Section 50.3 describes procedures necessary to conduct Point-in-Time Recovery.

50.1 Identifying Multiple Data Areas to Smart/RESTART

Smart/RESTART provides several means to identify the application storage that should be saved at checkpoint time and restored at either restart or ROLLBACK time. Multiple, discontinuous areas of storage can be specified using the techniques illustrated in the examples which follow.

The subsections which follow describe and illustrate how to define multiple, discontinuous areas of storage that Smart/RESTART should save at checkpoint time and restore at restart or rollback time.

50.1.1 Explicit CALLs to the INITIALIZE Service

The Smart/RESTART call level API lets you identify the storage you wish Smart/RESTART to save at checkpoint time as ***pairs of parameters*** in either of two formats. In the first format, each pair of parameters is specified as a storage address followed by a full word binary field containing the length of the field or structure you wish Smart/RESTART to save. The COBOL source fragments in Figures 50.1 and 50.2 illustrate this first technique:

***NOTE:** Chapter 11 of this publication describes the Call level API in detail.*

```

01  VITAL-STORAGE-1.
    05  VITAL-COUNTER-1      PIC S9(5) COMP.
    05  VITAL-FLAGS-1.
        10  VITAL-FLAG-1A    PIC X(1).
        10  VITAL-FLAG-1B    PIC X(1).

77  NON-VITAL-STORAGE-1      PIC X(5).

01  VITAL-STORAGE-2.
    05  VITAL-COUNTER-2      PIC S9(5) COMP.
    05  VITAL-FLAGS-2.
        10  VITAL-FLAG-2A    PIC X(1).
        10  VITAL-FLAG-2B    PIC X(1).

01  NON-VITAL-STORAGE-2.
    05  FIELD1                PIC S9(5) COMP.
    05  FIELD2                PIC S9(5) COMP.

01  VITAL-STORAGE-3.
    05  VITAL-COUNTER-3      PIC S9(5) COMP.
    05  VITAL-FLAGS-3.
        10  VITAL-FLAG-3A    PIC X(1).
        10  VITAL-FLAG-3B    PIC X(1).

01  VITAL-STORAGE-LENGTHS.
    05  VITAL-STORAGE-1-LENGTH PIC S9(5) COMP.      <=== (1)
    05  VITAL-STORAGE-2-LENGTH PIC S9(5) COMP.      <=== (1)
    05  VITAL-STORAGE-3-LENGTH PIC S9(5) COMP.      <=== (1)

MOVE 'L' TO FCA-INIT-STYLE.      <=== (2)
MOVE LENGTH OF VITAL-STORAGE-1 TO VITAL-STORAGE-1-LENGTH. <=== (3)
MOVE LENGTH OF VITAL-STORAGE-2 TO VITAL-STORAGE-2-LENGTH. <=== (3)
MOVE LENGTH OF VITAL-STORAGE-3 TO VITAL-STORAGE-3-LENGTH. <=== (3)
CALL 'RAIAPI' USING RAI-INITIALIZE,
    FCA,
    VITAL-STORAGE-1
    VITAL-STORAGE-1-LENGTH
    VITAL-STORAGE-2
    VITAL-STORAGE-2-LENGTH
    VITAL-STORAGE-3
    VITAL-STORAGE-3-LENGTH.

```

where

- (1) The length field is defined as a signed fullword binary numeric data item whose COBOL picture clause is S9(5) COMP.
- (2) The FCA-INIT-STYLE variable governs whether Smart/RESTART interprets each parameter pair as a storage address and length or as a starting-address and ending address. The default supplied by RAI is address/length. Here, we direct Smart/RESTART to treat the parameter list as a series of address / length pairs.
- (3) Set the lengths of the structures whose contents are to be saved at checkpoint time and restored at restart or rollback time.

Figure 50.1 *INITIALIZE call using Address / Length Technique*

Alternatively, you can make use of the COBOL 'LENGTH OF' special register on your calls to the INITIALIZE service as illustrated in the coding fragment in Figure 50.2. *This makes it unnecessary to define a length field or code a MOVE statement to set its value.*

```

01 VITAL-STORAGE-1.
   05 VITAL-COUNTER-1      PIC S9(5) COMP.
   05 VITAL-FLAGS-1.
      10 VITAL-FLAG-1A     PIC X(1).
      10 VITAL-FLAG-1B     PIC X(1).

77 NON-VITAL-STORAGE-1     PIC X(5).

01 VITAL-STORAGE-2.
   05 VITAL-COUNTER-2      PIC S9(5) COMP.
   05 VITAL-FLAGS-2.
      10 VITAL-FLAG-2A     PIC X(1).
      10 VITAL-FLAG-2B     PIC X(1).

01 NON-VITAL-STORAGE-2.
   05 FIELD1               PIC S9(5) COMP.
   05 FIELD2               PIC S9(5) COMP.

01 VITAL-STORAGE-3.
   05 VITAL-COUNTER-3      PIC S9(5) COMP.
   05 VITAL-FLAGS-3.
      10 VITAL-FLAG-3A     PIC X(1).
      10 VITAL-FLAG-3B     PIC X(1).

MOVE 'L' TO FCA-INIT-STYLE.                                     <=== (1)
CALL 'RAIAPI' USING RAI-INITIALIZE,
    FCA,
    VITAL-STORAGE-1
    BY CONTENT LENGTH OF VITAL-STORAGE-1                       <=== (2)
    VITAL-STORAGE-2
    BY CONTENT LENGTH OF VITAL-STORAGE-2                       <=== (2)
    VITAL-STORAGE-3
    BY CONTENT LENGTH OF VITAL-STORAGE-3.                       <=== (2)

```

where

- (1) Direct Smart/RESTART to treat the parameter list as a series of address / length pairs.
- (2) The use of the BY CONTENT clause on the CALL statement is *mandatory* because the default (BY ADDRESS) produces erroneous results.

Figure 50.2 INITIALIZE call using LENGTH OF special register

At run time, Smart/RESTART identifies the discrete storage areas it will save at checkpoint time (and restore at rollback or restart time).

```

SRS438I - Application specified a RAM area length of 00104 bytes.
SRS438I - Application specified a RAM area length of 00016 bytes.
SRS438I - Application specified a RAM area length of 00076 bytes.

```

50.1.2 Explicit CALLs to the INITIALIZE Service – Format II

In the second format, Smart/RESTART interprets the pair of parameters specified on the INITIALIZE call as a starting address and ending address. In this case, Smart/RESTART saves the entire range of storage between (and including) the starting and ending addresses. The COBOL fragment in Figure 50.3 illustrates how to save three discrete storage ranges.

```
01 VITAL-STORAGE-1.
   05 VITAL-COUNTER-1      PIC S9(5) COMP.
   05 VITAL-FLAGS-1.
       10 VITAL-FLAG-1A    PIC X(1).
       10 VITAL-FLAG-1B    PIC X(1).
77 VITAL-STORAGE-1-END     PIC X(1).

01 NON-VITAL-STORAGE-1.
   05 FIELD1               PIC S9(5) COMP.
   05 FIELD2               PIC S9(5) COMP.

01 VITAL-STORAGE-2.
   05 VITAL-COUNTER-2      PIC S9(5) COMP.
   05 VITAL-FLAGS-2.
       10 VITAL-FLAG-2A    PIC X(1).
       10 VITAL-FLAG-2B    PIC X(1).
77 VITAL-STORAGE-2-END     PIC X(1).

01 NON-VITAL-STORAGE-2.
   05 FIELD1               PIC S9(5) COMP.
   05 FIELD2               PIC S9(5) COMP.

01 VITAL-STORAGE-3.
   05 VITAL-COUNTER-3      PIC S9(5) COMP.
   05 VITAL-FLAGS-3.
       10 VITAL-FLAG-3A    PIC X(1).
       10 VITAL-FLAG-3B    PIC X(1).
77 VITAL-STORAGE-3-END     PIC X(1).

MOVE 'E' TO FCA-INIT-STYLE.
CALL 'RAIAPI' USING RAI-INITIALIZE,
                   FCA,
                   VITAL-STORAGE-1
                   VITAL-STORAGE-1-END
                   VITAL-STORAGE-2
                   VITAL-STORAGE-2-END
                   VITAL-STORAGE-3
                   VITAL-STORAGE-3-END.
```

<=== (1)

where

- (1) Direct Smart/RESTART to treat the parameter list as a series of starting and ending address pairs.

Figure 50.3 INITIALIZE call using Begin Address / End Address Technique

50.1.3 EXEC SRS SAVE statements

EXEC SRS is a high level interface to Smart/RESTART services that is analogous to EXEC SQL and EXEC CICS. Chapter 10 of this publication describes Smart/RESTART's EXEC level interface in detail.

The EXEC SRS BEGIN SAVE and EXEC SRS END SAVE statements are directives to Smart/PRECOMPILER. They let you identify the application storage areas that Smart/RESTART should save at checkpoint time and restore at restart or ROLLBACK time. The COBOL fragment in Figure 50.4 illustrates how to save three discrete storage ranges.

```
EXEC SRS BEGIN SAVE END-EXEC                <=== (1)
01  VITAL-STORAGE-1.
    05  VITAL-COUNTER-1          PIC S9(5) COMP.
    05  VITAL-FLAGS-1.
        10 VITAL-FLAG-1A        PIC X(1).
        10 VITAL-FLAG-1B        PIC X(1).

01  VITAL-STORAGE-2.
    05  VITAL-COUNTER-2          PIC S9(5) COMP.
    05  VITAL-FLAGS-2.
        10 VITAL-FLAG-2A        PIC X(1).
        10 VITAL-FLAG-2B        PIC X(1).
77  VITAL-STORAGE-77            PIC X(5).
EXEC SRS END SAVE END-EXEC                <=== (2)

01  NON-VITAL-STORAGE-1.
    05  FIELD1                   PIC S9(5) COMP.
    05  FIELD2                   PIC S9(5) COMP.

EXEC SRS BEGIN SAVE END-EXEC                <=== (1)
01  VITAL-STORAGE-3.
    05  VITAL-COUNTER-3          PIC S9(5) COMP.
    05  VITAL-FLAGS-3.
        10 VITAL-FLAG-3A        PIC X(1).
        10 VITAL-FLAG-3B        PIC X(1).
EXEC SRS END SAVE END-EXEC                <=== (2)

01  NON-VITAL-STORAGE-2.
    05  FIELD1                   PIC S9(5) COMP.
    05  FIELD2                   PIC S9(5) COMP.

EXEC SRS BEGIN SAVE END-EXEC                <=== (1)
01  VITAL-STORAGE-4.
    05  VITAL-COUNTER-4          PIC S9(5) COMP.
    05  VITAL-FLAGS-4.
        10 VITAL-FLAG-4A        PIC X(1).
        10 VITAL-FLAG-4B        PIC X(1).
EXEC SRS END SAVE END-EXEC                <=== (2)
```

where

- (1) The EXEC SRS BEGIN SAVE statement denotes the beginning of an area of storage that should be saved at checkpoint time and restored at restart or rollback time. The scope of the first pair of EXEC SRS BEGIN SAVE and END SAVE statements span several 01 and 77 level data structures.
- (2) The EXEC SRS END SAVE statement delimits the end of the storage range to be saved and restored.

Figure 50.4 *Identifying storage ranges to save and restore*

50.1.4 Smart/PRECOMPILER STORAGE and STORAGE-RANGE parameters

As an alternative to EXEC SRS BEGIN SAVE and EXEC SRS END SAVE requests, Smart/PRECOMPILER's STORAGE and STORAGE-RANGE parameters let you identify the names of data structures to be saved and restored -- *without the need to modify your source program*. For example, the following STORAGE specification directs Smart/PRECOMPILER to register three data structures with Smart/RESTART:

```
STORAGE(VITAL-STORAGE-1, VITAL-STORAGE-2,VITAL-STORAGE-3)
```

In addition, you can define the names of one or more data structures as *defaults* when Smart/PRECOMPILER is installed. When Smart/PRECOMPILER encounters any of these data structures, it *automatically* generates code to save and restore the structure(s) and all the fields which comprise it.

```
Smart/PRECOMPILER Defaults (#1 of 2)
Command ==>
SJS031 - Specify Profile defaults and press ENTER or press END/RETURN to exit
Restartable Application Defaults
Resource Manager . . . DB2      (for example: DB2, IMS, DB2/IMS; MQS)
DB2 Attachment . . . . 1      (1 - Smart/CAF, 2-TSO, 3 - IMS, 4 - Other)
Files managed . . . . . 2      (1 - All, 2 - None)
Host Language . . . . . 2      (1 - OS/VS COBOL, 2-other COBOLs, 3-PL/1)
Source Margins . . . . . 8 72  (for host language statements)
INITIALIZE Style . . . . 1      (1-Address/Length, 2-Start Addr/End Addr)

Names of work areas to be saved at checkpoint time and restored at restart time
Vital Work Area 1 . . . VITAL-STORAGE-1
Vital Work Area 2 . . . VITAL-STORAGE-2
Vital Work Area 3 . . . VITAL-STORAGE-3
Vital Work Area 4 . . .
Vital Work Area 5 . . .
```

Figure 50.5 *Smart/PRECOMPILER Installation Defaults*

50.2 Defining EXTERNAL Storage to Smart/RESTART

Storage defined with the COBOL EXTERNAL attribute can be referenced by any module within a run unit. When different programs within a run unit refer to EXTERNAL data items, they in fact reference the *same* storage. External storage is actually discontinuous to other data specified in a COBOL module's Working Storage Section.

Any EXTERNAL storage you want Smart/RESTART to save at checkpoint time and restore at restart time must be identified via an explicit call to the Smart/RESTART INITIALIZE service. Chapter 11.3 describes the Call Level Interface to the INITIALIZE service in detail. The COBOL source fragment in Figure 50.6 illustrates how to define external storage to Smart/RESTART.

```
01  EXTERNAL-STORAGE  EXTERNAL.                                <=== (1)
    05  EXTERNAL-COUNTER      PIC S9(5) COMP.
    05  EXTERNAL-SUBSCRIPT    PIC S9(5) COMP.
    05  EXTERNAL-PAGE-NUMBER  PIC S9(5) COMP.
    05  EXTERNAL-POINTER     PIC S9(5) COMP.
    05  EXTERNAL-FLAGS.
        10  EXTERNAL-FLAG1    PIC X(1).
        10  EXTERNAL-FLAG2    PIC X(1).
        10  EXTERNAL-FLAG3    PIC X(1).

01  EXTERNAL-STORAGE-LENGTH  PIC S9(5) COMP.                    <=== (2)

MOVE LENGTH OF EXTERNAL-STORAGE TO EXTERNAL-STORAGE-LENGTH.    <=== (3)
CALL 'RAIAPI' USING RAI-INITIALIZE,                               <=== (4)
    FCA,
    EXTERNAL-STORAGE,
    EXTERNAL-STORAGE-LENGTH.
```

where

- (1) The structure named EXTERNAL-STORAGE is defined with the EXTERNAL attribute.
- (2) The length field is defined as a signed fullword binary numeric data item whose COBOL picture clause is S9(5) COMP.
- (3) Set the length of the structure named EXTERNAL-STORAGE.
- (4) Specify the address and length of the storage defined with the EXTERNAL attribute on the call to the INITIALIZE service.

Figure 50.6 INITIALIZE call to define EXTERNAL storage

50.3 Point-in-Time Recovery with Smart/RESTART

By default, Smart/RESTART maintains two checkpoints. A typical restart occurs from the last successful checkpoint. By maintaining two checkpoints, Smart/RESTART can ensure there is always an intact checkpoint (that represents a point of consistency with respect to working storage, sequential file position and resource manager activity) from which to restart. Moreover, restart from the **LAST** checkpoint requires no manipulation of protected resources controlled by RRS compliant resource managers such as DB2, WebSphere MQ or IMS.

In addition to supporting application restart from the **last** successfully completed unit-of-work, Smart/RESTART lets you restart applications from **previous** checkpoints as well. Restart from the last checkpoint is the default mode of operation and the simplest from an operational point of view because in this case protected resources, working storage contents and sequential file position are *automatically* in synch.

In contrast, application restart from a previous checkpoint is more complicated. Before restart can be attempted, protected resources controlled by RRS compliant resource managers such as DB2, WebSphere MQ and IMS must first be **manually** restored to their state at the point in time from which restart should occur. The method used depends on the resource manager. With DB2, for example, *point in time recovery* must be performed via a utility program like DSN1LOGP. Once protected resources have been reset, Smart/RESTART can ensure that working storage and sequential file position are properly restored to the state which corresponds to the requested checkpoint.

Two Smart/RESTART parameters support point-in-time recovery: CKPT_DEPTH and RESTART_ID.

- The CKPT_DEPTH parameter specifies the number of discrete checkpoints that Smart/RESTART maintains on a rotational *LIFO* basis. When the number of physical checkpoints processed exceeds the CKPT_DEPTH, Smart/RESTART overlays the oldest checkpoint area.
- The RESTART_ID parameter identifies the checkpoint from which restart should occur. The RESTART_ID value specifies the numeric checkpoint identifier that appears in Smart/RESTART message SRS031I. Each of these checkpoint messages is date and time stamped to facilitate point-in-time recovery.

50.4 SRSOFF Dual Environment Facility

The SRSOFF facility lets you run object code which calls Smart/RESTART services in an environment in which Smart/RESTART is not present. Calls to Smart/RESTART may have been introduced automatically by Smart/PRECOMPILER. Alternatively, they may have been coded explicitly as EXEC SRS statements or through the Call Level Interface.

The SRSOFF facility lets you effectively disable Smart/RESTART. Thus, you can use the same source code and executable load modules both with and without Smart/RESTART.

The SRSOFF facility lets you toggle Smart/RESTART on and off. For example, to **deactivate** Smart/RESTART, include the following DD statement in your execution JCL.

```
//SRSOFF DD DUMMY
```

Smart/RESTART issues the following messages when it detects an SRSOFF DD statement:

```
SRSB43W SRSOFF DD FOUND - RAI-API BYPASSED
```

To **activate** Smart/RESTART, simply remove the SRSOFF DD statement.

50.5 Risk of Inconsistent Data with Language Environment (LE)

Smart/RESTART and Smart/RRSAF applications that run in Language Environment for z/OS and OS/390 can, under certain circumstances, create inconsistent data. A combination of Language Environment (LE) and Smart/RESTART options can cause these applications to commit changes that should be rolled back.

Applications running in Language Environment should abend when an unhandled condition of LE severity 2 or greater (such as a data exception or User abend) occurs. When an application program abends, both Smart/RESTART and RRS compliant resource managers such as DB2, WebSphere MQ and IMS roll back changes made since the last checkpoint. Unfortunately, the IBM supplied default for batch LE is to **trap** an abend and terminate the enclave (your application program) with a return and reason code. This default behavior can cause updates made by a failing application to be *committed* rather than *backed out*.

Four things have to **all** happen for the exposure to exist:

1. The Smart/RESTART ESTAE option, which defaults to ON, must be specified as OFF
2. The Smart/RESTART ESPIE option, which defaults to ON, must be specified as OFF
3. The Smart/RESTART ERROR_ACTION option, which defaults to ABEND, must be specified as IGNORE
4. The LE for z/OS and OS/390 option ABTERMENC must be set to RETCODE.

Fortunately, the default values for the Smart/RESTART options above are set when the product is installed, and typically do not change thereafter. The Smart/RESTART Reference manual specifically states that the ESTAE and ESPIE options should *only* be used at the direction of Relational Architects technical support.

The LE option ABTERMENC(RETCODE) directs LE to intercept an abend and terminate the enclave with a return code and reason code. This option can be set on a jobstep basis.

If **all** the above four conditions are true **and** LE encounters an unhandled condition of severity 2 or greater, then a failing application will have updates from in-flight units-of-work committed rather than rolled back. Committing changes (by both Smart/RESTART and RRS compliant resource managers) from an incomplete unit-of-work will most likely result in inconsistent data.

If a job step is restarted immediately and runs to normal completion, the inconsistent data will be visible only between the failed initial run and the restart run. However, these circumstances can cause Smart/RESTART to delete its internal checkpoint file making restart impossible.

50.6 Spanned Record Support

Smart/RESTART supports the use of spanned records (RECFM=VBS) with sequential input and output files. Spanned records occupy more than one block. That is, the logical record size is larger than the physical block size. The operating system (z/OS or OS/390) requires that sequential files with spanned records must have a variable blocked record format. Figure 50.7 shows the COBOL FD section for an input file with spanned records.

```
FD  INPUT-FILE
    RECORDING MODE S
    BLOCK CONTAINS 0 RECORDS
    RECORD IS VARYING IN SIZE FROM 80 TO 22111
    DEPENDING ON WS-LRECL.
01  INPUT-RECORD-80    PIC X(80).
01  INPUT-RECORD-950   PIC X(950).
01  INPUT-RECORD-22111 PIC X(22111).
...
77 WS-LRECL  PIC 9(6) VALUE 950.
```

Figure 50.7 *COBOL FD for an input file with spanned records*

50.7 The RMI parameter

During the execution of restartable applications, Smart/RESTART internally keeps two types of data -- checkpoint data and synchronization (sync) data. Checkpoint data is application storage (e.g. Working Storage in COBOL) and sequential file position and the BULK_CKPT parameter specifies its storage medium. Checkpoint data is the bulk of the data maintained by Smart/RESTART during execution. The remaining internal information required by Smart/RESTART is called synchronizing data.

Synchronizing data is information about the current state of a restartable application. It is kept in synch with changes the restartable application makes to RRS compliant resources such as those managed by DB2, WebSphere MQ and/or IMS.

Sync data is stored in either a DB2 table (typically) or an MQ queue. The IBM resource manager(s) used in an application program determine the location of the synchronizing data. The structure of the DB2 Synch table is documented in Appendix C of this publication.

The Smart/RESTART runtime parameter RMI (Resource Manager Interface) governs options for accessing the sync data. It specifies the name of a Smart/RESTART routine that interfaces between Smart/RESTART and the synchronizing data.

Values for the RMI parameter are load module names. Different load modules (interface routines) are needed by Smart/RESTART based on the resource manager(s) accessed by the restartable application. The Smart/RESTART interface routine implements a two-

phase commit protocol by which it synchronizes application processing and resource manager activity.

When Smart/RESTART is installed, a system-wide default is specified for the RMI parameter. When a job step requires a different option, the RMI parameter can be specified in RAINPUT.

50.7.1 Sync Data in DB2

DB2 based application programs typically make use of synchronizing data stored in a DB2 table. The Smart/RESTART program that accesses synchronizing data in DB2 is named SRSDBRM. This program contains all the SQL statements needed by Smart/RESTART for reading and writing synchronizing data. It also contains all the SQL statements used by Smart/RESTART when the bulk checkpoint medium is DB2. It is the *only* Smart/RESTART program containing any SQL statements.

Smart/RESTART supports multiple DB2 attachments. It does so by link-editing program SRSDBRM into multiple load modules, each load module supporting the use of a different DB2 attachment.

Programs using the TSO attachment to DB2 are required to include module DSNELI in their link-edit step. Load module SRSDBRMT is used by Smart/RESTART in conjunction with the TSO attachment. It includes CSECTs SRSDBRM and DSNELI. Programs using the Smart/RRSAF DB2 attachment from Relational Architects are required to link-edit module DCAHLI. Load module SRSDBRM is used by Smart/RESTART in conjunction with Smart/RRSAF DB2 access. It includes programs SRSDBRM and DCAHLI.

Other DB2 attachments are supported by load module SRSDBRMO. JCL to create this load module is provided in member SRSJDBRM of the DCACNTL library.

In addition to DB2 based applications, restartable IMS applications also maintain synchronizing data in DB2. These applications must include program DFSLI000 in their link-edit. Load module SRSDBRMI is used by Smart/RESTART in conjunction with the IMS attachment to DB2. It includes CSECTs SRSDBRM and DFSLI000.

A summary of the various Resource Manager Interface load modules which store synchronizing data in DB2 are shown below.

Load Module	Linked Stub	DB2 Attachment	Synchpoint Co-ordinator
SRSDBRMT	DSNELI	TSO	DB2
SRSDBRM	DCAHLI	Smart/RRSAF	DB2
SRSDBRMI	DFSLI000	IMS	IMS
SRSDBRMO	any	Any Other	DB2
SRSRDB2	DCAHLI	Smart/RRSAF	RRS

NOTE: Any batch application accessing DB2 needs an application plan that includes the package or DBRM for SRSDBRM -- in addition to a RMI load module. Section 6 of Chapter 3 describes how to BIND the SRSDBRM package or DBRM.

Smart/RESTART also supports restartable applications that are limited to VSAM and sequential file access. Typically, these applications have no synchronizing data as there are no IBM resource managers with which to synchronize. However, synchronizing data is required in order to use dynamically allocated checkpoint files. See Chapter 4.1.2.1 on the dynamic allocation of checkpoint files for more information.

50.7.2 DB2, WebSphere MQ and RRS

Program SRSDBRM issues SQL COMMITs and ROLLBACKs and is sufficient in a DB2 only environment. In a mixed DB2 and IMS environment, the IMS attachment co-ordinates IMS and DB2 changes. In an environment in which a mix of RRS compliant resources is accessed, synchpoint coordination is managed by the Resource Recovery Services (RRS) component of z/OS and OS/390.

RRS is required for co-ordinated checkpoint processing when:

- An application program accesses a mix of RRS compliant resources such as DB2, WebSphere MQ and IMS
- An application program accesses only MQ resources but synchronizing data is kept in DB2

RRS checkpoints are requested via SRRCMIT while rollbacks are requested with SRRBACK. These system services front-end and co-ordinate the checkpoint and rollback facilities of RRS compliant resource managers such as DB2 and WebSphere MQ. The Smart/RESTART program named SRSRDB2 issues SRRCMIT and SRRBACK requests. For all other DB2 access to synchronizing data by Smart/RESTART, program SRSRDB2 calls program SRSDBRM.

Load module SRSRDB2 should be specified in the RMI parameter for a mix of RRS compliant resources when synchronizing data is maintained in DB2. It includes programs SRSRDB2, SRSDBRM and DCAHLI. When using RRS, Smart/RESTART only supports Smart/RRSAF as the DB2 attachment.

<u>Load Module</u>	<u>Linked Stub</u>	<u>DB2 Attach</u>	<u>Synchpont Coordinator</u>
SRSRDB2	DCAHLI	Smart/RRSAF	RRS

Applications accessing a mix of RRS compliant resources can store synchronizing data in either a DB2 table or an MQ Queue. The preceding discussion covered synchronizing data in DB2 while the following topic addresses synchronizing data maintained by WebSphere MQ. Relational Architects recommends the use of DB2 for synchronizing data in this case.

50.7.3 Synchronization Data Maintained in WebSphere MQ

Synchronization data in an MQSeries Queue is supported in conjunction with applications using WebSphere MQ exclusively, as well as with applications which access several RRS compliant resource managers.

In an WebSphere MQ only environment, Smart/RESTART uses program SRSDMQS for all access to synchronizing data in an MQ Queue. SRSDMQS issues MQ checkpoints and rollbacks. Load module SRSDMQS should be specified in the RMI parameter for WebSphere MQ applications that store synchronizing data in an MQ queue. It includes CSECTs SRSDMQS and CSQBSTUB.

<u>Load Module</u>	<u>Linked Stub</u>	<u>MQ Attach</u>	<u>Synchpoint Coordinator</u>
SRSDMQS	CSQBSTUB	MQ	WebSphere MQ for z/OS

For applications which access WebSphere MQ queues as well as other RRS compliant resources, program SRSRMQS is used to maintain synchronizing data in a WebSphere MQ queue. SRSRMQS requests RRS checkpoints (SRRCMIT) and rollbacks (SRRBACK) and calls program SRSDMQS for all other MQ access. Load module SRSRMQS should be specified in the RMI parameter when RRS services are required and synchronizing data resides in an MQ queue. The SRSRMQS load module includes CSECTs SRSRMQS, SRSDMQS and CSQBRSTB.

<u>Load Module</u>	<u>Linked Stub</u>	<u>MQ Attach</u>	<u>Synchpoint Coordinator</u>
SRSRMQS	CSQBRSTB	MQ	RRS

50.7.4 RMI Summary

The table below summarizes the various load modules used by Smart/RESTART for accessing synchronizing data. The RMI parameter names one of these load modules. Note that any application program supported by Smart/RESTART can always access VSAM files and sequential files in addition to the resource managers shown below.

Load Module	Sync Data	IBM Resource Managers Used By Application Program
SRSDBRM	DB2 accessed via Smart/RRSAF	DB2 only
SRSDBRMT	DB2 accessed via TSO attachment	DB2 only
SRSDBRMI	DB2 accessed via IMS attachment	Any combination of DB2, WebSphere MQ and IMS
SRSDBRM0	DB2 accessed via other DB2 attachment	DB2 only
SRSRDB2	DB2 accessed via Smart/RRSAF under RRS	Any combination of RRS compliant resources. IMS must be accessed via ODBA
SRSRMQS	WebSphere MQ under RRS	Any combination of RRS compliant resources. IMS must be accessed via ODBA
SRSDMQS	WebSphere MQ	WebSphere MQ only

The default value for RMI is SRSDBRM. However, this can be changed when Smart/RESTART is installed, so if you depend on the default, check with the Smart/RESTART administrator.

Chapter 51

Restart Points

51.1 Introduction

Job steps run under the control of Smart/RESTART are classified as either initial runs or restart runs. The first run is initial. Should it abend, all subsequent runs are considered restart runs. There can be multiple restart runs. When a restart run finishes normally, the job step is considered complete.

Restarting a batch job step (as opposed to re-running from the beginning) requires that it take periodic checkpoints. When an initial run abends, the subsequent restart run begins from the last successful checkpoint in the initial run. Should a restart run abend, the *next* restart run starts from the last successful checkpoint in the *previous* restart run.

NOTE: The word "checkpoint" is used above, and elsewhere in this document, to represent a concept. IBM resource managers (DB2, IMS, WebSphere MQ) and IBM Resource Recovery Services (RRS) each have their own checkpoint commands, as does Smart/RESTART.

Smart/RESTART supports four Restart Point options which are specified via the RESTART_POINT parameter as: START, COMMIT, EXPLICIT or READ. They differ as to the processing point from which a program is resumed in a restart run.

RESTART_POINT(START) describes the default restart strategy. Figure 51.1 shows an initial run and a restart run using it.

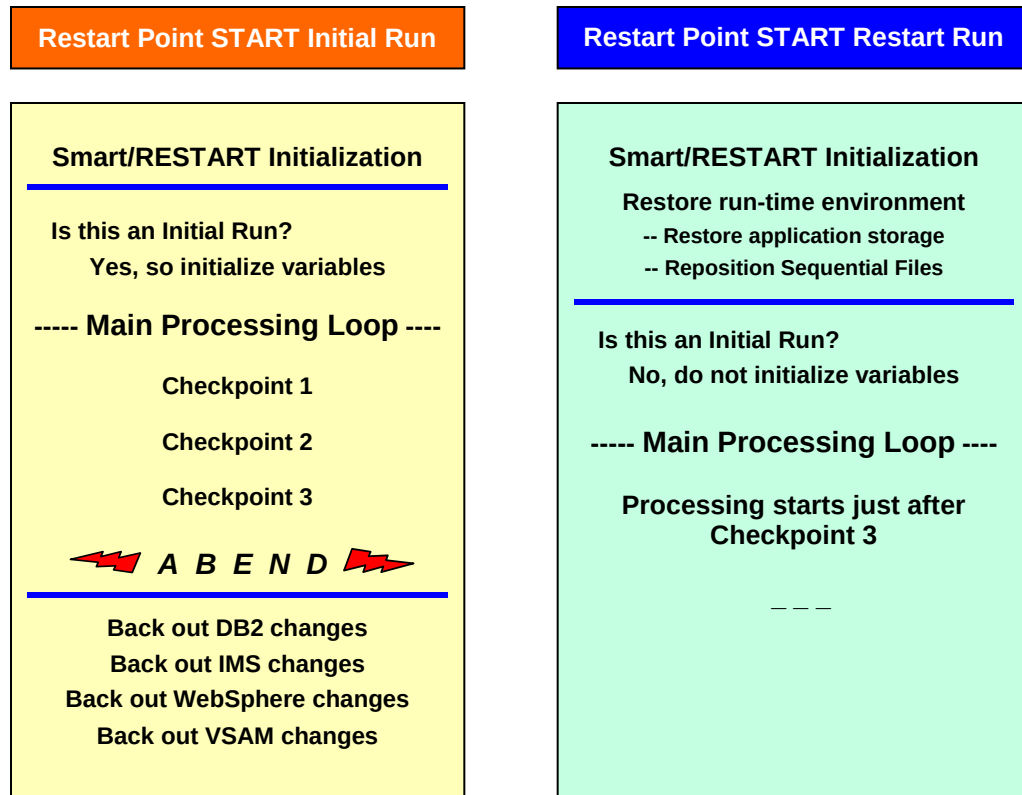


Figure 51.1 An Initial Run and a Restart Run Without Restart Points

As the name implies, execution always begins at the start of an application program. As Figure 51.1 shows, restartable programs begin with Smart/RESTART initialization. This call to Smart/RESTART to perform initialization is inserted automatically in the source code for programs prepared with Smart/PRECOMPILER. Programs prepared with COBOL COPY members or PL/1 INCLUDE text need to manually insert the necessary code.

The term "first application statement" is used to distinguish between processing that takes place outside the scope of an application (Smart/RESTART initialization) and the first statement that is part of the application. In Figure 51.1, *the first application statement* is a test for an initial run.

DB2 programs contain initialization code inserted by the DB2 precompiler or SQL coprocessor at the beginning of the program (but is not shown in Figure 51.1 because it is not relevant to restarts).

Typically, batch programs perform application initialization and then enter a main processing loop. This initialization (separate and distinct from Smart/RESTART initialization) often entails the initialization of variables with procedural statements and may involve other processing as well.

With Restart Point START, it is critical that the re-initialization of variables not take place in a restart run. This is why the first application statement shown in Figure 51.1 is a test for the run type. Figure 51.1 shows that in a restart run, application storage is restored to the state as of the last successful checkpoint, prior to the first application

statement. If variables were procedurally re-initialized in a restart run, they would be out of sync with the rest of the environment. Without Restart Points, application programs have to ensure that variable initialization only occurs in an initial run.

NOTE: *Variables are initialized either procedurally or statically. Procedurally refers to run-time initialization while statically in COBOL refers to Working Storage fields initialized with a VALUE clause. Assembler Language variables defined with DC and PL/1 variables defined with the INIT clause are also statically initialized. Statically initialized variables are **not** a problem for restarts.*

Smart/RESTART deals with various types of resources: VSAM files, application storage areas (in COBOL the Working Storage section), sequential files and protected resources maintained by such products as DB2, IMS and WebSphere MQ. Restartable application programs can update any combination of these six resources. When Smart/RESTART takes a **Smart** checkpoint, all types of resources are checkpointed together.

After an abend however, they are not dealt with identically. Figure 51.1 shows how changes to VSAM, DB2, IMS and WebSphere MQ resources are backed out immediately. Updates to these resources are logged in real-time as the application executes, enabling changes since the last successful checkpoint to be rolled back at abend time. This is true both with and without Restart Points.

NOTE: *All Smart/RESTART batch jobs that update IMS resources will dynamically back out IMS changes at abend time. This is not always the case for IMS batch jobs run without Smart/RESTART.*

In contrast, changes to application storage and sequential file position are not logged during execution. Instead, they are saved at every **Smart** checkpoint. They are not restored (to their state as of the last successful checkpoint) until the restart run. *When* in the restart run they are restored depends on the restart strategy. Figure 51.1 shows that with Restart Point START, they are restored as part of Smart/RESTART initialization at the *start* of the restart run.

51.2 Restart Points

Restart Points let you control, in a restart run, **when** application storage is restored and sequential files are repositioned -- a process known as restoring the run-time environment. Restart Points can also be thought of as 'restore points'.

Restoring the run-time environment at the *start* of a restart run places a burden on your program to ensure that the procedural initialization of variables only occurs in an initial run. Restart points are intended to be placed *after* application initialization and *before* the main processing loop. This allows application initialization to run unconditionally. Any changes made to variables in the initialization routines are reset when application storage is restored, at the restart point, to its state as of the last successful checkpoint.

With Restart Points, programs no longer need to be sensitive to the run type. This makes the conversion of existing non-restartable programs easier. Simply add a Restart Point just prior to the main processing loop.

Restart Points have no effect on the basic theme of restarting from the last successful checkpoint. The run-time parameter `RESTART_ID` is separate and independent of Restart Points.

The last successful checkpoint is not always the last executed checkpoint. Checkpoint pacing (as an integral feature of Smart/RESTART) can be used to control the frequency of checkpoints. When checkpoint pacing is used, checkpoints are classified as physical or logical. A physical (i.e. externalized) checkpoint is a real checkpoint while a logical checkpoint is one issued by the application that may or may not be honored by Smart/RESTART. The last successful checkpoint, when checkpoint pacing is employed, is the last *physical* checkpoint.

The use of explicit Restart Points is optional with `RESTART_POINT(START)` as the default mode of operation.

Restart Points have no effect on the `REPO_STYLE` used to reposition sequential files.

Implementing Restart Points requires coding one statement in the application program and specifying the `RESTART_POINT` parameter at run-time. The statement indicates *where* the Restart Point is within the application. The Restart Point strategies (`COMMIT`, `EXPLICIT` and `READ`) differ only in the type of source statement that indicates the Restart Point.

- With `RESTART_POINT(COMMIT)`, the indication is made with a checkpoint request -- either an `EXEC SQL COMMIT`, `SRRCMIT` call or a Smart/RESTART checkpoint. In an initial run, this first checkpoint functions as a normal checkpoint. In a restart run, the first checkpoint executed identifies the Restart Point to Smart/RESTART. The term "priming checkpoint" describes this first checkpoint, since it "primes" the restart rather than invoking an actual checkpoint.
- An `EXPLICIT` Restart Point is indicated by a call to Smart/RESTART's `RESTART` service. In effect, this call says "Restart Here!" To insulate programmers from the Smart/RESTART API, the call statement is provided as a `COPY` member for COBOL and as an `INCLUDE` for PL/1 and C.
- With Restart Point `READ`, the first read of a sequential file identifies the restart point. In a restart run, the run-time environment is restored. The read is then executed against a repositioned file.

Programs that conditionally open files are candidates for Restart Point processing. For example, an application might open a file and set a flag indicating the file is open. Later processing tests the flag to avoid opening the file a second time. Without Restart Points, this application is not restartable. When the file open flag is restored, along with the rest of program storage at restart time, it indicates the file is open. Programs that test this flag will never open the file and subsequent I/O will fail. Such applications can now take advantage of the fact that in a restart run, prior to the Restart Point, the application assumes it is performing an initial run and therefore opens the file.

51.3 RESTART_POINT Syntax

Chapter 12 of the Smart/RESTART Reference documents the RESTART_POINT parameter.

51.4 Restart Point START

With `RESTART_POINT(START)` the sequence of events at restart time (as illustrated in Figure 51.1) is as follows:

1. At the beginning of a restart run, Smart/RESTART gets control, determines that it is a restart run and that the restart strategy is `START`
2. Smart/RESTART restores the run-time environment to the state as of the last successful checkpoint in the previous run. Application storage areas are restored and sequential files are repositioned
3. An indication is set that this is a restart run
4. Control is passed to the first application statement in the program
5. The program can optionally test for the run type (initial or restart)

By the time the first application statement in the restarted program executes, all storage areas have been restored to their state as of the last checkpoint and sequential files have been repositioned. The only environmental difference between the restart run and the initial run is the run type indicator set by Smart/RESTART.

Application programs need to ensure that the procedural initialization of variables occurs **only** during an initial run. Initializing variables in a restart run (via COBOL MOVE statements, for example) will **overlay** the values restored by Smart/RESTART - when `RESTART_POINT(START)` is in force - and render them inconsistent with the rest of the environment. This is likely to result in incorrect output. Moreover, this type of error can not be detected by Smart/RESTART.

Figure 51.2 presents pseudo code for a program designed for use with Restart Point `START`. Note that the very first thing the program does is check for an initial run and ensures that the initialization routine only executes in an initial run. The assorted Open statements in Figure 2 must execute unconditionally.

```
If Initial Run Then
  Do
    Move initial values to fields
  End
Open VSAM and Sequential files
Open DB2 cursors and/or WebSphere MQ Queues
Do while (more input)                                (main processing loop)
  Do Unit-of-Work
    Read, Write and/or Update VSAM and Sequential Files
    Update DB2, WebSphere MQ and/or IMS resources
  End
  If Unit-of-Work Successful Then Checkpoint
  Else Rollback
End
```

Figure 51.2 *Pseudo Code for use with Restart Point START*

51.5 Restarting at the Main Processing Loop

Restart Points can be used to **logically** restart a program at a point **other** than the first application statement. The optimal restart point is just prior to the main processing loop because a failed job step is resumed at the last successful checkpoint -- a unit-of-work boundary.

The use of Restart Points is indicated in the RESTART_POINT parameter by specifying a value of EXPLICIT, COMMIT or READ.

When using Restart Points, programs execute normally up to the Restart Point. Application initialization (separate and distinct from Smart/RESTART initialization) runs unconditionally.

At the Restart Point, Smart/RESTART first backs out any changes made to protected resources (such as those managed by DB2, VSAM, IMS, and WebSphere MQ) during application initialization. Then it restores application storage and repositions sequential files to their state as of the last successful checkpoint. The net effect is a program and a run-time environment ready to resume processing just after the last checkpoint.

This is illustrated in Figure 51.3 which shows an initial run and restart run with Restart Points. The Restart Points in this example can be either (1) a priming checkpoint, (2) a sequential file read request, or (3) explicitly specified.

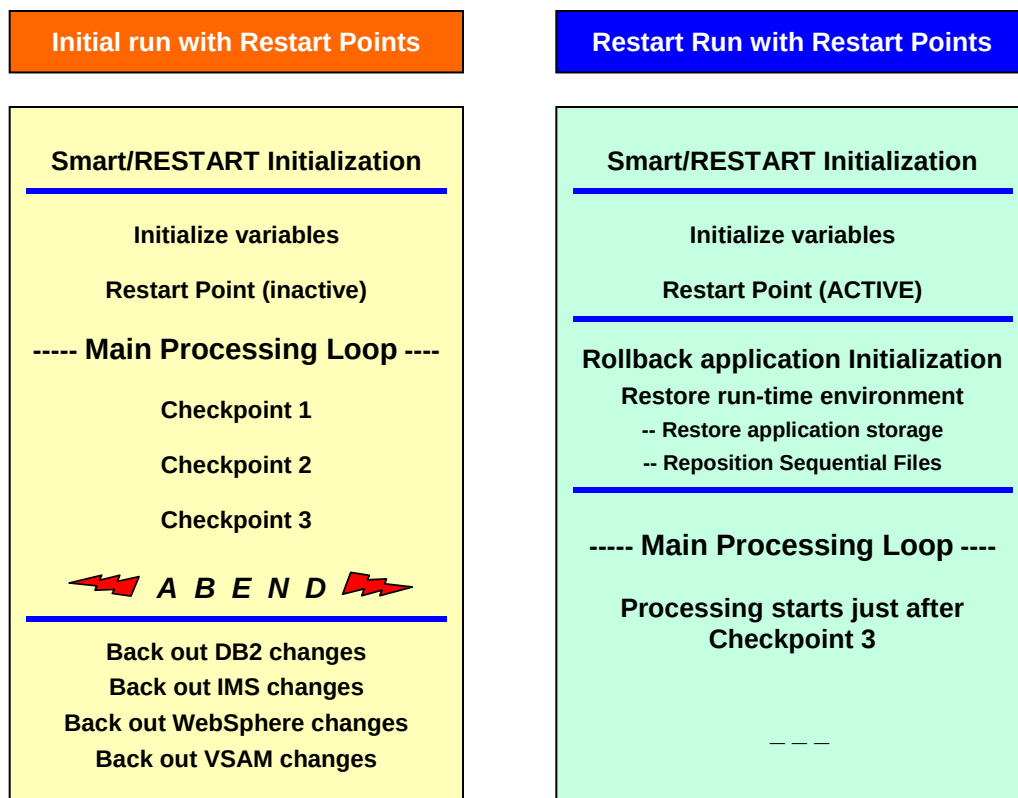


Figure 51.3 An Initial Run and a Restart Run With Restart Points

Like the example without Restart Points (shown previously in Figure 51.1), this illustrates an initial run that abends some time after the third checkpoint and is restarted at a point just after the third checkpoint. Also as in Figure 51.1, each job step starts with Smart/RESTART initialization and changes made after the third checkpoint to DB2, IMS, WebSphere MQ and VSAM are backed out at abend time. Much however, is different.

The initialization of variables is now unconditional and the program no longer tests for an initial run. Restart Points (a single source code statement) have been added. The Restart Point is inactive in an initial run but triggers the restoration of the run-time environment in the restart run. The restart run includes an extra step to roll back changes made to protected resources prior to the Restart Point.

With all restart strategies, by the time the main processing loop starts in a restart run, all application resources have been restored to their state as of the last successful checkpoint.

Smart/RESTART never changes or interferes with the control flow of a restartable program. In the restart run in Figure 51.3, for example, control flows sequentially from (1) Smart/RESTART initialization to (2) initializing application variables to (3) the Restart Point and then to (4) the main processing loop.

After Smart/RESTART initialization and prior to the Restart Point, restart is considered "pending". Changes made by the application program in this "restart pending" window are backed out by Smart/RESTART rollback processing. At the end of the restart pending window, Smart/RESTART performs rollback processing, restores the run-time environment to its state as of the last successful checkpoint, and then passes control to the next statement in the application program.

The exact sequence of events in a restart run using Restart Points (EXPLICIT, COMMIT or READ) is as follows:

1. At the start of a restart run, Smart/RESTART gets control. It determines this is a restart run and that the restart strategy is other than `RESTART_POINT(START)`
2. Smart/RESTART sets up a "restart pending" environment.
Message SRS100I is issued to indicate the beginning of the restart pending window.

```
SRS100I - RESTART processing is pending because the RESTART_POINT
SRS100I - parameter does not specify START
```

3. Control is passed to the first application statement in the program.
4. The application program executes until the Restart Point is encountered.
5. If the Restart Point is **EXPLICIT**, message SRS213I indicates the beginning of the restart point

```
SRS213I - RESTART processing is resumed by the first RESTART API call
SRS213I - since RESTART_POINT is set to EXPLICIT
```

If the Restart Point is **COMMIT**, message SRS211I indicates the beginning of the restart point

```
SRS211I - RESTART processing is resumed by the first CHECKPOINT request
SRS211I - since RESTART_POINT is set to COMMIT.
```


If the Restart Point is **READ**, message SRS212I indicates the beginning of the restart point

SRS212I - RESTART processing is resumed by the first RSAM READ request
SRS212I - since RESTART_POINT is set to READ.

6. Smart/RESTART performs a **Smart** ROLLBACK. Any and all changes made by the application program up to the Restart Point are backed out.
7. Smart/RESTART restores the run-time environment (application storage, file position, etc.) to the state as of the last successful checkpoint.
8. Control is passed to the first statement after the Restart Point. The restart is no longer considered pending. Smart/RESTART will now indicate to the program that this is a restart run.

To illustrate the environment before and after the restart point, consider a batch program where the main processing loop is driven by reading a sequential input file. Assume the program abended in the initial run on the 87th record and the last successful checkpoint was after record number 80.

If this input file is read prior to the restart point (that is, in the restart pending window) the first record read is record number 1. At the restart point, Smart/RESTART repositions the file to its position as of the last successful checkpoint. When the file is read after the restart point, (that is, in the main processing loop) the first record read is number 81.

In the restart pending window, initialization routines intended to run once are performed a second time. This is not a problem because Smart/RESTART backs out changes made while a restart is pending. The conversion of non-restartable programs to run restartably (with Restart Points) is straightforward because in both environments, all initialization processing occurs unconditionally.

Creating restartable programs that use Restart Points other than START entails the following:

- For newly developed programs, include a Restart Point
- When converting existing non-restartable programs, add a Restart Point

The “restart pending” window is optimally used for initializing variables with procedural statements. Other processing can occur in this window, but care should be taken in doing so, despite the fact that changes will be backed out.

A potential problem with application processing prior to the restart point can be illustrated by expanding the previous example. Suppose that in addition to reading input VSAM records, the program also inserts them into a DB2 table. When the restart run commences the first 80 records are already inserted. If the program processes the input file prior to the restart point, it would read record number 1 and attempt to insert it a second time into the same DB2 table.

Figure 51.4 shows the basic structure of restartable jobs with and without Restart Points. Job A, using Restart Point START, is a simplified version of the jobs shown in Figure 51.1. Job B, using Restart Points, clearly shows the “restart pending” window. Compared to Job A, it no longer tests for the run type but includes source code that indicates the restart point.

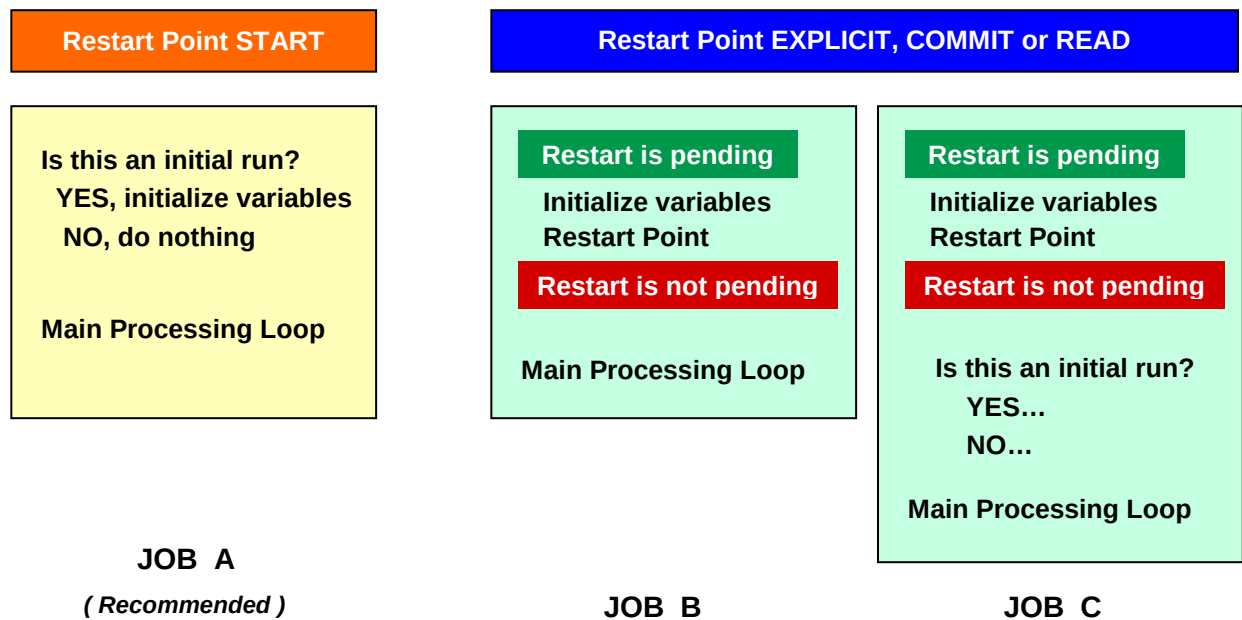


Figure 51.4 Structuring Restartable Jobs

Job C is a hybrid in that it uses a Restart Point and tests the run type. Jobs like this must be aware of the restart pending window. Prior to the Restart Point, Smart/RESTART **always** indicates an initial run. Only *after* the Restart Point does Smart/RESTART return an accurate indication of the run type. This is by design because prior to the Restart Point, restart has *not* occurred.

Pseudo code for programs using Restart Points is presented in Figure 51.5. It shows the optional run type test after the restart point. See Chapter 2.12 ('Distinguishing an Initial Run From a Restart Run') for more details.

```

Do Initialization
  Move initial values to work areas and fields
  Open VSAM and Sequential files
  Open WebSphere MQ Queues
End

Restart Point    (Priming checkpoint or Explicit or File Read)
Check for initial run vs. restart run    (optional)
Open DB2 cursors

Do while (more input)    (main processing loop)
  Do Unit-of-Work
    Read, Write and/or Update VSAM and Sequential Files
    Update DB2, WebSphere MQ and/or IMS resources
  End
  If Unit-of-Work Successful Then Checkpoint
  Else Rollback
End

```

Figure 51.5 Pseudo Code for use with Restart Point EXPLICIT and COMMIT

As previously described, the initialization routine runs unconditionally. The Open for WebSphere MQ queues, although shown in the initialization routine, can be coded anywhere in the application program. The Open commands for sequential files and VSAM datasets can also occur anywhere, provided the files are open before I/O occurs.

DB2 cursors are best opened after the restart point. To back out application initialization, Smart/RESTART performs rollback processing at the restart point. DB2 cursors are closed by rollbacks. This rollback will also backout changes made to VSAM files as well as to DB2, IMS, MQ resources -- prior to the restart point. Changes made to application storage areas (variables) are reset when storage is restored to the state as of the last successful checkpoint. As previously noted, sequential files are repositioned at the Restart Point. Furthermore, Smart/RESTART guarantees that committed records are *never* overwritten.

51.6 Restart Point EXPLICIT

The RESTART_POINT(EXPLICIT) strategy defers the restoration of the run-time environment until the point in the program that you explicitly designate. This point, typically the start of the main processing loop, is termed the “Explicit Restart Point”. In effect, an Explicit Restart Point says "Restart Here!".

Explicit Restart Points are indicated by a call to the Smart/RESTART API, as shown in Figure 51.6. Readers familiar with the API will note that the calls in Figure 51.6 invoke the RESTART service. To insulate programmers from the Smart/RESTART API, the call statement is provided as a COPY member for COBOL and as an INCLUDE for PL/I and C/C++. The third column in Figure 51.6 is the member name in the Smart/RESTART sample library (DCASAMP) for the included source code.

The source code in Figure 51.6 is shown here mainly for illustrative purposes. The use of COPY members or INCLUDEs is highly recommended for coding Explicit Restart Points. It provides flexibility for the future and is less prone to error.

Language	Source Code	DCASAMP Member
COBOL	CALL RAI-RAIAPI USING RAI-RESTART, FCA	SRSCREST
PL/I	CALL RAIAPI(RAI_RESTART,FCA);	SRSPREST
C/C++	RAIAPI(rai_restart, &fca);	SRSHEST

Figure 51.6 Coding Explicit Restart Points

NOTE: The FCA is a data structure included in every restartable program. The other parameter in each call is the new restart service.

Should an error occur during the Explicit Restart call, an E level message is issued, a user abend occurs and control does not return to your program. The possible return codes from the Restart service are:

0	Restoration of the run-time environment was not triggered. This is <i>normal</i> in an initial run.
2001	Restoration of the run-time environment was successful and a restart run is in progress.
4	A restart run was detected <i>before</i> this API call. The call is ignored.

The use of Explicit Restart Points is fully supported from Assembler Language.

In a restart run, the Explicit Restart Point *must* be executed prior to any checkpoint commands. Smart/RESTART abends the program if it encounters a checkpoint request while restart is pending. EXEC SQL SAVEPOINT requests can be issued prior to the Explicit Restart Point, but RAI does not recommend this practice.

Explicit Restart Points can be coded in either the main program or a sub-program, but RAI strongly recommends they be coded in the main program. The unit-of-work boundaries are best defined in the main program as is coding an Explicit Restart Point.

In an initial run, the Explicit Restart Point is basically a no-op.

51.7 Restart Point COMMIT

The RESTART_POINT(COMMIT) strategy defers the restoration of the run-time environment until the first checkpoint. The checkpoint that initiates restoring the run-time environment is referred to as a “priming” checkpoint. It serves only as a flag to Smart/RESTART since an actual checkpoint is not taken.

The concept of a priming checkpoint only applies in a restart run. In an initial run, the priming checkpoint is processed as a normal checkpoint, even though there may be no resource manager changes to commit.

The priming checkpoint is the first *executed* checkpoint, regardless of its physical location in the program. A priming checkpoint is coded the same as any other checkpoint. Anyone reading the source code is dependent on comments to differentiate the intent of a priming checkpoint from other checkpoints.

The following checkpoint requests are recognized as priming checkpoints.

- EXEC SQL COMMIT for DB2
- EXEC SRS COMMIT for the EXEC level Smart/RESTART API

- CALL "RAIAPI" USING RAI-CHECKPOINT, FCA for the call level Smart/RESTART API
- Call SRRCMIT

Note that IMS and WebSphere MQ checkpoint commands are **not** recognized as priming checkpoints, nor can EXEC SQL SAVEPOINT requests serve as priming checkpoints. EXEC SQL COMMIT and call SRRCMIT are recognized as priming checkpoints when used in conjunction with Smart/RRSAF as the DB2 attachment.

DB2 programs that use an attachment other than Smart/RRSAF can still use an EXEC SQL COMMIT as a priming checkpoint if they are prepared with Smart/PRECOMPILER and the CAF option is correctly set. Smart/PRECOMPILER has the ability to replace EXEC SQL COMMIT requests with calls to the Smart/RESTART checkpoint service. The advantage to this is that Smart/RESTART checkpoints are always recognized as priming checkpoints, regardless of the DB2 attachment.

Figure 51.5 shows a Restart Point after the initialization section and a checkpoint after a unit-of-work. In a DB2 program, for example, each of these could be different EXEC SQL COMMIT statements or they could both invoke the same subroutine to issue the COMMIT. Priming checkpoints can also be issued from sub-programs. They are not limited to the main program.

In addition to Restart Point COMMIT, a number of other Smart/RESTART features also govern the behavior of checkpoints.

Checkpoint pacing categorizes checkpoints as either logical or physical. Logical checkpoints are ones issued by the application but not honored by Smart/RESTART. Coding CKPT_PACE(10) in RAINPUT causes the first nine checkpoint requests to be treated as logical checkpoints. Despite this, the first checkpoint request issued by the program can be a priming checkpoint.

The INIT_COMMIT parameter, when set to NEXT instructs Smart/RESTART to ensure that the first logical EXEC SQL COMMIT request is treated as a physical COMMIT. With Restart Point COMMIT, the first logical EXEC SQL COMMIT request in a restart run is never a physical checkpoint. Rather it is a priming checkpoint which results in rollback processing. In effect, Restart Point COMMIT overrides INIT_COMMIT (NEXT) in a restart run.

When the INIT_COMMIT parameter is set to BASELINE or REFRESH, it causes Smart/RESTART to issue a checkpoint as part of its initialization processing. This checkpoint is never a priming checkpoint either, because it is only taken in an initial run.

Smart/RESTART can also take checkpoints based on **external events** (as described in Chapter 44 of this User Guide). For example, externally defined units-of-work and their associated checkpoints can be controlled by reading an input driver file. The first checkpoint triggered by an external definition can be a priming checkpoint.

51.8 Restart Point READ

The `RESTART_POINT(READ)` strategy defers restoration of the run-time environment until the first read of a repositionable sequential file. Like the other deferred restart strategies (Explicit and Commit), Restart Point READ works best if the restart point is just prior to the main processing loop.

Restart Point READ is designed for use with programs where changing the source code is not an option. When source code changes are possible, `RESTART_POINT(EXPLICIT)` or `RESTART_POINT(START)` are preferable.

The file read request that initiates restoring the run-time environment is referred to as the priming read. It serves as both a trigger for restart processing and as a file read request. The run-time environment is restored first and then the read is executed against a file that has been repositioned. This contrasts with a priming checkpoint, which does not result in an actual checkpoint.

Like priming checkpoints, priming reads only exist in a restart run. In an initial run, the file read is processed as a normal I/O request.

Return codes from a priming read are the same as from any other sequential file read. The sole indication that the run-time environment has been restored is from the Environment Check Service. After a priming read, it will indicate a restart run.

Priming reads can only be issued to sequential files accessed via the RSAM component of Smart/RESTART. Typically, all sequential files are accessed using RSAM. Reads of VSAM files, even when accessed sequentially, are not treated as priming reads. The `REPOSITION` parameter determines the files that are accessed with RSAM.

The priming read is the first *executed* file read, regardless of its physical location in the program. Priming reads can be coded in either the main program or a sub-program, but it is recommended to code them in the main program.

51.9 Advantages of `RESTART_POINT(EXPLICIT)` vs. `COMMIT` and `READ`

An Explicit Restart Point offers several advantages over Restart Points triggered by a priming `COMMIT` or priming `READ`.

- It is easier to identify where Restart takes place because the sole purpose of `RESTART_POINT(EXPLICIT)` is to indicate "Restart here"
- Its action is predictable
- It works with all programming languages that support standard z/OS linkage conventions

- There is no ambiguity as to the meaning of program source code. When used to indicate Restart Points, a checkpoint command or file read does double duty. As a result, With Explicit Restart Points in contrast, Smart/RESTART can perform validity checks at run time that can not be performed with priming checkpoints or priming reads.
- Compared to COMMIT, RESTART_POINT(EXPLICIT) works with all RRS compliant resource managers (DB2, IMS, WebSphere MQ)
- Compared to COMMIT, RESTART_POINT(EXPLICIT) avoids the issue of whether an EXEC SQL COMMIT can be a potential priming checkpoint
- Compared to COMMIT, RESTART_POINT(EXPLICIT) avoids conflicts with other Smart/RESTART facilities with respect to checkpoints
- A priming checkpoint is processed as a normal checkpoint in an initial run, even though typically, there are no resource manager changes to commit. In contrast, an explicit restart command does nothing in an initial run.

51.10 Object Transparency and Restart Points

Restart Points require adding a statement to your program to indicate the Restart Point. This obviously requires access to the source code and the ability to prepare the application. Object Transparency is designed for use with programs that are impractical or impossible to change and recompile (such as when source code is not available). Despite this, Object Transparency can be used with both Restart Point COMMIT and READ.

The goal of Object Transparency is to take an existing load module and enable it for restartable operation. To accomplish this without program preparation requires changes to the JCL to run the load module under the control of a Smart/RESTART control program. Object Transparency can be used with any load module, even one for which source code exists.

Since Restart Point COMMIT and READ depend on standard host language statements, they can be used with existing non-restartable programs and load modules. However, these programs should be examined to ensure that they will operate correctly if restarted from either the first executed checkpoint or first sequential file read request. Any ambiguity can be removed by inserting an API call to the RESTART service just prior to the main processing loop and re-compiling the program.

51.11 Mismatched Specifications

This section discusses some situations that can occur in a restart run when the restart strategy specified by the `RESTART_POINT` parameter is at variance with the restart strategy the source code actually implements. The material in this section helps to avoid these situations.

If the `RESTART_POINT` parameter specifies `START` but the program is coded with an Explicit Restart Point, the restart strategy will be `START`. When the Explicit Restart Point is encountered in a restart run, Smart/RESTART issues message `SRS214W` warning that it was ignored. Since the restart strategy is `START`, the program *must not* run initialization routines in a restart run. Doing so will produce incorrect results.

```
SRS214W - RESTART API call is ignored since RESTART processing is not pending
```

If the `RESTART_POINT` parameter is `COMMIT` and the program is coded with an Explicit Restart Point, the results depend on whether a checkpoint is requested at run-time prior to the Explicit Restart Point. If so, the restart strategy is `COMMIT`, the Explicit Restart Point is ignored, and message `SRS214W` is issued when the Explicit Restart Point is encountered. If, however, the Explicit Restart Point is encountered prior to any checkpoint commands, then Smart/RESTART treats this as an error. Message `SRS202E` is issued (as shown below) and a user abend is triggered with a reason code of 00001009.

```
SRS202E  An explicit RESTART request is denied.
SRS202E  The RESTART_POINT parameter was not set to EXPLICIT.
SRS202E  Specify RESTART_POINT(EXPLICIT) and resubmit the job for
         a manual RESTART
```

If the `RESTART_POINT` parameter is `READ` and an Explicit Restart Point is encountered *prior* to the priming read, then as above, message `SRS202E` is issued and a user abend is triggered with a reason code of 00001009.

If the `RESTART_POINT` parameter is `EXPLICIT` and there are two different Explicit Restart Points in the program, the first one is honored (in a restart run) and the second one causes Smart/RESTART to issue warning message `SRS214W`. This warning is issued only once per job step, even if multiple Explicit Restart Points are ignored.

If the `RESTART_POINT` parameter is `EXPLICIT` but there is no Explicit Restart Point in the program, Smart/RESTART will trigger an abend at the first checkpoint. The topic on Explicit Restart Points notes that checkpoint commands can not be issued prior to the Explicit Restart Point -- this is basically the same situation. Message `SRS201E` (shown below) is issued and a user abend occurs with a reason code of 00001009.

```
SRS201E - An implicit RESTART request during CHECKPOINT is denied.
SRS201E - The RESTART_POINT parameter was not set to COMMIT. Specify
SRS201E - RESTART_POINT(COMMIT) and resubmit the job for a manual RESTART
```

Note that Smart/RESTART can not detect the equivalent situation with Restart Point `COMMIT` -- i.e. it can not determine whether the priming checkpoint has been omitted. The first qualifying checkpoint will be treated as a priming checkpoint -- whether intended by the programmer or not.

If the `RESTART_POINT` parameter is `READ` and a checkpoint is encountered prior to the priming read, then as above, message `SRS201E` is issued and a user abend is issued.

When the `RESTART_POINT` parameter is `EXPLICIT` or `COMMIT`, sequential file reads issued prior to the Restart Point are treated as normal I/O requests. Smart/RESTART does not consider this a mismatched specification.

To ensure that an application program runs with the intended restart strategy, Smart/RESTART provides access to the `RESTART_POINT` parameter at run-time. Application programs can read but not change the restart strategy. Figure 51.7 illustrates evaluating the restart strategy from COBOL using the Smart/RESTART call level API.

In addition to the code in Figure 51.7, COBOL programs have to manually include COPY member `SRSCEDA` in their Working Storage section. It is not automatically included by Smart/PRECOMPILER.

```
CALL RAI-RAIAPI USING RAI-ENVCHECK FCA EDA.  
EVALUATE EDA-RESTART-POINT  
  WHEN 'C'  DISPLAY 'RESTART_POINT is COMMIT'  
  WHEN 'E'  DISPLAY 'RESTART_POINT is EXPLICIT'  
  WHEN 'R'  DISPLAY 'RESTART_POINT is READ'  
  WHEN 'S'  DISPLAY 'RESTART_POINT is START'  
  WHEN OTHER DISPLAY 'RESTART_POINT is invalid'  
END-EVALUATE
```

Figure 51.7 *Accessing and Examining the `RESTART_POINT` strategy in COBOL*

51.12 Distinguishing Between an Initial Run and a Restart Run

Restart Points can change the restart environment in such a way that the standard methods of differentiating an initial run vs. a restart run are not always valid. The following discussion describes additional mechanisms to determine the run type in different circumstances. Regardless of the method used to determine the run type, the test should only be performed once.

RESTART_POINT (START)

`RESTART_POINT(START)` permits you to use the standard techniques described in Chapter 2 to determine the type of run. Sample COBOL code for this purpose is shown in Figure 51.8 and should be placed just after the Procedure Division header.

```
EVALUATE RETURN-CODE
  WHEN 2001 DISPLAY 'THIS IS A RESTART RUN'
  WHEN 0    DISPLAY 'THIS IS AN INITIAL RUN'
  WHEN OTHER DISPLAY 'SMART/RESTART INITIALIZATION ERROR
                     DETECTED'
END-EVALUATE
```

Figure 51.8 *Determining the type of run with RESTART_POINT(START)*

RESTART_POINT (COMMIT, EXPLICIT and READ)

The use of Restart Points is indicated by the COMMIT, EXPLICIT or READ option of the RESTART_POINT parameter. Restart Points change the rules for determining the run type.

The main reason for this is that the restart environment does not exist *prior* to the Restart Point. Also, a priming checkpoint can be issued from a sub-program, which can invalidate the testing of the Smart/RESTART control field to determine the type of run.

When using Restart Points, the environment in a restart run prior to the Restart Point exhibits characteristics of *both* an initial run and a restart run. Externally, it seems like a restart run, in that an initial run suffered a failure and the batch job is re-submitted. Internally, however, the restart environment does not exist ***until the Restart Point is encountered***. Indeed, the main purpose of the Restart Point is to indicate where processing should be resumed.

In a restart run, tests for the run type (as in Figure 51.9) prior to the Restart Point will indicate an initial run. Only after the Restart Point will Smart/RESTART indicate that this is a restart run. *As such, the test for run type belongs just after the Restart Point.*

Smart/RESTART provides an additional mechanism for determining the type of run. Implemented as part of Smart/RESTART's environment check service, this mechanism ***always*** works, regardless of the Restart Point option in effect at run-time.

The environment check service is invoked via the Smart/RESTART call level API. For many restartable programs, especially those prepared with Smart/PRECOMPILER, this can be the first introduction of application code that is aware of Smart/RESTART. COBOL programs can call the environment check service using code similar to that shown in Figure 51.9.

```

CALL RAI-RAIAPI USING RAI-ENVCHECK,
                     FCA.
IF RETURN-CODE = 0 THEN
  DISPLAY 'SMART/RESTART IS ACTIVE'
  IF FCA-RESTART-RUN THEN
    DISPLAY 'THIS IS A RESTART RUN'
  ELSE
    DISPLAY 'THIS IS AN INITIAL RUN'
  END-IF
ELSE
  DISPLAY 'SMART/RESTART IS NOT ACTIVE'
END-IF

```

Figure 51.9 *Determining the type of run using the environment check service*

The return code following the call statement will be either 0 or 4. A value of 4 does not indicate an error, it means that the Smart/RESTART environment is not active. For a program that runs in both batch and CICS, a return code of 4 indicates that it is not running in batch mode.

The CALL statement in Figure 51.9 can be replaced by the COBOL COPY member SRSCENVC (from the DCASAMP library) to insulate programmers from the Smart/RESTART API. Sample COBOL code that both calls the environment check service and tests the return code (much like Figure 51.9) is provided in COBOL COPY member SRSCRCHK.

The sample code in Figure 51.9 merely displays the run type. You may prefer to set a flag field so that the type of run is available later in the program without having to call Smart/RESTART again. The run type test can be done in a sub-program, but for multi-module applications, RAI recommends that only the main module (program) be restart enabled.

Checking in COBOL for a RETURN-CODE value of +2001 is not valid when using Restart Points. The code in Figure 51.8 is only valid as the first application code in a program that uses RESTART_POINT(START).

There are two other issues when using a Restart Point of COMMIT. As just described, distinguishing between an initial run and a restart run should occur just after a priming checkpoint. When using the CKPT_TIME parameter with an action value of COMMIT, there is no one statement in your program that can be identified as the “priming” checkpoint. With this option, checkpoints are not issued based on code in your program. Instead, checkpoints are issued based on elapsed time. It is therefore not appropriate to use this in a program that uses RESTART_POINT(COMMIT) and requires knowledge of the run type.

External checkpoints based on a driver file also make it impossible to point to a specific statement in a program that triggers a priming checkpoint. The driver file determines the issuance of checkpoints based on either reading a specific number of input records (COMMIT_FILE with the COUNT option) or data in the input records (COMMIT_FILE with the CONTENT option). Determining when a priming checkpoint has been issued requires testing after every read of the input driver file. RAI recommends not using

external checkpoints with `RESTART_POINT(COMMIT)` in a program that needs knowledge of the run type.

In general, programs using Restart Point `COMMIT` or `READ` should not be sensitive to the run type. These options are designed for use with programs whose source code can not be changed. Programs that need to be aware of the run type are good candidates for `RESTART_POINT(EXPLICIT)` or `RESTART_POINT(START)`.

Debugging Procedures

This chapter deals with debugging in a Smart/RESTART environment. Debugging may involve problems with restartable applications or possibly defects in Smart/RESTART itself. This chapter presents a debugging procedure as a series of steps to help you characterize, identify and resolve problems. Within that context, this Chapter describes Smart/RESTART options, features and facilities that provide debugging support. The diagnostic information produced (in varying amounts) can be analyzed onsite and if necessary furnished to RAI Support.

One viable approach to problems that arise is as follows:

I Review Job Outputs

- I.1 Check JES messages (as described in Section 52.1.1)
- I.2 Check messages issued by Smart/RESTART (Section 52.1.2)
- I.3 Check messages issued by Smart/RRSAF (as described in Section 52.1.3)
- I.4 Review application specific messages and output produced by your program for indications of how far processing proceeded and why it might have failed.
- I.5 In the event of an abend, check any dump produced by the operating system and/or by Smart/RESTART. The SRSDUMP produced by Smart/RESTART contains diagnostic information pertaining to both Smart/RESTART and the IBM Language Environment (See Section 52.1.4)

Until the application jobstep runs successfully to completion, be sure to save the complete JOB output for both the initial run as well as any restart run(s).

II Review Job Setup and Inputs

- II.1 Check any Smart/RESTART runtime parameters specified via the RAINPUT DD statement (as described in Section 52.2.1)
- II.2 Review the composite Smart/RESTART runtime options as described in Section 52.2.2
- II.3 Verify that the target DB2 subsystem and application plan name are correct (Section 52.2.3)
- II.4 Check the DB2 application plan BIND to ensure the PKLIST (package list) includes the Smart/RESTART package named SRSDBRM (Section 52.2.4)
- II.5 Check whether the job is set up properly for either an initial run (cold start) or restart as described in Section 52.2.5
- II.6 Ensure you are running on the proper z/OS system (for either an initial run or restart run) as only certain systems may be licensed for Smart/RESTART, as discussed in Section 52.2.6.
- II.7 Check for procedures that might be inconsistent with successful restart or are at variance with your organization's guide to best practices (Section 52.2.7)

III Gather additional Diagnostics as required

- III.1 Produce additional Smart/RESTART messages by specifying MSG_DISPLAY(DEBUG) as described in Section 52.3.1.
- III.2 Obtain an Entry / Exit trace of application modules 'registered' with Smart/RESTART, as described in Section 52.3.2
- III.3 Enable function tracing of both Smart/RESTART API service calls as well as I/O requests to repositionable files (Section 52.3.3)
- III.4 Dump and review Operating Shift definitions (Section 52.3.4)
- III.5 Dump and review SQL_TRAP definitions (Section 52.3.5)
- III.6 Enable tracing of SQL calls, RRSAF and CAF services, DB2 commands and/or IFI requests as described in Section 52.3.6
- III.7 Make use of the debugging features of Smart/MONITOR (Section 52.3.7)
- III.8 Produce a formatted dump of the application's checkpoint file with the Smart/RESTART Utility program named SRSUTIL. (Section 52.3.8)

IV Operational Issues

- IV.1 Ensure repositionable files are copied correctly to avoid reblocking as described in Section 52.4.1.

52.1.1 Job Message LOG

The first step of the debugging procedure should be to review the messages appearing in the JES Message Log (JESMSGLOG). This log contains diagnostic messages written by Smart/RESTART and other software components which participate in the application run. Note that Smart/RESTART error messages are often issued *in response* to messages issued by system components like z/OS and DB2. You should consult the messages and codes manuals that correspond to these error messages.

52.1.2 The Smart/RESTART message LOG

Smart/RESTART issues various messages during a restartable jobstep to notify users about the progress of an application run. The Smart/RESTART message LOG may include *error*, *warning* and *informational* messages as described in the sections which follow. Error and warning messages are discussed first, followed by a discussion of Smart/RESTART informational messages.

The destination of these messages is specified via the MSGDD command (with DDname SRSRPRINT being the default). See Chapter 14 of the Smart/RESTART Reference Manual for a detailed description of the MSGDD command.

52.1.2.1 Error Messages

Smart/RESTART E-level messages denote a severe error or exceptional condition that forces Smart/RESTART to abnormally terminate the jobstep. The application becomes a candidate for manual or automatic restart.

Example: Smart/RESTART intercepts an SOC7 abend within a restartable application and reports the failure with a series of messages like the following:

```
RAI998E - Abend S 0C7 detected in subtask DCRA      Reason = 00000000
SRS096E - The Smart/RESTART abend handler detected an application failure
SRS096E - Smart/RESTART will flag the JOB as abended and requiring RESTART
SRS168E - A SOC7 ABEND has occurred
. . .
SRST016E - Smart/RESTART has terminated abnormally. Job RESTART is required
SRST021E - One or more errors occurred during the current run
SRST021E - Look for E - level messages issued by Smart/RESTART
```

Note that message RAI998E is issued whenever the RAI generic ESTAE routine traps an abend. The reference to subtask DCRA in the RAI998E message identifies the RAI component that *established* abend trapping. In this case, the component code 'DCRA' identifies Smart/RESTART. The RAI998E message means that Smart/RESTART *detected* an abend, not that Smart/RESTART itself *suffered* an abend.

Smart/RESTART duplexes its error messages to both the message file and the JOB LOG to better ensure that error messages are noticed by operations personnel.

52.1.2.2 Warning Messages

Smart/RESTART W-level messages warn users of events during an application run that are significant and/or require attention. Note though that Smart/RESTART *continues to process normally* after issuing a warning message.

Example: When a `CLEAR_CHKPT(BEFORE)` command appears in the RAINPUT stream, message SRTI123W (below) is issued to alert the user that Smart/RESTART will erase the checkpoint file and any residual restart information it contains. The `CLEAR_CHKPT(BEFORE)` directive is typically used during development and testing to force an initial run.

SRSI123W - The CLEAR_CHKPT parameter was set to BEFORE. An INITIAL run will
SRSI123W - be forced by destroying any residual restart data.

52.1.2.3 Informational Messages

Smart/RESTART I-level messages provide descriptive information and notifications of normal events such as:

- The Smart/RESTART Version and Release level.

SRSI003I - Smart/RESTART Version 8 Release 1 Modification 3 Level A1

- The resource manager(s) in use, if any:

DCA123I - Smart/RRSAF Version 812 connected to DSN7 via RRSAF - - Level A1.

- Compile time information about the application program that includes the compiler name, compile date and compiler options:

SRS336I - SRS8CPT was compiled using Enterprise COBOL for z/OS Version 030300
SRS336I - on 06/17/2005 at 18:06:18 . PGM received control through a nonCOBOL
SRS336I - call contains 00112 Procedure and 00273 Data Division statements.
SRS466I - SRS8CPT is main module of its ENCLAVE
SRS467I - 00001 QSAM File Definitions have been detected within module SRS8CPT

- Application initialization and termination

SRS469I - Module SRS8CPT was entered for the first time.

- Repositionable file activity such as OPEN

SRS432I - An application request to OPEN file OUTFILE detected by RSAM
SRS946I - Application DCBEXIT invoked to verify file attributes
SRS933I - BUFNO will be calculated for file OUTFILE regardless of DCBBUFNO.
SRS018I - Smart/RESTART File OUTFILE successfully Opened for OUTPUT
SRS024I - File OUTFILE currently consists of 01 volumes and 01 datasets.
SRS024I - DDName.. DSName..... Volume
SRS024I - OUTFILE DCA810.OUTFILE2 RA034A
SRS023I - DDName.. Mode.. LRECL BLKSIZE RECFM DSORG BUFNO Extended
SRS023I - OUTFILE OUTPUT 00080 06160 FB PS 016 No
SRS606I - Transparent OPEN for file OUTFILE from module SRS8CPT completed.

- Checkpoint and rollback processing

```
SRS610I - Disk file OUTFILE at trk 00000 blk 001 offset 0004640 rec 00000058
SRSB06I - A total of 00010 blocks were written during checkpoint at RRN 00000061
SRS406I - PHASE1 of COMMIT completed successfully
SRS410I - PHASE2 of COMMIT completed successfully
SRS601I - PHASE3 of COMMIT completed successfully
SRSB40I - LCOMMIT 0000002 issued while REGISTERED module SRS8CPT was in control
SRS031I - CHECKPOINT 0000002 is complete. CHECKPOINT ID is 0000002
SRS750I - Prediction interval bypassed due to insufficient input activity.
```

The preceding messages illustrate just some of the events about which Smart/RESTART issues notifications during an application run. Smart/RESTART issues informational messages for many other processes as well, such as a concatenation transition or a volume switch for a repositionable file.

You can limit the number and type of informational messages Smart/RESTART issues during a run via the MSG_DISPLAY command. Each Smart/RESTART message is defined in terms of a message filtering attribute of either *terse*, *verbose* or *debug*. In conjunction with the setting of the Smart/RESTART message display option (TERSE|VERBOSE|DEBUG) in effect, the two govern whether a particular message is logged or suppressed. Messages defined with the TERSE attribute are not suppressed while debugging messages (i.e. those defined with the debug attribute) are subject to the most restrictive filtering. Chapter 11 of the Smart/RESTART Reference Manual provides detailed information about the MSG_DISPLAY command.

The related command CKPT_MSG_LIMIT (described in Chapter 15 of the Smart/RESTART Reference) lets you suppress even the *terse* informational messages pertaining to checkpoint events, once Smart/RESTART exceeds a user specified limit.

Finally, note that certain events cause Smart/RESTART to *reset* the message suppression level specified by the MSG_DISPLAY and CKPT_MSG_LIMIT commands. In response to events such as the abnormal termination of a jobstep or a restart attempt, Smart/RESTART will externalize all informational messages -- regardless of their category.

52.1.3 Smart/RRSAF Message LOG

When Smart/RRSAF is used to access DB2, messages whose prefix is DCA appear in the Smart/RESTART message LOG. Should a DB2 or SQL related error occur, check the SRSRPRINT and JES LOGs for messages with the DCA prefix that embed and externalize information returned by DB2 itself (as illustrated in the following SQL error diagnostic).

```
DCA118W - Failed SQL statement SELECT was in program SRSDBRM : SQLCODE = -924
DCA118W - Section number in plan = 0004 . Precompiler statement number = 0000358
DCA096I - DSNT408I SQLCODE = -924, ERROR: DB2 CONNECTION INTERNAL ERROR, 0001,
DCA096I - 0100, 00F30018
DCA096I - DSNT418I SQLSTATE = 58006 SQLSTATE RETURN CODE
DCA096I - DSNT415I SQLERRP = DSNAET03 SQL PROCEDURE DETECTING ERROR
```

The DB2 related information embedded within a DCA message may include the DB2 message ID and message text along with any abend code, return code, eight character reason code and/or SQLCODE returned by DB2. The DB2 Messages and Codes manual provides definitive information about these errors.

52.1.3.1 Connect to DB2

```
DCA080I - Smart/RRSAF Version 8 Release 1 Modification 2 Level A1
DCA080I - (c) Relational Architects International - All rights reserved
DCA080I -
DCA080I - Job Name = RJSJ8CPT Step Name = SRS8CPT Procstep Name =
DCA080I - Run Date = 2005.206 : 07/25/2005 Run Time = 15.39.42
.
DCA123I - Smart/RRSAF Version 812 connected to DSN8 via RRSAF - - Level A1
DCA125I - A DB2 accounting record will be written when program exits
DCA011I - Thread created for application plan SRS8CPT
```

52.1.3.2 Disconnect from DB2

Disconnect messages from DB2 upon abnormal termination:

```
DCA014I - Thread for application plan SRS8CPT has terminated ABNORMALLY
DCA015I - Smart/RRSAF has disconnected from DB2 subsystem DSN8
DCA016I - Smart/RRSAF processing completed
```

Disconnect messages upon normal termination:

```
DCA014I - Thread for application plan SRS8CPT has terminated NORMALLY
DCA015I - Smart/RRSAF has disconnected from DB2 subsystem DSN8
DCA016I - Smart/RRSAF processing completed
```

Several Smart/RRSAF commands let you suppress SQL diagnostics. See the Smart/RRSAF User Guide for detailed information about the following commands: DIAG, DIAG_END_OF_JOB, DIAG_LIMIT, SQL_ERROR and SQL_WARNING.

52.1.4 Smart/RESTART formatted DUMP

In the event of an abend, check any dump produced by the operating system and/or by Smart/RESTART. Starting with Version 8.10, Smart/RESTART optionally produces a formatted dump that contains diagnostic information pertaining to both Smart/RESTART and the IBM Language Environment.

Chapter 15 of the Smart/RESTART Reference Manual provides detailed information about the SRS_DUMP and DUMPDD commands. The SRS_DUMP command controls what output the formatted dump should contain while the DUMPDD command specifies the DDname of the file to which the dump should be written.

The formatted dump includes the following information:

- **Registers and PSW at ABEND:**

```

SRSB37I - Abend code S0C7      System ID MP3A      SCP SP7.0.4      FMID HBB7707
.
SRSB34I -  EC PSW AT ABEND    078D3000 000084F8      ILC 06      IC 0007
.
SRSB34I -  ABEND REGS 0- 3    000087E0 00007B2D 000089F8 0001C1BC
SRSB34I -                4- 7    00000044 00000000 00000000 00076498
SRSB34I -                8-11   00007734 00008650 00008A78 00007CC0
SRSB34I -               12-15   00007714 0002B030 400084EC 8ABDA9A8
.
SRSB34I -  DATA AT PSW -10   83F905EF F223D150 A1F8FA20 D1508202
SRSB34I -                + 0   940FD150 F822D150 D150F332 A1F8D150
SRSB34I -                +10   5820D10C 07F25820 905C58F0 202C4110
.
SRSB34I -  CURRENT DSA + 0    00104001 0001B230 00081000 400084EC
SRSB34I -                +10   8ABDA9A8 000087E0 00007B2D 000089F8
SRSB34I -                +20   0001C1BC 00000044 00000000 00000000
SRSB34I -                +30   00076498 00007734 00008650 00008A78
SRSB34I -                +40   00007CC0 00007714

```

- **The application program CSECT and offset where the interrupt occurred:**

```

SRSB32I - Application Cobol CSECT SRS8CPT was in control at time ofabend.
SRSB35I - Interrupt occurred at + 0000EE0 into module SRS8CPT

```

- **Main COBOL control blocks -- TGTs**

- **IBM Language Environment Load List entries (CLLE)**

- **The System Diagnostic Work Area (SDWA)**

- **Various Smart/RESTART control blocks which describe each repositionable file, every application program module 'registered' with Smart/RESTART and the Smart/RESTART run-time options block. Repositionable file information appearing in the dump includes both file control blocks and the contents of the file's record buffers. Program module information includes both a formatted 'CSB' control block and the current contents of module storage to be saved at checkpoint time and restored at restart time.**

52.2 Smart/RESTART Inputs and Setup

Smart/RESTART runtime behavior is governed by various parameters specified in the RAINPUT file, in conjunction with a set of Smart/RESTART tables. These include Smart/RESTART's Table of Profile Defaults (which is always present) and optionally a Smart/RESTART Table of Shared defaults, Table of SQL_TRAP definitions and Table of Operating Shift Definitions. Smart/RESTART runtime behavior can be modified by user written exits and application calls to the Smart/RESTART profile modification service as well.

The Smart Jobstream Series Installation and Customization Guide provides detailed information about SQL_TRAP definitions (Chapter 5), defining Operating Shift Definitions (Chapter 6), defining Shared Defaults (Appendix B) and writing user exits (Appendix A).

52.2.1 RAINPUT control file

Smart/RESTART writes the contents of the RAINPUT file to its message file when the PRINT_CNTL parameter is set (or defaulted) to ON:

```
SRSI206I - Smart/RESTART instream control statements follow :
PLAN(SRS8CPT)          SYSTEM(DSN7)
CLEAR_CHKPT(AFTER)     CKPT_FREQ(1)          MSGDD(SRSRPRINT)
VERIFY_JID(NO)         MSG_DISPLAY(DEBUG)     ESTAE(ON)
REPOSITION(ALL)
RESTART_POINT(START)
SRS_DUMP(ON)
SRSI208I - Smart/RESTART control statements are completed
```

52.2.2 Run-time options block

Smart/RESTART writes out the complete set of profile options in effect at run-time to the message file when the MSG_DISPLAY option is set to DEBUG, as illustrated in the figure below. Note that profile parameters defined instream with the JCL override default values, while dynamic modifications requested through the Profile service take precedence over both.

```
16:45:23 RMCB001I - Formatting PRF:
ABDEL.00000000 ABEND.N ABENL.00000000 ABENN.00000005 ABP .Y ACRET.N ADCR .Y
AJID .J ARECO.Y AROLL.N ASMI .Y AUTA .N AUTO .Y
BBDW .I BSAM .W BULK .V CCSID.00000037 CBLKS.00000060 CDB2 .W CDB2#.00000000
CDBC .W CDBC#.00000000 CDPH.00000002 CET .00000000 CETA .I CKDDN.SRSCHECK CKENQ.Y
CKPTN.00000001 CKDYN.Y CLRC .N CRB .N CRS .F CVE .Y DATE .06/08/05
DAYS .11111110 DCBE .Y DCR .N DELF .SRSSKIP DMPDD.SRSDUMP DPC .SRSP810 DPS .N
DSPA .00000010 DSPN .00000100 EACT .Y EDITF.SRSEDIT END .00:00 ERASE.A ERCK .A
ERIN .A ERIO .R ERT .00000000 ERTA .W ESPIE.Y ESTAE.Y EXCKP.SRSXCKPT
EXINI.SRSXINIT EXPRF.SRSXPROF EXINF.SRSXFETC EXINR.SRSXREAD EXTRM.SRSXTERM FIIO .Y
GDGA .N GROUP. HLQ .DCA810.H HMLIM.00000000 HPACE.00000000 ID .SRS$TPD IEXIT.Y
INCF .R INCK .N INFL .W ISVC#.00000246 JOBID.RJSJ8CPT LFP .Y LGRDD.SRSLOGR
LMSPN.00001000 LNLIM.00000040 MDBRN.DBRMNAME MQMGR.CSQ1 MQQNM.SRS810.CHECKPOINT.QUEUE MRATE.D
MSGDD.SRSRPRINT NAME .SRS#TSD NDSN .A ODMG .N ODVR .W PLAN .SRS8CPT PRED .Y
PRI .00000001 PRST.Y PSAMP.00000001 PUCT .N RAMMX.00000020 RAMST.N RDMP .Y
RECOY.N RELIM.00000003 REOF .W REPOY.A REPR .N REPSY.A RESP .S
RF .00000000 RLCD1.00000911 RLCNT.00000003 RMI .SRSRDB2 RP .00000003 RSAM .Y RSVNM.RA17
RSTID.00000000 RTYP .A SAID .N SEC .00000001 SHFT .N SMDC . SMMC .
SMSC . SOUT .H SRLB .Y SSID . SUBP .N START.00:00 STAT .00000100
STRG .S SVCTY.E TIME .10.42 TRAP .Y TRCAL.N TRCIO.N UNIT .SYSDA
VERID.N VOL .RA0347 VSI .N VSO .R QELIM.00000200 QWAIT.00:01:00 WEC .N
WHC .N WRC .11 XMEN .N ZLRL .W
16:45:23 RMCB002I - PRF formatted successfully
```

Each profile operand in the preceding listing corresponds to a Smart/RESTART command. The operands appear in alphabetical order in the message listing. Similarly, the operands of the SRS#TSD macro (resident in the DCAMACS library) are defined in alphabetical order within the macro.

The profile operand named *MRATE* (appearing in bold, italicized and underlined text above) illustrates how to interpret the operand and its corresponding value. In searching the SRS#TSD macro for the string *MRATE*, we find the following:

```
&P.MRATE DC      CL1'&MSG_DISPLAY'  MSG_DISPLAY: Msg selection criteria
&P.MRATT EQU     C'T'                TERSE . Show only essential message
&P.MRATV EQU     C'V'                VERBOSE Show all but debug messages
&P.MRATD EQU     C'D'                DEBUG . Don't suppress any message
```

As such, the MRATE operand corresponds to the MSG_DISPLAY command while the value D denotes a setting of DEBUG. Therefore, the value of MSG_DISPLAY is DEBUG.

Shared profile

Smart/RESTART writes the following TERSE messages to identify the currently selected shared profile:

```
SRSI215I - Shared profile for job name RAI78CPT and program name RAIAPPL
SRSI215I - will be in effect between 00:00 and 00:00
```

Smart/RESTART then externalizes the shared profile's operands as described above. Note that the selection of a shared profile is specified via the **PROFILE(name)** command where *name* refers to a shared profile entry defined within the Smart/RESTART Table of Shared Defaults. Any PROFILE(name) command should be specified as the first parameter in the RAINPUT stream. Otherwise, profile options *preceding* the PROFILE(name) command will be overridden. Conversely, any runtime options specified *after* the PROFILE(name) command will override the corresponding options defined by the shared profile.

52.2.3 DB2 Plan name and Target subsystem

Verify the target DB2 subsystem and application plan name are both correct.

Note that the DB2 subsystem ID and DB2 application plan name can be overridden via the SYSTEM and PLAN commands respectively (specified in the RAINPUT stream). Chapter 4 of the Smart/RRSAF User Guide describes these commands in detail. In addition, you can specify the DB2 subsystem ID and application plan name via the JCL PARM string. Section 4.23 of the Smart/RRSAF User Guide describes the SCAN_JCL_PARM command used to accomplish this in detail.

52.2.4 Include the Smart/RESTART Package

Check the BIND for the DB2 application plan to ensure the BIND's package list includes the SRSDBRM package required by Smart/RESTART. Chapter 14 of the Smart/RESTART Reference Manual describes the COLLECTION and SET_PACKAGE commands which influence the search and selection of DB2 packages from appropriate package collections.

52.2.5 Check job setup

Check whether the job is set up properly for either an initial run (cold start) or restart with respect to considerations such as allocation and deletion files. When restarting multi-step jobs, ensure processing resumes from the appropriate step.

52.2.6 Run on a Licensed z/OS System

Ensure you are running on the proper z/OS system (for either an initial run or restart run) as only certain systems may be licensed for Smart/RESTART. Restarted jobs may run on different systems following a DB2 outage or other system failure for a variety of reasons that include Job class and Job redirection software.

The use of Smart/RESTART and Smart/RRSAF may be restricted to a particular CPU or group of CPUs. Contact your Smart Jobstream Series administrator to verify on which CPUs these products are licensed to run.

52.2.7 Error Prone Practices and Procedures

The Smart/RESTART publications document certain procedures which are inconsistent with successful restart (such as editing committed records). Other procedures may be problematic and difficult to use correctly even if they are strictly speaking 'valid'. Ensure that any new or unusual procedures used in the context of a restartable application are not at variance with your organization's guide to best practices.

52.2.8 Check for Correct DB2 BIND ENABLE Options

The application's DB2 plan must be bound with the correct ENABLE options in order to run successfully. Particularly if you are transitioning between CAF and RRSAF attachments, RAI recommends that you either specify ENABLE (*) or omit the ENABLE specification entirely as it will default to ENABLE (*). You can also explicitly specify ENABLE (DB2CALL) for CAF or ENABLE (RRSAF) for the RRSAF attachment.

52.3 Gathering additional Diagnostics

It may be necessary to gather additional diagnostic information in order to resolve a problem. When feasible, you should enable debug level messaging and request additional diagnostics. Then simply rerun and reproduce the problem. This is readily done during an application's development and testing phases but becomes more difficult once an application is deployed in production.

Even in production, however, you can take steps to gather additional diagnostics during a *restart* run. In addition, note that Smart/RESTART automatically enables debug level messaging during any restart attempt (starting with Smart/RESTART Version 7.10) and during abnormal termination (starting with Version 7.35).

52.3.1 MSG_DISPLAY(DEBUG)

RAI recommends that a debug level of messaging be requested when problems arise. This externalizes additional messages of an informational nature that Smart/RESTART issues during its processing. Specify MSG_DISPLAY(DEBUG) in the //RAINPUT stream to activate this option.

52.3.2 Trace Entry and Exit from 'registered' Modules

Smart/RESTART can trace the entry and exit of each discrete application module which calls the Smart/RESTART INITIALIZE and TERMINATE services (and in so doing 'register' themselves with Smart/RESTART). Additional messages identify the registered modules whose storage areas are saved during checkpoint processing. These trace messages are written to the Smart/RESTART message log when MSG_DISPLAY specifies DEBUG.

```
SRS476I - Application module SRS8CMB has been entered
SRS476I - Application module SRS8CMC has been entered
SRST044I - Application module SRS8CMC is in the process of exiting.
. . .
SRS483I - Working Storage saved for module SRS8CMA at CHECKPOINT 0000001
SRS483I - Working Storage saved for module SRS8CMB at CHECKPOINT 0000001
SRS483I - Working Storage saved for module SRS8CMC at CHECKPOINT 0000001
```

Specify CSECT_TRACE(ON) to obtain *all* these messages. Alternatively, specify CSECT_TRACE(MIN) to obtain information about work areas saved at checkpoint time but suppress entry and exit messages. Chapter 15 of the Smart/RESTART Reference describes this command in detail.

52.3.3 Function Trace

The FUNC_TRACE command enables tracing of application calls to Smart/RESTART services. These include calls for service through the Smart/RESTART API as well as I/O requests to repositionable files.

```
SRSB95I - INITIALIZE API Function request in progress.
. . .
SRS588I - OPEN API Function request issued from module SRS8CPT
. . .
SRS588I - CHECKPOINT API Function request issued from module SRS8CPT
. . .
SRS589I - WRITE file OUTFILE request issued from module SRS8CPT
SRS589I - WRITE file OUTFILE request issued from module SRS8CPT
```

I/O function tracing can be useful to resolve discrepancies between the number of records *actually* processed by Smart/RESTART versus the number of records you think were or should be processed. Use I/O tracing judiciously however, because Smart/RESTART writes DEBUG message SRS589I for each I/O request to every repositionable file. Chapter 15 of the Smart/RESTART Reference describes the FUNC_TRACE command in detail.

52.3.4 Check Operating shift Definitions

When debug level messaging is enabled, Smart/RESTART writes messages to describe each operating shift defined to it:

```
17:09:30 RMCB001I - Formatting SDE:
IDH .00000001 SCORE.00000001 HPACE.00000010 HTIME.00000000 HHMM .00:00 DAYS .11111110 FLAG1.00100000
FLAG2.10000000 FLAG3.00000000 FLAG4.00000000 FLAG5.10000000 NAME .T0000CPM100SID . GRPN .
JOBN . PGMN .
17:09:30 RMCB002I - SDE formatted successfully
```

Each operand in the listing above is documented in Chapter 6 of the Smart Jobstream Series Installation and Customization Guide. The operands appear in alphabetical order in the listing. Similarly, the operands of the SRS#OSD macro (resident in the DCAMACS library) are defined in alphabetical order within the macro.

In addition, note that Smart/RESTART identifies the currently active operating shift with the following TERSE message:

```
SRSC141I - New operating shift T0000CPD2 # 2 is in effect till 1800 hours
```

In this message, T000CPD2 denotes the name of the shift (specified via the SHIFT operand) while 2 is a unique, ordinal number that Smart/RESTART internally assigns to the shift.

52.3.5 Check SQL trap definitions

When a formatted dump is requested, Smart/RESTART externalizes the SQL trap definitions which are currently active. The following group of messages are written for each SQL trap entry:

```
SRS - SQL_TRAP Information Block (QIB): Address=0AC71AC0 Length=00000054
.
0AC71AC0 D8C9C240 00000000 0AC71A60 03AB0000 00000000 00000000 00000000 00000000 *QIB      G -      *
0AC71AE0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 F0F0F0F3 *          0003*
0AC71B00 F0F0F0C0 80000000 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF *000{      *
```

The SQL trap entry is mapped as follows:

Offset	Type	Length	Name	Description
0 (0)	Character	4	QIBID	Eye catcher
4 (4)	Address	4	QIBPREV@	Points to previous SQL trap entry
8 (8)	Address	4	QIBNEXT@	Points to next SQL trap entry
12 (C)	Binary	2	QIBSQLCD	SQLCODE in hex
60 (3C)	Zoned decimal	8	QIBQWAIT	Wait interval, in hhmmsssth
68 (44)	Bit	1	QIBFLAG1	Strategy: X'80' - automatic retry X'40' - trigger user ABEND X'20' - return control to application

52.3.6 SQL and RRSF tracing

Smart/RRSAF supports tracing of SQL calls, RRSF and CAF services, DB2 commands and/or IFI requests. You can gather this trace information for diagnostic purposes whether RRSF (Resource Recovery Services Attachment Facility) or CAF (Call Attach Facility) is used as the DB2 attachment option. Chapter 6 of the Smart/RRSAF User Guide describes how to obtain these traces.

52.3.7 Debugging with Smart/MONITOR

Smart/MONITOR is an ISPF dialog component of Smart/RESTART with numerous features that can be helpful in debugging problems with restartable applications. These facilities can be used in production as well as during an application's development and testing phases. These features include the ability to view application storage and display information associated with repositionable files. Chapter 30 of the Smart/RESTART User Guide discusses Smart/MONITOR facilities in some detail.

In addition, Smart/MONITOR provides several debugging and trace facilities for *self* diagnostics (i.e. debugging problems within Smart/MONITOR itself).

- The DEBUG command externalizes messages issued by the component which interfaces between Smart/RESTART and Smart/MONITOR. These messages provide useful diagnostics when Smart/MONITOR commands (like SUSPEND, RESUME, FORCE and QUIESCE) fail to effect their target restartable application.
- The TRACE command directs Smart/MONITOR to trace its own REXX source statements that are executed during a Smart/MONITOR session.

52.3.8 SRSUTIL and debugging

In certain instances, it may be useful to obtain a formatted dump of the checkpoint file. Smart/RESTART provides the SRSUTIL utility program for this purpose. RAI Support may ask you to run the SRSUTIL utility with PARM=DUMP to help debug errors within Smart/RESTART itself.

52.4 Operations with repositionable files

Smart/RESTART maintains control information for each repositionable file it manages. This information includes details such as the file's record format, block size, track and block number, block offset, logical record number and device type. Smart/RESTART uses this information to reposition the file should a rollback be requested or a restart occur. Repositioning errors result if these file characteristics change between a failure and subsequent repositioning attempt. To avoid repositioning errors, follow these guidelines:

- Avoid editing *committed* records of a repositionable file *between* runs. Doing so can **prevent** successful restart.

- When editing *uncommitted* records, the relative position of prior committed records within the file must be preserved. In other words, committed records must not be moved between blocks.
- Further, we would caution against modifying or removing any records (input or output) from sequential files (including IMS GSAM databases) after an abend. Even editing uncommitted input records can be problematic depending on how the input file was created.
- If a VB file is edited and the "PRESERVE ON" option is not used, trailing blanks will be truncated and re-blocking can occur."
- Should you need to copy a repositionable file to another dataset, make sure to preserve the original file's block structure. You can use a standard utility like IEBGENER to copy a *fixed blocked* dataset that ran out of space to a larger file.
- When copying *variable blocked* datasets or *fixed blocked* datasets that contain embedded *short blocks*, use the COPY function of the SRSUTIL utility instead of a standard utility like IEBGENER. This is because IEBGENER can shift logical records within a variable blocked dataset into different physical blocks. This in turn prevents successful restart when Smart/RESTART tries to reposition the file to a specific block and record. The SRSJUCPY member of the DCACNTL library contains illustrative JCL for the SRSUTIL COPY utility. Alternatively, you can use IEBGENER provided you specify RECFM=U for the SYSUT1 file referenced by IEBGENER.

Reblocking may not be obvious. But if reblocking does occur, it may cause records to be skipped at restart time without causing your job to abend. Should re-blocking occur or be unavoidable successful restart is still possible if the REPOSITION_REFRESH command is specified. REPOSITION_REFRESH directs Smart/RESTART to re-read all logical records through the last successful checkpoint, albeit with a significant performance penalty during RESTART or ROLLBACK processing. Chapter 12 of the Smart/RESTART Reference describes the REPOSITION_REFRESH command in detail.

Smart/LOGGER for VSAM

60.1 Introduction

Smart/LOGGER for VSAM automates application restart to reduce computer down time, shorten application restart times, and ensure data integrity for VSAM users. Smart/LOGGER for VSAM works with the Smart/RESTART and Smart/RRSAF products to provide automated restart capabilities for applications that access VSAM resources. In addition, Smart/LOGGER for VSAM, by utilizing VSAM Record Level Sharing, further improves data availability by allowing a batch job to share VSAM data with other batch jobs and CICS regions.

60.1.1 Key features of Smart/LOGGER

- Independent from CICS or any other transaction processing system.
- Provides journaling for updated, deleted, and inserted records in VSAM datasets (KSDS, ESDS, RRDS and VRRDS). Saves record images from which original data can be restored.
- Performs dynamic backout of all changes applied to VSAM datasets when ROLLBACK is requested or program restart is detected.
- Requires no changes to either application JCL or program source code.
- Allows for dynamic creation and allocation of log datasets.
- Eliminates the time-consuming and error-prone manual processing that can further complicate failure recovery and delay completion of required batch processing.
- Smart/LOGGER is language independent and operates with COBOL, PL/1, Assembler, C/C++ or any other programming language or code generating system.

NOTE: *In the current release, application language support is limited by the constraints imposed by the Smart/RESTART product.*

- All logging is performed *before* changes are applied to a dataset (the Smart/LOGGER write-ahead feature), thus guaranteeing dataset backout in the event of abnormal termination.

60.1.2 Key benefits

- Lets you restart abnormally terminated and incomplete jobs from the last checkpoint, thus eliminating the need to rerun jobs from the beginning. This saves CPU time and improves utilization of the batch window.
- Prevents data corruption and improves batch processing time.
- Improves data availability.
- Altered data records are logged in a VSAM Linear Data Set – the most efficient access method available under z/OS and OS/390. Logging is performed with speed comparable to virtual storage paging. This provides a substantial advantage in comparison with IMS or CICS logging.

60.1.3 Product synergy

Smart/LOGGER interoperates with both Smart/RESTART and Smart/RRSAF.

***NOTE:** The present version of Smart/LOGGER cannot operate independently. It is tightly coupled with and controlled by Smart/RESTART which acts as the application restart coordinator.*

60.2 Smart/LOGGER processing overview

Smart/LOGGER is a transparent facility that requires no application changes. During execution of your application program, Smart/LOGGER issues messages that describe its activity. In order to understand these messages, you should be familiar with the Smart/LOGGER processing flow.

60.2.1 Initialization process

After Smart/RESTART obtains control and performs its own initialization, it invokes a Smart/LOGGER initialization module, passing relevant start-up parameters. There are two distinct execution paths which Smart/LOGGER may follow. The execution path chosen depends upon the run mode: INITIAL or RESTART.

The Smart/LOGGER LDS is considered to be *consistent* if it contains no data records or if the last log action was a successful CKPT call. In this latter case, the last logical unit-of-Work (UOW) was completed and all changes to recoverable datasets active during this UOW were made permanent. Otherwise, the Log LDS is considered to be *inconsistent*.

The run mode is selected based upon the following criteria:

1. An INITIAL run is forced when CLEAR_CKPT(BEFORE) is specified in the RAINPUT stream. If CLEAR_CKPT(BEFORE) is not specified, then Smart/RESTART determines the run mode from its checkpoint dataset and indicates this run mode to Smart/LOGGER. During an INITIAL run, Smart/LOGGER overwrites the Log LDS -- irrespective of the status of the LDS (either consistent or inconsistent).

NOTE: *You must be extremely careful when requesting an initial run since a loss of data is possible. Changes logged to the LDS will be permanently lost. Thereafter, data can only be restored from previously secured image copies.*

2. A RESTART run is selected when Smart/RESTART determines the prior application run abended. This is the case when the Smart/RESTART checkpoint dataset indicates RESTART and CLEAR_CKPT(BEFORE) is not specified in the RAINPUT stream. During a RESTART run, Smart/LOGGER attempts to read the Log LDS in order to determine whether data backout is required. The determination to backout is based upon the following criteria in the specified order:

The Log LDS must be usable

- The Smart/LOGGER version must be the same as that registered in the Log LDS
- The LDS JOBID must match the Smart/RESTART JOBID
- The LDS must be **inconsistent** – backout is required to restore consistency.

When all of these conditions are met, Smart/LOGGER initiates a dynamic rollback procedure. If Smart/LOGGER successfully completes rollback, then all application data is restored in recoverable dataset(s) to their state as of the last successful checkpoint. In addition, the Log LDS is reinitialized and restored to a consistent state.

60.2.2 The Logging process

Smart/LOGGER remains dormant and does not interfere with application processing until the application issues an OPEN request. Smart/LOGGER obtains control at this point and checks whether the DDNAME being opened is specified as a RECOVERABLE file in the RAINPUT stream. If the file is defined as recoverable, then Smart/LOGGER installs its own I/O request intercepts and returns control to the application. Upon every subsequent I/O request directed to the recoverable file, Smart/LOGGER receives control and analyzes the action to be performed by the I/O request. Smart/LOGGER makes the following decisions:

- If a new record is to be inserted in the dataset, a record key or RRN is logged to allow a possible delete operation to reverse the insert.
- If a record is to be deleted from the dataset, then an image of the existing record is logged to permit an insert operation to reverse the delete.

- If a record is to be updated, then the before image of the record is logged so the update can be reversed and the original record restored.

In all cases, logging is performed ahead of the I/O operation to insure the original record image is secure before the I/O operation changes the recoverable file. This feature is called *write-ahead logging*.

60.2.3 Checkpoint process

Checkpoint processing is initiated when Smart/RESTART receives a CHECKPOINT or COMMIT request. Smart/LOGGER checkpoint processing is comprised of the following steps:

- For every recoverable dataset altered in the last unit-of-work, issue a CLOSE TYPE=T macro to flush all I/O buffers.
- Compute final checkpoint statistics: I/O counts, CPU times, etc. and advance the checkpoint counter.
- Write control information to the LDS.

After the checkpoint completes successfully, the Log LDS is in a consistent state.

60.2.4 Backout process

There are two phases to Smart/LOGGER backout processing:

- Using the Log LDS to rebuild the control blocks which correspond to each recoverable DDNAME.
- For each recoverable file, undo all logged changes in reverse order. For example, inserted records are deleted, deleted records are inserted, and updated records are restored to their original values.

Smart/LOGGER backout processing can be initiated by:

- A Smart/RESTART ROLLBACK request (Only the Phase 2 processing described above is executed).
- Upon a RESTART run. (Both Phase 1 and 2 processing described above are executed).
- By the Smart/LOGGER batch utility (SRLJBU). (Both Phase 1 and 2 processing described above are executed).

In all cases, the objective of backout is to undo all changes applied to recoverable datasets since the last checkpoint. When the backout procedure encounters an I/O error due to a duplicate or absent record, Smart/LOGGER issues a warning message but backout continues until complete. For example, such I/O errors can occur when the write ahead log contains a change that was not externalized to a recoverable dataset because an abnormal termination took place.

60.3 Batch Backout Utility (SRLJBU)

You can use the Smart/LOGGER Batch Backout Utility (program SRLJBU) in any of the following cases:

- When the application job abends and you wish to perform ‘standalone’ backout. You can make this choice to exercise better control over the backout process. You can take image copies of datasets needing backout and save the Log LDS. Stand-alone backout reduces backout complexity by isolating Smart/LOGGER from Smart/RESTART processing.
- Although unlikely, it is possible that when Smart/RESTART initiates checkpoint processing, the Smart/LOGGER checkpoint phase may complete successfully but phase II or III of Smart/RESTART checkpoint processing do not. In such a case, there is a mismatch between the Smart/RESTART checkpoint record and the Smart/LOGGER LDS. The SRLJBU utility can be used to force backout of the last completed unit-of-work to restore all resources managed by Smart/RESTART and Smart/LOGGER to a mutually consistent state.

NOTE: *This situation is also detected by Smart/RESTART during a restart run and is handled automatically -- without the need to submit a SRLJBU job.*

- Should the application abend before the first checkpoint is issued, then it is not possible for Smart/RESTART to restart the application. (This limitation is a deliberate Smart/RESTART design objective.) In such a case, an initial run is assumed upon restart. Therefore, you must use the SRLJBU utility to backout changes made during the incomplete unit-of-work before the application can be rerun.

JCL for the SRLJBU utility can be found in the SRLJBU member of the DCACNTL library. The following figure illustrates how to run the SRLJBU utility to backout an incomplete unit-of-work.

```

//          JOB
//*-----
//*          This job executes the Smart/LOGGER Batch Utility - SRLJBU
//*-----
//BACKOUT EXEC PGM=SRLBU
//STEPLIB DD DSN=DCAvrm.DCALOAD,DISP=SHR
//SRLJRNL DD DSN=RAI.SRL.SRLJRNL,DISP=SHR
//KSDD DD DSN=RAI.SRL.KSDS,DISP=SHR
//KSDD1 DD DSN=RAI.SRL.KSDSPATH,DISP=SHR

ICH70001I RAI028 LAST ACCESS AT 18:31:07 ON FRIDAY, MARCH 27, 1998
IEF236I ALLOC. FOR RAI2BUR BACKOUT
IEF237I 02D6 ALLOCATED TO STEPLIB
IEF237I 02D2 ALLOCATED TO
IEF237I 02D5 ALLOCATED TO SRLJRNL
IEF237I 02D3 ALLOCATED TO KSDD
IEF237I 02D3 ALLOCATED TO KSDD1
SRL500I - Smart / LOGGER Version 1 Release 1 Modification 0 Level A1
SRL501I - (c ) Relational Architects , Inc - All rights reserved.
SRL502I -
SRL503I - Job Name = RAI2BUR Step Name = BACKOUT Procstep Name =
SRL505I -
SRL018I - Job ID BATCH UTILITY is assigned to this jobstep.
SRL012I - Logger SRLJRNL DSN = SRL.SRLJRNL is open and ready for work
SRL243I - BATCH Utility run detected - attempt dynamic ROLLBACK.
SRL245I - LOG is consistent and synchronized with datasets
SRL230D - ROLB was issued by the process BATCH
SRL301D - KSDD is VSAM KSDS LRECL = 4096 CISCIZE = 4608 DSN = RAI.SRL.KSDS
SRL224D - KSDD FJB was rebuilt from a LOG record.
SRL301D - KSDD1 is VSAM KSDS LRECL = 4096 CISCIZE = 4608 DSN = RAI.SRL.KSDSPATH
SRL224D - KSDD1 FJB was rebuilt from a LOG record.
SRL247I - ROLLBACK ..Insert ..Delete ..Update .....IO ..AvgCPU ..TotCPU
SRL248I - KSDD.... 1 1 1 3 510 0
SRL248I - KSDD1... 0 0 0 0 0 0
SRL258D - Control information overwritten.
SRL242I - ROLB successfully completed.
SRL238I - Log records are synchronized with datasets being logged.
SRL252D - Control information updated.
SRL247I - CHECKPNT ..Insert ..Delete ..Update .....IO ..AvgCPU ..TotCPU
SRL248I - KSDD.... 0 0 0 0 510 0
SRL248I - KSDD1... 0 0 0 0 0 0
SRL249I - Checkpoint 00000002 established as a new point of consistency.
SRL255I - Smart / LOGGER termination is NORMAL..
SRL257I - Summary of all unit-of-works activities by DDNAME :
SRL247I - FINAL... ..Insert ..Delete ..Update .....IO ..AvgCPU ..TotCPU
SRL248I - KSDD.... 1 1 1 3 510 0
SRL248I - KSDD1... 0 0 0 0 0 0
SRL232D - All FJBs are freed.
SRL011I - Logger SRLJRNL was closed.
SRL233D - LSB is freed.
SRL215I - Exit processing completed.
IEF142I RAI2BUR BACKOUT - STEP WAS EXECUTED - COND CODE 0000

```

60.4 Smart/LOGGER Installation Verification Procedure (IVP)

The COBOL program SRS8CV is used to verify Smart/LOGGER installation and illustrate its functionality. The application source code resides in the SRS8CV member of the DCASAMP library. This section presents a brief description of the SRS8CV application. Section 60.4.3 provides criteria by which to identify a successful run of the SRS8CV program.

60.4.1 SRS8CV overview

At entry, SRS8CV expects files with the following DDNAMEs:

INFILE	- A sequential input driver file with 70 records
KSDSFILE	- An empty VSAM KSDS
RRDSFILE	- An empty VSAM RRDS

Upon exit:

INFILE	- An unchanged driver input file
KSDSFILE	- A KSDS file with 70 records
RRDSFILE	- An empty VSAM RRDS

SRS8CV execution flow:

1. The program reads records from the sequential file INFILE and inserts these records into the KSDSFILE. Since record keys are not in order, inserts are random. All work is divided into logical units-of-work which are delineated by COMMIT calls (which implicitly trigger Smart/RESTART and Smart/LOGGER checkpoint processing). After each Checkpoint completes, Smart/LOGGER issues statistics messages which show the number of I/O operations for each recoverable dataset in the last unit-of-work.
2. The KSDSFILE is read sequentially and the RRDSFILE is loaded.
3. The KSDSFILE is read sequentially. Records are deleted while the same records in the RRDSFILE are updated.
4. The program takes a deliberate S0C7 abend while processing the 18th unit-of-work.
5. The application restarts automatically (via the Smart/RESTART Automatic Restart Facility).
6. It is expected that SRS8CV will issue 18 physical and 72 logical checkpoints in an entire successful run.

60.4.2 Detailed Description of SRS8CV Processing

Let's take a closer look at application activity for every checkpoint.

KSDS Random Load - unit-of-work 1 - 7.

1. Open Input INFILE and Output KSDSFILE. Read INFILE Records until end of file is reached.
2. Set the KSDSFILE key using the value from the input record starting at column 53 for 10 bytes.
3. Insert a new record into the RRDSFILE using the key obtained in step 2.
4. If the INFILE record represents a new account, then trigger checkpoint processing by calling the SRSSAMPQ program. (Member SRSSAMPQ of the DCASAMP library contains the application source code.) Start a new logical unit-of-work.
5. Close the files INFILE and KSDSFILE.

Sequentially read KSDSFILE and randomly load RRDSFILE - unit-of-work 8 - 14.

1. Open files KSDSFILE and RRDSFILE for input and output respectively. Position file KSDSFILE to the very first record.
2. Read KSDSFILE records sequentially until end of file.
3. Set the RRDSFILE key using the value of the KSDSFILE record starting at column 1 for a length of 5 bytes.
4. Insert a new record into KSDSFILE using the key obtained in step 3.
5. If the KSDSFILE record is a new account record, then trigger checkpoint processing and start a new unit-of-work.
6. Close files KSDSFILE and RRDSFILE.

Sequentially read and delete records from RRDSFILE and randomly update the KSDSFILE - unit-of-work 15-21

1. Open files KSDSFILE and RRDSFILE for I/O processing. Point RRDSFILE to the very first record.
2. Sequentially read RRDSFILE records until end of file.
3. Set the KSDSFILE key using a substring of the RRDSFILE record starting at column 53 for a length of 10 bytes.
4. Randomly read a KSDSFILE record using the key value obtained in step 3.
5. Delete the current RRDSFILE record.

6. If a current record represents a new account, then trigger checkpoint processing and start a new unit-of-work.
7. Close files KSDSFILE and RRDSFILE.

60.4.3 SRS8CV verification of a successful run.

The JCL to run SRS8CV is tailored by the Smart/RESTART installation process and can be found in SRS8CV member of the DCAFILE dataset. The job should complete with a COMP CODE of 0. However, a *deliberate* SOC7 abend occurs during the run which is reflected in the job log. This is a *normal* condition. The following job log illustrates a successful run:

```

$HASP373 SRS8CV STARTED - INIT      5 - CLASS K - SYS RALP
IEF403I SRS8CV - STARTED
-JOBNAME  STEPNAME  PROCSTEP    RC   EXCP   CONN    TCB  SRB  CLOCK   SERV  PG  PAGE
-SRS8CV DEFINE          00    70    295    .00    .00    .0  8885     1  1    0
+SRSI035I - Smart / RESTART initialization in progress
+SRSI050I - Connection to RAI server RAI_SERVER successful.
+SRS082I - RAINPUT file parms will override defaults
+SRSC122I - Smart / RESTART error abends will be issued using user code 2005
+SRSI039I - MSGDD specified a log. Subsequent messages directed to file SRSPRINT
+
+SRS096E - Smart / RESTART ESTAE entered to flag JOB as abended; JOB will require
+a restart
+SRS168E - A SOC7 ABEND has occurred. Prepare to perform a restart
IEA995I SYMPTOM DUMP OUTPUT
SYSTEM COMPLETION CODE=0C7  REASON CODE=00000007
TIME=16.18.53  SEQ=00970  CPU=0000  ASID=002D
PSW AT TIME OF ERROR 078D3000  0000841A  ILC 6  INTC 07
ACTIVE LOAD MODULE      ADDRESS=00006CC0  OFFSET=0000175A
NAME=SRS8CV
DATA AT PSW 00008414 - FA33D3D0  D3D8F363  A2FFD3D0
GPR  0-3  000096A8  0000732E  00008402  0001E1BC
GPR  4-7  00009B98  50008D06  00009D88  00000000
GPR  8-11 00009D38  00006D70  00009DD8  000085DC
GPR 12-15 00006D40  00009548  50008F18  88BCAC68
END OF SYMPTOM DUMP
+SRS853I - Message file closed
+SRSA46I - Releasing all subpool 0, 1, 2 virtual storage
+SRSA73I - Subpool 002 was not allocated and needs no explicit release
+SRSA49I - Preparing for final transfer to application
+SRSA49I
+SRSI035I - Smart / RESTART initialization in progress
+SRSI050I - Connection to RAI server RAI_SERVER successful.
+SRS082I - RAINPUT file parms will override defaults
+SRSC122I - Smart / RESTART error abends will be issued using user code 2005
+SRSI039I - MSGDD specified a log. Subsequent messages directed to file SRSPRINT
IEC161I 056-084,SRS8CV,SRS8CV,KSDSFILE,,,
IEC161I RAI_XN.DCAVRM.SRSLAMP.KSDSFILE,
IEC161I RAI_XN.DCAVRM.SRSLAMP.KSDSFILE.DATA,
IEC161I SYS1.ICF.RA.UCAT1.VRA0002
IEC161I 056-084,SRS8CV,SRS8CV,KSDSFILE,,,

```

```

IEC161I  RAIXN.DCAVRM.SRLSAMP.KSDSFILE,
IEC161I  RAIXN.DCAVRM.SRLSAMP.KSDSFILE.INDEX,
IEC161I  SYS1.ICF.RA.UCAT1.VRA0002
IEC161I  062-086,SRS8CV,SRS8CV,KSDSFILE,,,
IEC161I  RAIXN.DCAVRM.SRLSAMP.KSDSFILE,
IEC161I  RAIXN.DCAVRM.SRLSAMP.KSDSFILE.DATA,
IEC161I  SYS1.ICF.RA.UCAT1.VRA0002
IEC161I  056-084,SRS8CV,SRS8CV,RRDSFILE,,,
IEC161I  RAIXN.DCAVRM.SRLSAMP.RRDSFILE,
IEC161I  RAIXN.DCAVRM.SRLSAMP.RRDSFILE.DATA,
IEC161I  SYS1.ICF.RA.UCAT1.VRA0002
IEC161I  062-086,SRS8CV,SRS8CV,RRDSFILE,,,
IEC161I  RAIXN.DCAVRM.SRLSAMP.RRDSFILE,
IEC161I  RAIXN.DCAVRM.SRLSAMP.RRDSFILE.DATA,
IEC161I  SYS1.ICF.RA.UCAT1.VRA0002

-SRS8CV SRS8CV      00  1745   6951   .04 .00 .4  240K  1      0  0  0   0
-SRS8CV PRINT       00    48    158   .00 .00 .0  4412  1      0  0  0   0
IEF404I SRS8CV - ENDED
-SRS8CV ENDED.  NAME-ALEX  TOTAL TCB CPU TIME=   .05  TOTAL ELAPSED TIME=   .5
$HASP395 SRS8CV ENDED

```

You can verify successful completion of the SRS8CV job by examining the output written to the file named SYSOUT. At the very **end** of the output, you should see the following:

```

SRS8CV017I -           Read      Read  Write write  Read  Write Delete
SRS8CV012I - 03005041 0018 0070  >0111  0070 >0041  >0041  0070 >0041  6582154883 9571160
SRS8CV012I - 03550000 0019 0070  >0121  0070 >0051  >0051  0070 >0051  9406530619 2109960
SRS8CV012I - 04089695 0020 0070  >0131  0070 >0061  >0061  0070 >0061  9866450856 4648760
SRS8CV006I -
SRS8CV006I - RRDS  Service end of file condition
SRS8CV012I - 04986786 0021 0070  >0140  0070 >0070  >0070  0070 >0070  9991556488 7120020
SRS8CV014I - KSDS  Service successfully closed
SRS8CV014I - RRDS  Service successfully closed
SRS8CV016I - Processing completed

```

60.5 Smart/LOGGER Customization

Smart/LOGGER software resides in the Smart Jobstream Series (SJS) libraries. Refer to the *SJS Installation and Customization Guide* (RAI publication SJS-001) for details about these libraries. Smart/LOGGER installation is complete once all distribution libraries are loaded and the Smart/LOGGER IVP is run. This section deals with issues concerning Smart/LOGGER Customization.

60.5.1 Table of System Defaults (SRL\$TSD)

You can update the defaults supplied by Relational Architects for Smart/LOGGER. To assemble and link edit a new Table of System Defaults for Smart/LOGGER (load module SRL\$TSD), first review and revise the values supplied in member SRL\$TSD of the DCACNTL library. The SRL\$TSD member is the source input to the assembler. Operands of the SRL\$TSD macro (which resides in the DCAMACS dataset) define various operational defaults for Smart/LOGGER.

Next, create a new Table of System Defaults load module for Smart/LOGGER using the SRLBTSD JCL member of the DCACNTL library. The low level qualifier of the Smart/LOGGER load library is DCALOAD.

Figure 60.1 illustrates member SRL\$TSD of the DCACNTL dataset. Operands of the SRL#TSD macro define Smart/LOGGER defaults.

```

-----
                                TITLE 'SRL$TSD - SRL TABLE OF PROFILE DEFAULTS'
*-----
*      Function
*      =====
*      Smart/LOGGER Table of run-time defaults
*
*-----
SRL$TSD SRL#TSD  DOC=NO,           <== Do not print macro definition  +
                                DDNAME=SRLJRNL,       <== DDNAME used as a log LDS      +
                                DSNAME=NONE,           <== DSNAME used as a log LDS      +
                                HLQ=RAI,               <== High Level Qualifier of Journal+
                                PRIMARY=1,            <== Primary allocation in units   +
                                SECONDRY=1,           <== Secondary allocation in units  +
                                MAXSPAN=1000,         <== Max no. of records in Journal  +
                                UNITS=TRK,            <== CYL/TRK/BLK                   +
                                REUSE=YES,            <== YES/NO - Journal dataset reuse +
                                TRACE1=NO,            <== YES/NO - Trace Entry/Exit Parm +
                                TRACE2=NO,            <== YES/NO - I/O Trace             +
                                SVC8=NO,              <== YES/NO - Intercept SVC 8       +
                                MSGSUP=D,             <== T/V/D - Message suppression   +
                                MSGDD=SRLPRINT,        <== Logger Message LOG DD name    +
                                LGDEL=YES,             <== Delete dynamic journal dataset +
                                LGFMT=NO,              <== Format journal dataset         +
                                CKPTID=-1,            <== Restart Checkpoint ID         +
                                VOLSER=(RA0001,RA0002)
PUNCH  ' IDENTIFY SRL$TSD(''SRL110#AA - TABLE OF SYSTEM DEFAULTS'')'
      END
-----
```

Figure 60.1 Smart/LOGGER Table of System Defaults

The following table describes the SRL#TSD macro parameters. You should not change any parameters that do not appear in this table.

<i>Parameter</i>	<i>Default</i>	<i>Description</i>
DDNAME	SRLJRNL	The DDNAME to which the Smart/LOGGER LDS will be allocated. If this DDNAME is not preallocated, then Smart/LOGGER dynamically creates and allocates a VSAM LDS.
DSNAME	NONE	The name assigned to the Smart/LOGGER LDS when the DDNAME (described above) is not specified in the application JCL and must be dynamically allocated. NONE directs Smart/LOGGER to create a unique name for every application run. (See section 60.5.2.2).
HLQ	RAI	The high level qualifier assigned to any dynamically created LDS. (See section 60.5.2.3).
PRIMARY	1	Primary allocation of the LOGGER LDS in space increments specified by the UNITS parameter
SECONDARY	1	Secondary allocation of the LOGGER LDS in space increments specified by the UNITS parameter
MAXSPAN	1000	The maximum number of journal records that can be logged for the current Logical Unit of Work. Please note that Smart/LOGGER triggers user abend 2005 with reason code 0000F002 if the number of journal records logged exceeds the MAXSPAN setting, and LGFMT=NO. (See section 60.5.1)
UNITS	TRK	LDS primary and secondary allocation units: TRK or CYL
REUSE	YES	Allow reuse (YES) or prevent reuse of the Logger LDS which contains valid information – whether checkpointed or not. When REUSE is set to YES, the Smart/LOGGER LDS will be overwritten upon an initial run, even if the LDS contains log records which were not checkpointed.
LGDEL	YES	Governs whether any dynamically allocated LDS will be deleted (YES) or kept (NO) upon successful completion of the application. If the program abnormally terminates, the Smart/LOGGER LDS is never deleted.
LGFMT	NO	Governs whether or not Smart/LOGGER formats journal dataset with high values in order to reserve all possible DASD space to be used by the journal. Please note that Smart/LOGGER triggers user abend 2005 with reason code 0000F003 if the number of journal records logged represents 80% of free space and LGFMT=YES.
VOLSER		Specifies a list of DASD volumes on which dynamically allocated VSAM LDS files can be created. You can specify up to 6 volumes.
MSGSUP	D	Message suppression level: D-Debug, V=Verbose, T=terse. This parameter is overridden by the Smart/RESTART RAINPUT parameter MSG_DISPLAY. See the Smart/RESTART Reference for details.
MSGDD	SRLPRINT	Defines the DDNAME to which Smart/LOGGER output messages are written. This parameter is overridden by the Smart/RESTART RAINPUT parameter MSGDD. See the Smart/RESTART Reference for details.

60.5.2 Logger Linear Dataset (LDS)

Smart/LOGGER uses a VSAM Linear Dataset (LDS) to save before images of records changed in recoverable datasets, i.e. VSAM datasets under Smart/LOGGER control. (VSAM LDS access is very efficient and is controlled by the MVS DIV macro.) The LDS can be predefined before the application executes or may be created dynamically at execution time.

60.5.2.1 LDS DASD space calculation.

The LDS must be of a size sufficient to store an entire unit-of-work, which includes all the records changed between two checkpoints. Note that the beginning of program execution and normal program termination are treated as implied checkpoints. VSAM datasets are allowed up to 116 data extents. Thus, to utilize DASD space more efficiently, a secondary allocation should be specified.

You can calculate the required LDS allocation as follows:

$$\text{total_bytes} = 300 + (512 * \text{number_of_files}) + \text{number_records} * \text{max_record_length}$$

where:

<code>total_bytes</code>	Total number of bytes for the VSAM LDS.
<code>number_of_files</code>	Number of VSAM files monitored by Smart/LOGGER
<code>number_of_records</code>	Maximum number of records changed in a logical unit of work.
<code>max_record_length</code>	Maximum record length

After `total_bytes` are computed, determine the number of 4K pages which need to be defined in the LDS:

$$\text{pages} = \text{total_bytes} / 4096$$

Assuming a 3390 DASD device is used (with a usable capacity of 48K per track), there will be 12 pages per track. The first page of the first track contains control information and should be excluded from the space computation. Therefore, an allocation specification of TRACK(1) will allow the LDS to have up to 11 usable pages. An allocation of TRACK(1 1) will allow the LDS to expand to up to 1427 pages (1428 / 12 = 119 extents including the primary allocation).

60.5.2.2 Manual definition of a VSAM LDS

Smart/LOGGER uses a VSAM linear dataset (LDS) as a repository for all changes applied to VSAM datasets under its control. You can create the VSAM LDS via JCL or IDCAMS and allocate this dataset to the DDNAME specified in the Smart/LOGGER Table of System Defaults (SRL\$TSD). Alternatively, you can let Smart/LOGGER dynamically allocate the VSAM LDS for you at execution time.

The following IDCAMS control statements can be used to allocate a VSAM LDS suitable for use by Smart/LOGGER. These control statements reside in member SRLJLDS of the DCACNTL library.

```
DEFINE CLUSTER (NAME(SRL.JOURNAL) -
                VOLUMES(WORK01,WORK02) -
                TRACKS(1,1) -
                SHAREOPTIONS(1,3) -
                LINEAR)
```

60.5.2.3 Dynamic creation of the VSAM LDS

If the VSAM LDS is not allocated in the application JCL, then Smart/LOGGER attempts to dynamically allocate the LDS at run time. An example of Smart/LOGGER dynamic allocation messages for the VSAM LDS appear below.

```
SRL013D - Logger DDNAME SRSLOGR is not specified in the JCL.
RID002D -
IDCAMS  SYSTEM SERVICES                                TIME: 14:13:24
RID002D -  DEFINE CLUSTER(NAME(RAI.RAI7SRLR.SRLSMP1.L1998091.C1413236  -
RID002D -  ) LINEAR VOLUMES( RA0001                                ) -
RID002D -  TRACKS      (00001 00001) REUSE      ) -
RID002D -  DATA (NAME(RAI.RAI7SRLR.SRLSMP1.L1998091.D1413236  ))
RID002D - IDC0508I DATA ALLOCATION STATUS FOR VOLUME RA0001 IS 0
RID002D - IDC0001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0
RID002D - IDC0002I IDCAMS PROCESSING COMPLETE. MAXIMUM CONDITION CODE WAS 0
SRL015I - Logger SRSLOGR DSN = RAI021.RAI7SRLR.SRLSMP1.L1998091.C1413236 was
successfully allocated
SRL018I - Job ID B03D7063CE5F1D01 is assigned to this jobstep.
SRL012I - Logger SRSLOGR DSN = RAI021.RAI7SRLR.SRLSMP1.L1998091.C1413236 is open
and ready for work
```

Figure 60.2 Messages issued upon LDS dynamic allocation

Smart/LOGGER generates the name of the VSAM LDS in accordance with the following schema:

hlq.jobname.stepname.Ldate.xtime

where

hlq	the High Level Qualifier defined by the HLQ parameter in SRL\$TSD (see section 60.5.1)
jobname	the job name
stepname	the job step name
date	the current date
x	A prefix which denotes either the Cluster 'C' or Data 'D' component of the LDS.
time	The current time

Parameters used by the LDS dynamic allocation routine are specified in the Smart/LOGGER Table of System Defaults (module SRL\$TSD described in Section 60.5.1). These parameters are: HLQ, PRIMARY, SECONDARY, UNITS, LGDEL, and VOLSER.

60.6 Messages and codes

Smart/LOGGER issues four levels of messages: types E-, W-, I-, and D-. Issuance of specific message levels is governed by the MSG_DISPLAY parameter in the RAINPUT stream. Alternatively, when Smart/LOGGER is used without Smart/RESTART, message levels are governed by the MSGDEST parameter of SRL@DFLT macro (see section 60.5.1).

This section documents the Smart/LOGGER E- and W- level messages only. E-level messages denote severe errors which are issued when further processing is impossible. This may happen when an abnormal termination occurs within Smart/LOGGER or when processing is logically impossible due such conditions as errors in specified parameters, wrong or inconsistent datasets, I/O errors, etc.

W-level messages denote warnings and signify an unusual or error prone execution path. They can also be used as guides to diagnose processing errors that may eventually culminate with an E-level message.

E- and W-level messages cannot be suppressed by the MSG_DISPLAY command. They are always issued.

I-level messages are informational in nature. They are self explanatory and require no user action. I-level messages are not documented. I-level messages are issued when the MSG_DISPLAY parameter is set to either VERBOSE or DEBUG.

Lastly, D-level messages are issued for a debugging purposes. D-level messages are obtained by setting the MSG_DISPLAY parameter to DEBUG (which should only be done at the direction of RAI technical support). D-level messages are not documented.

Many messages contain variable values (designated by && characters), whose meaning is described in the following table:

&&DDNAME	Refers to the name of a recoverable file controlled by Smart/LOGGER.
&&LOGRDD	Identifies the DDNAME to which the Log LDS is allocated.
&&DIVFUNC	Identifies a Data in Virtual (DIV) function. One of the following functions can be reported: IDENTIFY, ACCESS, MAP, UNMAP, SAVE, UNACCESS, UNIDENTIFY.
&&DIVRC	DIV error return code. Refer to <i>MVS Programming: Assembler Services Reference</i> , GC28-1910, DIV macro description.
&&DIVRRC	DIV error reason code. Refer to <i>MVS Programming: Assembler Services Reference</i> , GC28-1910, DIV macro description.

**SRL010E - Logger &&LOGRDD error in DIV function &&DIVFUNC,
RC=&&DIVRC Reason=&&DIVRRC**

An error occurred when executing the Data in Virtual function &&DIVFUNC while accessing the Log LDS. &&DIVRC and &&DIVRRC refer to the MVS DIV macro return and reason codes respectively. When &&DIVRC = 8 and &&DIVRRC = 00E10802, then the Log LDS is out of space. In that case, reallocate the Log LDS specifying greater secondary allocation or, if disk space was depleted, move the LDS to a different DASD volume. All other errors should be reported to RAI technical support.

SRL014E - Attempt to dynamically allocate LOGGER file failed.

Dynamic allocation of the Log LDS failed. In the job log, locate the messages which indicate the reason for failure. One possible reason is insufficient space. Correct the problem and rerun the job.

**SRL017W - Logger &&LOGRDD was not freed and DSN=&&LOGRDSN
was not deleted.**

A Log LDS was dynamically allocated but the job failed to completed successfully. This message warns the user to use the dynamically allocated Log LDS for backout processing. This dataset should not be deleted until backout is successfully completed.

SRL019W - Logger &&LOGRDD was not deleted due to previous errors.

An LDS was dynamically allocated. Processing was successful but the LDS was not deleted due to IDCAMS errors. Review the job log searching for IDCAMS error messages. Try to identify and correct the error by examining system message IEC161I.

SRL205E - &&DDNAME open failed.

The application file allocated to the DDname &&DDNAME failed to open. This is not a fatal error. Rather, it is up to the application to determine whether to continue processing or to abnormally terminate. The application can issue a userabend to allow subsequent restart processing. Smart/LOGGER will issue diagnostic messages that identify the error code associated with the failed open.

SRL206W - &&DDNAME close failed - file still open.

An application issued a CLOSE macro for the file named &&DDNAME which failed. Examine the system log for additional error messages to identify the reason for failure.

SRL208W - &&DDNAME was already open when an OPEN request was received.

The application issued an OPEN for the file named &&DDNAME which was already in the open state. Examine the program logic to identify the reason for a second open.

SRL211E - &&DDNAME failed to create FJB - Program logic error.

This is a Smart/LOGGER logic error. One reason is a lack of virtual storage in the application address space. Increase the REGION parameter and rerun the job. Report the problem to RAI technical support.

SRL212E - &&DDNAME FJB not found - Program logic error.

This is a Smart/LOGGER logic error. Collect complete job output and e-mail or FAX it to RAI technical support.

SRL222E - &&DDNAME VSAM &&EXIT exit entered upon GET request.

An I/O error occurred when Smart/LOGGER issued a GET request to obtain a record image before a change. This is a Smart/LOGGER logic error. Collect complete job output and e-mail or FAX it to RAI technical support.

SRL225E - &&DDNAME FJB was not found - logic error, abend w/S0C1.

This is a Smart/LOGGER logic error. Collect complete job output and e-mail or FAX it to RAI technical support.

SRL226E - &&DDNAME ROLB unrecoverable error - run was aborted.

Examine messages preceding this error message. Try to resolve the problem based upon this information. If unable to resolve the problem, then collect complete job output and e-mail or FAX it to RAI technical support.

SRL229W - Smart/LOGGER Abnormal Termination is in progress.

Smart/LOGGER detected an abnormal termination -- this job will require restart processing with the backout of uncommitted changes.

SRL231E - &&DDNAME is not allocated but is required for ROLB.

The Smart/LOGGER rollback process determined that the file named &&DDNAME is not allocated in the JCL. The &&DDNAME was specified in the RECOVER parameter of the RAINPUT stream and must be allocated upon a restart run, even if there was no update activity detected for this dataset. Correct the JCL to specify the missing &&DDNAME and rerun the job.

**SRL240E - LOG is inconsistent but RESTART was not requested.
Data may be lost or corrupted. Run will be aborted.**

The Log LDS was found to be inconsistent (there was no CKPT issued for the last unit-of-work). Records which were not checkpointed must be backed out to preserve the integrity of the last unit-of-work. However, Smart/RESTART specified an initial run. If the decision to forego a backout was deliberate, you must delete and define a new Log

LDS to clear all the records from the LDS. Otherwise Smart/LOGGER will refuse to run to prevent accidental and permanent data loss.

SRL241E - ROLB failed. Examine preceding messages to identify the cause of error, correct the error and rerun the job.

This error is issued in the event that backout processing itself fails. This can happen, for example, when inserting records into a recoverable dataset and there is no more space available on the volume for the recoverable file. In this case, move the recoverable file to a volume with sufficient space, run the SRLJBU utility to backout the incomplete unit-of-work or simply restart the job and allow Smart/RESTART to backout uncommitted changes automatically.

SRL254W - Initial run is forced by RunTime parameters - LOG dataset will be erased.

Smart/RESTART indicated an initial run and Smart/LOGGER bypassed its examination of a Log LDS which may be inconsistent. All LDS data records are lost. It is the user's responsibility to ensure that an initial run was requested deliberately.

SRL260W - Last Smart/RESTART CKPT call failed while LOG was successfully CKPT - LOG ROLB is forced.

A Smart/LOGGER checkpoint completed successfully but phase II or phase III of Smart/RESTART checkpoint processing failed to complete successfully. The last successfully checkpointed unit-of-work on the Log LDS will be forcefully backed out in order to synchronize the Log LDS with the Smart/RESTART checkpoint dataset. Processing will continue.

SRL261W - Smart/RESTART CKPT differs from Smart/LOGGER by more than one

There is a mismatch between the Smart/RESTART Checkpoint dataset and the corresponding Smart/LOGGER LDS. The Log LDS remains unaltered and processing is permitted to continue. This message is a precursor to further error messages which may cause processing to terminate abnormally.

SRL262E - Smart/RESTART JOBID is not the same as Smart/LOGGER. LOG dataset does not belong to this job.

There is a mismatch between the Smart/RESTART Checkpoint dataset and the Smart/LOGGER LDS since the registered JOBID's are not the same. The specified Checkpoint data set and Log LDS were created by different jobs. Locate a correct Log LDS corresponding to the specified Checkpoint file and rerun this job.

SRL264W - &&DDNAME failed to reposition at &&TEXT &&KEY

During a RESTART run and following successful backout, Smart/LOGGER issued a POINT macro to reposition the recoverable file named &&DDNAME. The POINT operation failed. The &&text variable assumes a value of either KEY or RRN while the value of &&key represents either the KEY or RRN value. Look for a preceding error message that displays an error code.

SRL266W - &&DDNAME VERIFY REFRESH failed.

The file named &&DDNAME was opened with a return code of 118 which caused Smart/LOGGER to issue a VSAM VERIFY macro. This VERIFY request failed. Execution will continue. Examine the preceding message to identify the VSAM VERIFY error code.

Appendix A

Smart/RESTART Interoperation

with Other Products

See appendix I of the Smart Jobstream Series Installation and Customization Guide for detailed information on Smart/RESTART Interoperation with Other Products.

Smart/RESTART Data Structures

Smart/RESTART provides a set of data structures with which your application can request services and Smart/RESTART can return feedback to your program. Smart/RESTART provides these structures as members of its DCASAMP dataset. You can include these structures in your application's working storage through your compiler's COPY facility or an equivalent function. Alternatively, you can explicitly code these structures yourself.

The COPY members are named in accordance with the following convention:

- All COPY member names start with the Smart/RESTART component prefix SRS.
- The fourth character of the member name denotes the **host language** in which the structure is embedded or expanded. For example, C denotes a COBOL copy member. Similarly, P denotes PL/1, A stands for Assembler Language and H designates a C header file.

The Smart/RESTART data structures are as follows:

- The Smart/RESTART service names are defined by an SRSxAPI structure
- The Function Communications Area or FCA is defined by an SRSxFCA structure
- A File Control Specification (or FCS) is defined by an SRSxFCS structure. An FCS must be defined for each OS file or VSAM dataset whose I/O is performed by calling the Smart/RESTART I/O service routines.

The Smart/RESTART Reference provides detailed descriptions of these data structures and the fields which comprise them. This appendix simply lists the three structures in each of the languages Smart/RESTART supports:

- | | |
|----------------------|---|
| • Assembler language | DCASAMP members whose names start with SRS@ |
| • COBOL | DCASAMP members whose names start with SRSC |
| • C | DCASAMP members whose names start with SRSH |
| • PL/1 | DCASAMP members whose names start with SRSP |

B.1 Assembler Language DSECTs

```

*-----
*
*      NAME
*      ====
*
*      SRS@API - SMART/RESTART Service names
*
RAI_SERVICES      DS      0D      Service names
RAI_ABEND         DC      CL12'ABEND'
RAI_ALLOCATE      DC      CL12'ALLOCATE'
RAI_CANCEL        DC      CL12'CANCEL'
RAI_CHECKPOINT    DC      CL12'CHECKPOINT'
RAI_CKPT_IMMED    DC      CL12'CKPT_IMMED'
RAI_CLOSE         DC      CL12'CLOSE'
RAI_CURSOR_HOLD   DC      CL12'CURSOR_HOLD'
RAI_END_OF_JOB    DC      CL12'END_OF_JOB'
RAI_ENVCHECK      DC      CL12'ENVCHECK'
RAI_FILE_INFO     DC      CL12'FILE_INFO'
RAI_FREE          DC      CL12'FREE'
RAI_FSOPEN        DC      CL12'FSOPEN'
RAI_INITIALIZE    DC      CL12'INITIALIZE'
RAI_MESSAGE_LOG   DC      CL12'MESSAGE_LOG'
RAI_OPEN          DC      CL12'OPEN'
RAI_PROFILE       DC      CL12'PROFILE'
RAI_READ          DC      CL12'READ'
RAI_ROLLBACK      DC      CL12'ROLLBACK'
RAI_ROLLIMS       DC      CL12'ROLLIMS'
RAI_SNAP          DC      CL12'SNAP'
RAI_STOP_RUN      DC      CL12'STOP_RUN'
RAI_SQLCA         DC      CL12'SQLCA'
RAI_START         DC      CL12'START'
RAI_TERMINATE     DC      CL12'TERMINATE'
RAI_WRITE         DC      CL12'WRITE'
RAI_RAIAPI        DC      CL8'RAIAPI'
RAI_RESTART       DC      CL12'RESTART'
RAI_ENVLOAD       DC      CL12'ENVLOAD'
RAI_STORAGE_REST  DC      CL12'STORAGE_REST'
*
*
*      End of Smart/RESTART service names
*
RAI_SERVICESLEN EQU      *-RAI_SERVICES

*-----
*
*      NAME
*      ====
*
*      SRS@FCA - SMART RESTART FUNCTION COMMUNICATIONS AREA
*
*-----
FCA      DSECT ,      FUNCTION COMMUNICATIONS AREA
.COMMON  ANOP
*
*      User function communications area mapping
*
FCASTAT  DS      CL4      API Request completion status
*      '      '      -> API Function successful
*      'REST'      -> Restart is in progress
*      'ROLL'      -> ROLLBACK to CKPT API occurred
*      'EC00'      -> External COMMIT successful
*      'EC08'      -> External COMMIT failed
*
FCARETC  DS      H      API Function return code - binary
FCAERROR DS      CL8      Extended error information
FUNCTION DS      CL12     API Function request in EBCDIC
FCAABCDE DS      H      User abend code in binary

```

*			
FCADMPDP DS	CL1	User abend dump option	
FCADMPDF EQU	X'F0'	- Don't print dump	
FCADMPON EQU	X'F1'	- Print dump	
*			
FCACTERM DS	CL1	Subprogram terminate request option	
FCACKEEP EQU	X'F0'	- Don't free csb	
FCACSTOP EQU	X'F1'	- Free CSB	
*			
FCACINIT DS	CL1	Subprogram initial option for GOBACK	
FCALEAVE EQU	X'F0'	- Working storage not cleared	
FCACLEAR EQU	X'F1'	- Working storage cleared (REUS)	
*			
FCAREDEF DS	CL1	Subprogram INIT API UAB rebuild req	
FCAREDOF EQU	X'F0'	- Calculate UAB extents first init	
FCAREDON EQU	X'F1'	- Recalculate UAB extents each init	
*			
FCADSTKW DS	CL4	DST eyecatcher	
FCADST@ DS	F	-> DST	
*			
FCACSNAM DS	CL8	CSECT name, set by precompiler	
*			
FCACLID DS	CL1	CSECT language ID set by precompiler	
FCACLCBV EQU	C'V'	- OS/VS COBOL	
FCACLCB2 EQU	C'2'	- VS COBOL 2	
FCACLPL1 EQU	C'P'	- PL/1	
FCACLC EQU	C'C'	- C/370	
FCACLA EQU	C'A'	- 370 ASSEMBLER	
*			
FCAXLAPI DS	CL1	At initialize requests func xlation	
FCAXLIMS EQU	C'I'	- Precompiler IMS environment	
FCAXLIDM EQU	C'M'	- Precompiler IDMS environment	
FCAXLDB2 EQU	C'D'	- Precompiler DB2 environment	
*			
FCASPRF3 DS	CL1	reserved for precompiler option flag	
FCASPRF4 DS	CL1	reserved for precompiler option flag	
*			
FCATRSMD DS	CL1	API mode	
FCATRSR EQU	C'P'	- precompiler transparency	
FCATRSM1 EQU	C'M'	- manual api mode	
FCATRSM2 EQU	C' '	- manual api mode	
FCATRSM3 EQU	X'00'	- manual api mode	
FCATRSOB EQU	C'O'	- object transparency mode	
FCATRSAR EQU	C'A'	- Automatic RAM addr/len calculation	
*P.TRSTL EQU	C'S' (reserved)	- STEALTH mode of operations	
*			
FCAHREOP DS	CL1	Precomp CSECT host opens at restart	
FCAHRON EQU	C'O'	- reopen all files defined in module	
FCAHROF1 EQU	C'N'	- no precompiler restart reopens	
FCAHROF2 EQU	C' '	- no precompiler restart reopens	
FCAHROF3 EQU	X'00'	- no precompiler restart reopens	
*			
FCACOMIT DS	CL1	Commit feedback status	
FCACOMPH EQU	C'P'	- physical commit has completed	
FCACOMLO EQU	C'L'	- logical commit has completed	
*			
FCACW DS	CL1	Checkpoint data verify at RESTART	
FCACWVOK EQU	C' '	- Verify completed successfully	
FCACWVER EQU	C'V'	- CKPT data verify mismatch	
*			
FCANP DS	XL1	RAIAPI deflection flag	
FCANPACT EQU	X'77'	- Deflect all RAIAP calls	
FCANPOFF EQU	X'78'	- Perform RAIAP function	
*			
FCARAMST DS	CL1	RAM initialize style	
FCARAMSN EQU	C'N'	- Non-initialized (default value)	
FCARAMSL EQU	C'L'	- Range: Start / Size pairs	
FCARAMSE EQU	C'E'	- Range: Start / End pairs	
*			
FCAPCVRM DS	H	Precompiler Version,Release,Mod.	
FCAMQCHN DS	F	MQSeries connection handle is set	
*			
		either by user, prior to INITIALIZE	
		when QMGR is UNKNOWN, or by	
		Smart/RESTART	
*			

```

*
FCACKPTN DS      H      Logical CKPTs per physical CKPT
FCACKCNT DS      H      CKPT calls since last physical CKPT
FCAPRETC DS      H      User program return code
*
          DS      H      *** reserved ***
*
*
*      Do#ag
*
*      Use the two counters below whenever the checkpoint pacing
*      criteria was specified via CKPT_PACE instead of CKPT_FREQ
*
FCAHPACE DS      F      Logical CKPTs per physical CKPT
FCATPACE DS      F      CKPT calls since last physical CKPT
*
*      end#ag
*
FCASPAR3 DS      CL8      *** spare ***
*
FCALEN   EQU     *-FCA      FCA STRUCTURE LENGTH

```

```

* -----
*
*      SMART/RESTART
*      FILE CONTROL STRUCTURE
*
*      MAIN CONTROL BLOCK USED TO HOUSE FILE RELATED STATUS IN THE
*      USERS MEMORY.
*
* -----
FCS        DSECT ,      FILE CONTROL STRUCTURE
FCSDDNAM DS      CL8      File's DDname
FCSMODE DS      CL8      Intent - 'input' or 'output'
FCSAMTYP DS      CL1      Access method indic - V=VSAM S=BSAM
FCSSTAT DS      CL1      FCS status code - character
FCSOK EQU      C'0'      . read/write performed
FCSEOD EQU      C'1'      . end of data reached
FCSIOERR EQU      C'2'      . I/O error occurred
FCSNOTAL EQU      C'3'      . DDname not allocated
FCSKEYOK EQU      C'4'      . Specified start key is valid
FCSKEYIV EQU      C'5'      . Specified start key invalid
*
FCSLRECL DS      H      LRECL
FCSBLKSZ DS      H      BLKSIZE
*
FCSRECFM DS      CL3      RECFM in EBCDIC (F,FB,V,VB and VBS)
FCSRECF EQU      C'F'      . fixed
FCSRECV EQU      C'V'      . variable
FCSRECU EQU      C'U'      . undefined
FCSRECD EQU      C'?'      . ascii (not supported yet)
FCSRECB EQU      C'B'      . blocked
FCSRECZA EQU      C'A'      . asa
FCSRECZM EQU      C'M'      . machine
FCSRECSB EQU      C'S'      . spanned
*
FCSWADVO DS      CL1      Write after advancing option
FCSWADV EQU      C'A'      . Write after advancing in progress
*
FCSWDIR DS      CL1      Write after advancing direction
FCSWAFTE EQU      C'A'      . Apply amount to 'after' for write
FCSWBEO EQU      C'B'      . Apply amount to 'before' for write
*
FCSWUNIT DS      CL1      Write after advancing amount units
FCSWLINE EQU      C'L'      . Amount units = lines
FCSWCHAN EQU      C'C'      . Amount units = channels
*
FCSWCMIX DS      CL1      Write after/before mixture for file
FCSWCASA EQU      C'A'      . Use ASA
FCSWCMCH EQU      C'M'      . Use MCH
*
FCSdtype DS      CL1      Dummy file discrimination flag
FCSDNORM EQU      C'N'      . standard file

```

```

FCSDDUMY EQU C'D' . dummy file
*
FCSWRAMT DS H Write advancing amount
* 0-255 for lines or channels
*
FCSCSEQ# DS H concatenation seq num of current dsn
FCSVSEQ# DS H volume sequence num for current dsn
*
FCSLEVEL DS H FCS Version Release Mod. Level
FCSLEVEL_CURRENT EQU FCSLEVEL_V720 Latest FCS level
FCSLEVEL_FIRST EQU FCSLEVEL_V712 First introduced FCS level
FCSLEVEL_V720 EQU 720 FCS level 720
FCSLEVEL_V712 EQU 712 FCS level 712
*
FCSRESRV DS CL2 *** spare ***
*
FCSRECID DS OCL8 Combination of TTRZ or RBA and BUFX
* and VOLSEQ in order to uniquely
* position to a logical record
FCSIDRBA DS CL4
FCSIDBFX DS CL2
FCSIDVNM DS CL2
*
* Fields governing dynamic allocation
*
DS OD
FCSDSN DS CL44 Data set name
FCSMBR DS CL8 Member name
FCSUNIT DS CL8 Unit
FCSVOL DS CL6 VOLSER
FCSPRIM DS H primary allocation
FCSSEC DS H secondary allocation
FCSRETPD DS H retention period (tape label)
FCSDSSEQ DS H data set sequence number, tape label
*
FCSLABEL DS XL1 Label type
FCSLBNL EQU X'01' . NL
FCSLBSL EQU X'02' . SL
FCSLBNSL EQU X'03' . NSL
FCSLBBLP EQU X'04' . BLP
*
FCSALLUN DS CL1 Allocation units
FCSALCYL EQU C'C' . cylinders
FCSALTRK EQU C'T' . tracks
FCSALBLK EQU C'B' . blocks - not yet used
*
FCSDISPS DS XL1 Initial disposition status
FCSDSOLD EQU X'F1' -> X'01' . old
FCSDSMOD EQU X'F2' -> X'02' . mod
FCSDSNEW EQU X'F3' -> X'04' . new
FCSDSSHR EQU X'F4' -> X'08' . shr
*
FCSDISPN DS XL1 Normal disposition
FCSDISPA DS XL1 Abnormal disposition
FCSDSUNC EQU X'F1' -> X'01' . uncatlg
FCSDSCAT EQU X'F2' -> X'02' . catlg
FCSDSDEL EQU X'F3' -> X'04' . delete
FCSDSKEE EQU X'F4' -> X'08' . keep
*
FCSLEN712 EQU *-FCS FCS structure length at V712
*
FCSRSTYL DS CL1 Repositioning style (output field)
FCSRSTAF EQU C'A' . After committed record
FCSRSTAT EQU C'T' . At committed record (re-read)
FCSRSTNO EQU C'N' . File is not repositionable
*
FCSEBLKSI8 DS OXL8 Block size in eight bytes
DS F High order half (reserved)
FCSEBLKSI DS F Block size - fullword
*
DS 2F Reserved
*
FCSLEN EQU *-FCS FCS structure length
FCSLEN720 EQU *-FCS FCS structure length at V720

```

B.2 COBOL Copy Members

```

*-----
*
*   SMART/RESTART SERVICE NAMES FOR COBOL
*
*-----
Ø1  RAI-SERVICES.
    05  RAI-ABEND          PIC X(12)  VALUE 'ABEND'.
    05  RAI-ALLOCATE      PIC X(12)  VALUE 'ALLOCATE'.
    05  RAI-CANCEL        PIC X(12)  VALUE 'CANCEL'.
    05  RAI-CHECKPOINT    PIC X(12)  VALUE 'CHECKPOINT'.
    05  RAI-CKPT-IMMED    PIC X(12)  VALUE 'CKPT_IMMED'.
    05  RAI-CLOSE         PIC X(12)  VALUE 'CLOSE'.
    05  RAI-CURSOR-HOLD   PIC X(12)  VALUE 'CURSOR_HOLD'.
    05  RAI-END-OF-JOB    PIC X(12)  VALUE 'END_OF_JOB'.
    05  RAI-ENVCHECK      PIC X(12)  VALUE 'ENVCHECK'.
    05  RAI-FILE-INFO     PIC X(12)  VALUE 'FILE_INFO'.
    05  RAI-FREE          PIC X(12)  VALUE 'FREE'.
    05  RAI-FSOPEN        PIC X(12)  VALUE 'FSOPEN'.
    05  RAI-INITIALIZE    PIC X(12)  VALUE 'INITIALIZE'.
    05  RAI-MESSAGE-LOG   PIC X(12)  VALUE 'MESSAGE_LOG'.
    05  RAI-OPEN          PIC X(12)  VALUE 'OPEN'.
    05  RAI-PROFILE       PIC X(12)  VALUE 'PROFILE'.
    05  RAI-READ          PIC X(12)  VALUE 'READ'.
    05  RAI-ROLLBACK      PIC X(12)  VALUE 'ROLLBACK'.
    05  RAI-ROLLIMS       PIC X(12)  VALUE 'ROLLIMS'.
    05  RAI-SNAP          PIC X(12)  VALUE 'SNAP'.
    05  RAI-STOP-RUN      PIC X(12)  VALUE 'STOP_RUN'.
    05  RAI-SQLCA         PIC X(12)  VALUE 'SQLCA'.
    05  RAI-START         PIC X(12)  VALUE 'START'.
    05  RAI-TERMINATE     PIC X(12)  VALUE 'TERMINATE'.
    05  RAI-WRITE         PIC X(12)  VALUE 'WRITE'.
    05  RAI-RAIAPI        PIC X(08)  VALUE 'RAIAPI'.
    05  RAI-RESTART       PIC X(12)  VALUE 'RESTART'.
    05  RAI-ENVLOAD       PIC X(12)  VALUE 'ENVLOAD'.
    05  RAI-STORAGE-REST  PIC X(12)  VALUE 'STORAGE-REST'.
*--  END OF RAI SERVICE NAMES STRUCTURE -----

```

```

*-----
*
*   SMART/RESTART
*
*   FUNCTION COMMUNICATIONS AREA
*
*-----
Ø1  FCA.
    05  FCA-STATUS        PIC X(4).
        88  FCA-NORMAL      VALUE SPACES.
        88  FCA-RESTART     VALUE 'REST'.
        88  FCA-ROLLBACK    VALUE 'ROLL'.
        88  FCA-EXT-COMMIT-OK VALUE 'EC00'.
        88  FCA-EXT-COMMIT-ERR VALUE 'EC08'.
        88  FCA-INITIAL-RUN  VALUE SPACES.
        88  FCA-RESTART-RUN  VALUE 'REST'.
    05  FCA-RETURN-CODE    PIC S9(4)  VALUE +0  COMP.
    05  FCA-EXTENDED-ERROR-MSG PIC X(8).
    05  FCA-LAST-FUNCTION  PIC X(12).
    05  FCA-USER-ABEND-CODE PIC S9(4)  VALUE +0  COMP.
    05  FCA-DUMP-OPTIONS   PIC 9(1)   VALUE 0.
        88  FCA-DUMP-NOPRINT VALUE 0.
        88  FCA-DUMP-PRINT  VALUE 1.
    05  FCA-TERM-OPTIONS   PIC 9(1)   VALUE 0.
        88  FCA-TERM-CONTINUE VALUE 0.

```



```

      88 FCA-TERM-LASTCALL      VALUE 1.
05 FCA-WS-INIT-OPTIONS        PIC 9(1)  VALUE 0.
      88 FCA-WS-INIT-LEAVE      VALUE 0.
      88 FCA-WS-INIT-CLEAR      VALUE 1.
05 FCA-INITAPI-REDEFINE       PIC 9(1)  VALUE 0.
      88 FCA-DEFINE-FIRST      VALUE 0.
      88 FCA-DEFINE-ALWAYS     VALUE 1.
05 FCA-RESERVED               PIC X(8).
05 FCA-CSECT-NAME             PIC X(8)  VALUE SPACES.
05 FCA-CSECT-LANGUAGE-ID     PIC X(1).
      88 FCA-COBOL-VS           VALUE 'V'.
      88 FCA-COBOL-2           VALUE '2'.
      88 FCA-COBOL-FOR-MVS      VALUE '2'.
      88 FCA-COBOL-FOR-OS390    VALUE '2'.
      88 FCA-PL1                VALUE 'P'.
      88 FCA-C370               VALUE 'C'.
      88 FCA-ASM370             VALUE 'A'.
05 FCA-FILLER                 PIC X(3).
05 FCA-TRS-MODE               PIC X(1).
      88 FCA-PRECOMP            VALUE 'P'.
      88 FCA-MANUAL-API         VALUE 'M'.
05 FCA-HOST-REOPENS           PIC X(1).
      88 FCA-RESTART-REOPENS    VALUE 'O'.
      88 FCA-NO-REOPENS         VALUE 'N'.
05 FCA-COMMIT                 PIC X(1).
      88 FCA-COMMIT-PHYSICAL     VALUE 'P'.
      88 FCA-COMMIT-LOGICAL      VALUE 'L'.
05 FCA-VERIFY-DATA            PIC X(1).
      88 FCA-DATA-NOT-VALID      VALUE 'V'.
      88 FCA-DATA-VALID         VALUE ' '.
05 FCA-COMPARE-WARNING        REDEFINES
FCA-VERIFY-DATA              PIC X(1).
05 FCA-FUNCTION-DEFLECTION    PIC X(1).
      88 FCA-DEFLECT-ALL         VALUE 'I'.
*      88 FCA-ALLOW-ALL         VALUE 'I'.
*
05 FCA-INIT-STYLE             PIC X(1)  VALUE 'N'.
      88 FCA-INIT-STYLE-NONE     VALUE 'N'.
      88 FCA-INIT-STYLE-LEN      VALUE 'L'.
      88 FCA-INIT-STYLE-END      VALUE 'E'.
05 FCA-SRSPC-LEVEL            PIC S9(4)  VALUE +0  COMP.
05 FCA-MQ-CONNECT-HANDLE      PIC S9(9)  VALUE +0  COMP.
05 FCA-CKPT-FREQ              PIC S9(4)  VALUE +0  COMP.
05 FCA-CKPT-COUNTER           PIC S9(4)  VALUE +0  COMP.
05 FCA-PGM-RETURN-CODE        PIC S9(4)  VALUE +0  COMP.
05 FCA-FILLER3                PIC S9(4)  VALUE +0  COMP.
05 FCA-CKPT-PACE              PIC S9(9)  VALUE +0  COMP.
05 FCA-CKPT-PACE-COUNTER      PIC S9(9)  VALUE +0  COMP.
05 FCA-FILLER2                PIC X(8).
*--- END OF FCA STRUCTURE -----

```

```

* -----
*
* SMART/RESTART
* FILE CONTROL STRUCTURE
*
* DEFINE ONE FCS FOR EACH CONCURRENTLY OPENED FILE
*
* -----
01 :TAG:-FCS.
05 :TAG:-DDNAME                PIC X(8)  VALUE SPACES.
05 :TAG:-INTENT                PIC X(8)  VALUE SPACES.
05 :TAG:-ACCESS                PIC X(1)  VALUE 'S'.
      88 :TAG:-BSAM             VALUE 'S'.
      88 :TAG:-VSAM             VALUE 'V'.
05 :TAG:-FLAG                  PIC 9(1)  VALUE 0.
      88 :TAG:-OK               VALUE 0.
      88 :TAG:-END-OF-FILE       VALUE 1.
      88 :TAG:-ERROR-OCCURED     VALUE 2.
      88 :TAG:-NOT-ALLOCATED     VALUE 3.
      88 :TAG:-KEY-OK            VALUE 4.
      88 :TAG:-KEY-INVALID       VALUE 5.

```

* File characteristics area - overrides JCL when specified

```

05 :TAG:-CHARACTERISTICS.
10 :TAG:-LRECL          PIC S9(4) COMP VALUE 0.
10 :TAG:-BLKSIZE        PIC S9(4) COMP VALUE 0.
10 :TAG:-RECFM          PIC X(3)  VALUE SPACES.
10 :TAG:-ADVO           PIC X(1)  VALUE SPACES.
88 :TAG:-ADV            VALUE 'A'.
10 :TAG:-DIRECTION      PIC X(1)  VALUE SPACES.
88 :TAG:-AFTER          VALUE 'A'.
88 :TAG:-BEFORE         VALUE 'B'.
10 :TAG:-ADV-UNIT       PIC X(1)  VALUE SPACES.
88 :TAG:-LINE           VALUE 'L'.
88 :TAG:-CHAN           VALUE 'C'.
10 :TAG:-DIRECTION-MIX  PIC X(1)  VALUE SPACES.
88 :TAG:-CASA           VALUE 'A'.
88 :TAG:-CMCH           VALUE 'M'.
10 :TAG:-RESRV1         PIC X(1)  VALUE SPACES.
10 :TAG:-ADVANCING      PIC 9(3)  COMP VALUE ZEROES.
10 :TAG:-CONCAT-SEQ-NUM PIC 9(3)  COMP VALUE ZEROES.
10 :TAG:-VOLUME-SEQ-NUM PIC 9(3)  COMP VALUE ZEROES.

```

*

*

*

FCS Version Release Modification Level identifier

```

05 :TAG:-LEVEL          PIC S9(4) VALUE +720 COMP.
88 :TAG:-LEVEL-CURRENT  VALUE +720.
88 :TAG:-LEVEL-V720     VALUE +720.
88 :TAG:-LEVEL-V712     VALUE +712.

```

*

```

05 :TAG:-RESRV2         PIC X(2)  VALUE SPACES.

```

*

Record identifier for use with the "START" service only

```

05 :TAG:-RECID          PIC X(8)  VALUE SPACES.

```

*

*

Dynamic allocation information area - no dynamic allocation attempted if the data set name is blank

```

05 :TAG:-DYNAMIC-ALLOCATION.
10 :TAG:-DSNAME         PIC X(44)  VALUE SPACES.
10 :TAG:-MEMBER         PIC X(8)   VALUE SPACES.
10 :TAG:-UNIT           PIC X(8)   VALUE SPACES.
10 :TAG:-VOLSER         PIC X(6)   VALUE SPACES.
10 :TAG:-PRIMARY-SPACE  PIC S9(4)  COMP VALUE 0.
10 :TAG:-SECONDARY-SPACE PIC S9(4)  COMP VALUE 0.

```

*

Tape related fields

```

10 :TAG:-TAPE-RETPD     PIC S9(4)  COMP VALUE 0.
10 :TAG:-TAPE-DSN-SEQNUM PIC S9(4)  COMP VALUE 0.
10 :TAG:-TAPE-LABEL     PIC X(1)   VALUE SPACE.
88 :TAG:-TAPE-LABEL-NL  VALUE '1'.
88 :TAG:-TAPE-LABEL-SL  VALUE '2'.
88 :TAG:-TAPE-LABEL-NSL VALUE '3'.
88 :TAG:-TAPE-LABEL-BLP VALUE '4'.

```

```

10 :TAG:-DASD-ALLOC-UNITS PIC X(1)  VALUE SPACE.
88 :TAG:-DASD-CYLINDERS   VALUE 'C'.
88 :TAG:-DASD-TRACKS     VALUE 'T'.
10 :TAG:-DISP-STATUS      PIC X(1)  VALUE SPACE.
88 :TAG:-DISP-STATUS-OLD  VALUE '1'.
88 :TAG:-DISP-STATUS-MOD  VALUE '2'.
88 :TAG:-DISP-STATUS-NEW  VALUE '3'.
88 :TAG:-DISP-STATUS-SHR  VALUE '4'.
10 :TAG:-NORMAL-DISP      PIC X(1)  VALUE SPACE.
88 :TAG:-NORMAL-DISP-UNCATLG VALUE '1'.
88 :TAG:-NORMAL-DISP-CATLG  VALUE '2'.
88 :TAG:-NORMAL-DISP-DELETE VALUE '3'.
88 :TAG:-NORMAL-DISP-KEEP  VALUE '4'.
10 :TAG:-ABNORMAL-DISP    PIC X(1)  VALUE SPACE.
88 :TAG:-ABNORMAL-DISP-UNCATLG VALUE '1'.
88 :TAG:-ABNORMAL-DISP-CATLG  VALUE '2'.
88 :TAG:-ABNORMAL-DISP-DELETE VALUE '3'.
88 :TAG:-ABNORMAL-DISP-KEEP  VALUE '4'.

```

```

*   Next output field set by the Smart/RESTART OPEN
*   service or FILE-INFO API call

      05 :TAG:-REPO-STYLE          PIC X(1)  VALUE SPACES.
          88 :TAG:-REPO-STYLE-AFTER VALUE 'A'.
          88 :TAG:-REPO-STYLE-AT   VALUE 'T'.
          88 :TAG:-REPO-STYLE-NONE VALUE 'N'.

      05 :TAG:-EBLKSIZ8          PIC S9(16) COMP VALUE 0.
      05 :TAG:-EBLKSIZ-DETAIL    REDEFINES
      :TAG:-EBLKSIZ8.
          10 :TAG:-EBLKSIZ-HIGH   PIC S9(8)  COMP.
          10 :TAG:-EBLKSIZ       PIC S9(8)  COMP.

      05 :TAG:-RESRV3            PIC X(8)   VALUE SPACES.

* END OF FCS STRUCTURE -----

```

B.3 C Language Header Files

```

/*-----*/
/*
/*   Smart/RESTART Service names for C
/*
/*-----*/

char rai_abend[13]      = "ABEND      ";
char rai_allocate[13]   = "ALLOCATE   ";
char rai_cancel[13]     = "CANCEL     ";
char rai_checkpoint[13] = "CHECKPOINT ";
char rai_ckpt_immed[13] = "CKPT_IMMED ";
char rai_close[13]      = "CLOSE      ";
char rai_cursor_hold[13] = "CURSOR_HOLD ";
char rai_eoj[13]        = "END_OF_JOB ";
char rai_envcheck[13]    = "ENVCHECK   ";
char rai_message_log[13] = "MESSAGE_LOG ";
char rai_file_info[13]  = "FILE_INFO  ";
char rai_free[13]       = "FREE       ";
char rai_fsopen[13]     = "FSOPEN     ";
char rai_initialize[13]  = "INITIALIZE ";
char rai_open[13]       = "OPEN       ";
char rai_profile[13]     = "PROFILE    ";
char rai_read[13]       = "READ      ";
char rai_rollback[13]   = "ROLLBACK  ";
char rai_rollims[13]    = "ROLLIMS   ";
char rai_snap[13]       = "SNAP      ";
char rai_stop_run[13]   = "STOP_RUN   ";
char rai_sqlca[13]      = "SQLCA      ";
char rai_start[13]      = "START      ";
char rai_terminate[13]  = "TERMINATE ";
char rai_write[13]      = "WRITE     ";
char rai_raiapi9"       = "RAIAPI   ";
char rai_restart[13]    = "RESTART    ";
char rai_envload[13]    = "ENVLOAD   ";
char rai_storage_rest[13] = "STORAGE_REST";

/* -- END OF RAI SERVICE NAMES STRUCTURE -----*/

/*-----*/
/*
/*   SMART/RESTART
/*
/*   FUNCTION COMMUNICATIONS AREA FOR C
/*
/*-----*/

```

```

struct srs_fca {
    char          fca_status[4];
    short int     fca_return_code;
    char          fca_error_msg[8];
    char          fca_last_function[12];
    short int     fca_user_abend_code;
    unsigned char fca_dump_options;
    unsigned char fca_term_options;
    unsigned char fca_init_options;
    unsigned char fca_init_redefine;
    char          fca_reserved1[8];
    char          fca_csect_name[8];
    char          fca_csect_language_id[1];
    char          fca_reserved2[3];
    char          fca_trs_mode[1];
    char          fca_host_reopens[1];
    char          fca_commit[1];
    char          fca_compare_warning[1];
    char          fca_function_deflect[1];
    char          fca_init_style[1];
    short int     fca_srspc_level;
    char          fca_mq_connect_handle[4];
    short int     fca_ckpt_freq;
    short int     fca_ckpt_counter;
    short int     fca_pgm_return_code;
    short int     fca_reserved3;
    int           fca_ckpt_pace;
    int           fca_ckpt_pace_counter;
    char          fca_reserved4[8];
};

/*-----*/
/*
/*      REMARKS:
/*      1. FUNCTION COMMUNICATIONS AREA.
/*      2. REQUEST COMPLETION STATUS (SEE BELOW):
/*      '    ' - FUNCTION SUCCESSFUL
/*      'REST' - RESTART IN PROGRESS
/*      'ROLL' - ROLLBACK JUST OCCURED
/*      'EC00' - EXTERNAL COMMIT JUST ISSUED WAS SUCCESSFUL
/*      'EC08' - EXTERNAL COMMIT JUST ISSUED FAILED
/*      3. FUNCTION RETURN CODE.
/*      4. EXTENDED ERROR INFORMATION.
/*      5. FUNCTION REQUEST.
/*      6. USER ABEND CODE.
/*      7. USER ABEND DUMP OPTION (SEE BELOW):
/*      0 - DO NOT PRINT DUMP
/*      1 - PRINT DUMP
/*      8. SUBPROGRAM TERMINATE REQUEST OPTION (SEE BELOW):
/*      0 - DO NOT FREE CSB
/*      1 - FREE CSB - NO FURTHER CALLS TO CSB
/*      9. SUBPROGRAM INITIAL OPTION FOR ... (SEE BELOW):
/*      0 - ... NOT CLEARED
/*      1 - ... CLEARED
/*      10. SUBPROGRAM INITIALIZE API REDEFINITION (SEE BELOW):
/*      0 - ... DEFINE SUBPROGRAM SAVE AREAS FIRST INIT API CALL
/*      1 - ... REDEFINE SUBPROGRAM SAVE AREAS EACH INIT API CALL
/*      11. RESERVED
/*      12. CSECT NAME
/*      13. CSECT LANGUAGE ID IS EQUAL TO P FOR PL/I
/*      14. TRANSPARENCY MODE MAY BE 'M' OR 'P' FOR A PL/I CSECT
/*      15. PRECOMPILER INSERTED HOST REOPENS NOT USED IN PL/I
/*      16. FCA COMMIT TYPE
/*      P - PHYSICAL COMMIT COMPLETED
/*      L - LOGICAL COMMIT COMPLETED
/*      17. FCA data compare validation check at restart
/*      18. FCA function deflection flag
/*      19. FCA MQ Series connection handle
/*
/*
/*      Smart/RESTART maintains data areas <20> and <21> whenever a
/*      command CKPT_FREQ is used to set the checkpoint pacing rate:
/*
/*      20. logical CKPTs per Physical ckpt
/*

```

```

/*      21. CKPT calls since last physical ckpt      */
/*      */
/*      22. Controls the way in which program storage description */
/*      fields for areas to be checkpointed are interpreted when */
/*      specified as arguments of the initialize function.      */
/*      N - (non-initialized field)                        */
/*      L - Length - pairs of data items recognized as      */
/*              - <data_item>,<length-field>.                */
/*      E - End      - pairs of data items recognized as      */
/*              - <starting_field>,<ending_field>.            */
/*      23. Precompiler Version, Release and Modification level */
/*      24. Precompiler preserves the value of the PL/1 special */
/*      register RETCODE during the RAI_TERMINATE API call      */
/*      */
/*      Smart/RESTART maintains data areas <25> and <26> whenever a */
/*      command CKPT_PACE is used to set the checkpoint pacing rate: */
/*      */
/*      25. Logical checkpoints per physical checkpoint      */
/*      26. Checkpoint calls since last physical checkpoint  */
/*-----*/
/*--- end of fca structure -----*/

/*-----*/
/*      */
/*      SMART/RESTART      */
/*      */
/*      FILE CONTROL STRUCTURE FOR C      */
/*      */
/*-----*/

struct srs_fcs {
char fcs_ddname[8];
char fcs_intent[8];
char fcs_access[1];
char fcs_flag[1];

/* 1 */
/* 2 */
/* 3 */
/* 4 */
/* 5 */

/* FILE CHARACTERISTICS AREA - OVERRIDES JCL WHEN SPECIFIED */

short int fcs_lrecl;
short int fcs_blksize;
char fcs_recfm[3];
char fcs_adv [1];
char fcs_dir [1];
char fcs_aunit[1];
char fcs_cmix [1];
char fcs_resrx[1];
char fcs_advan[2];
char fcs_ccseq[2];
char fcs_vlseq[2];

/* 6 */
/* 7 */
/* 8 */
/* 9 */
/* 10 */
/* 11 */
/* 12 */
/* 13 */
/* 14 */
/* 15 */
/* 16 */

/* FCS Version Release Modification Level identifier */

short int fcs_level;

char fcs_resrv[2];

/* 17 */
/* 18 */

/* RECORD IDENTIFIER FOR USE WITH "START" SERVICE ONLY */

char fcs_recid[8];

/* 19 */

/* DYNAMIC ALLOCATION INFORMATION AREA - NO DYNAMIC */
/* ALLOCATION ATTEMPTED IF THE DATA SET NAME IS BLANK */

char fcs_dsname[44];
char fcs_member[8];
char fcs_unit[8];
char fcs_volsr[6];
short int fcs_prim;
short int fcs_sec;

/* 20 */
/* 21 */
/* 22 */
/* 23 */
/* 24 */
/* 25 */

```

```

/* TAPE RELATED FIELDS                                     */
short int fcs_retpd;                                       /* 27 */
short int fcs_dsseq;                                       /* 28 */
char fcs_label;                                           /* 28 */
char fcs_allun;                                           /* 29 */
char fcs_disps;                                           /* 30 */
char fcs_dispn;                                           /* 31 */
char fcs_dispa;                                           /* 32 */
char fcs_repo_style[1];                                    /* 33 */
long int fcs_eblksize_high;                                /* 34 */
long int fcs_eblksize;                                    /* 35 */
char fcs_resrv3[8];                                       /* 18 */
};

/*-----*/
/* REMARKS:                                                */
/* 1. FILE CONTROL STRUCTURE                               */
/* 2. FILE DDNAME                                           */
/* 3. INTENT - 'INPUT' OR 'OUTPUT'                         */
/* 4. ACCESS METHOD INDICATOR - V=VSAM S=BSAM              */
/* 5. FCS STATUS CODE:                                     */
/* 0 - READ/WRITE PERFORMED                               */
/* 1 - END OF DATA REACHED                               */
/* 2 - I/O ERROR OCCURED                                  */
/* 3 - NOT ALLOCATED                                       */
/* 6. LOGICAL RECORD LENGTH (LRECL)                       */
/* 7. BLOCK SIZE (BLKSIZE)                                 */
/* 8. RECORD FORMAT (RECFM):                               */
/* F - FIXED                                               */
/* FB - FIXED BLOCKED                                     */
/* V - VARIABLE                                            */
/* VB - VARIABLE BLOCKED                                  */
/* U - UNDEFINED                                           */
/* 9. FLAGW FOR COBOL WRITE ADVANCING                     */
/* 10. FLAGW FOR COBOL WRITE ADVANCING                    */
/* 11. FLAGW FOR COBOL WRITE ADVANCING                    */
/* 12. FLAGW FOR COBOL WRITE ADVANCING                    */
/* 13. reserved                                           */
/* 14. ADVANCING OPTION AMOUNT                             */
/* 15. CONCATENATED FILE SEQUENCE NUMBER                  */
/* 16. VOLUME SEQUENCE NUMBER FOR CURRENT DATASET         */
/* 17. Identify FCS Version Release Modification Level:   */
/* 720 - Smart/RESTART Version 7.2.0.                    */
/* 712 - Smart/RESTART Version 7.1.2.                    */
/* 18. RESERVED                                           */
/* 19. COMBINATION OF TTRZ OR RBA AND BUFX AND VOLSEQ IN  */
/* Order TO UNIQUELY POSITION TO AN LREC                    */
/* 20. DATA SET NAME                                     */
/* 21. MEMBER NAME                                         */
/* 22. UNIT                                                */
/* 23. VOLUME SERIAL NUMBER                               */
/* 24. PRIMARY ALLOCATION                                  */
/* 25. SECONDARY ALLOCATION                                */
/* 26. RETENTION PERIOD (TAPE LABEL)                     */
/* 27. DATA SET SEQUENCE NUMBER (TAPE LABEL)            */
/* 28. LABEL TYPE:                                         */
/* '1' - NL                                                */
/* '2' - SL                                                */
/* '3' - NSL                                               */
/* '4' - BLP                                               */
/* 29. ALLOCATION UNITS                                     */
/* 'C' - CYLINDERS                                         */
/* 'T' - TRACKS                                            */
/* 30. INITIAL DISPOSITION STATUS:                       */
/* '1' - OLD                                               */
/* '2' - MOD                                               */
/* '3' - NEW                                               */
/* '4' - SHR                                               */
/* 31. NORMAL DISPOSITION:                                */
/* '1' - UNCATLG                                           */
/* '2' - CATLG                                             */
/* '3' - DELETE                                           */
/* '4' - KEEP                                              */

```

```

/* 32. ABNORMAL DISPOSITION: */
/* '1' - UNCATLG */
/* '2' - CATLG */
/* '3' - DELETE */
/* '4' - KEEP */
/* 33. File repositioning style */
/* 'A' - After committed record */
/* 'T' - At committed record - reread committed record */
/* 'N' - Non-repositionable file */
/* Note: This feedback field is preset by Smart/RESTART */
/* OPEN or FILE_INFO call level service */
/* 34. *** reserved *** */
/* 35. Hi-order half of Blocksize */
/* 36. Block size (Larger Block Interface) */
/*-----*/

/* END OF FCS STRUCTURE -----*/

```

B.4 PL/1 Include Members

```

/*-----*/
/* */
/* SMART/RESTART SERVICE NAMES FOR PLI */
/* */
/*-----*/
DECLARE
1 RAI_SERVICES STATIC,
2 RAI_ABEND CHAR(12) INIT('ABEND'),
2 RAI_ALLOCATE CHAR(12) INIT('ALLOCATE'),
2 RAI_CANCEL CHAR(12) INIT('CANCEL'),
2 RAI_CHECKPOINT CHAR(12) INIT('CHECKPOINT'),
2 RAI_CKPT_IMMED CHAR(12) INIT('CKPT_IMMED'),
2 RAI_CLOSE CHAR(12) INIT('CLOSE'),
2 RAI_CURSOR_HOLD CHAR(12) INIT('CURSOR_HOLD'),
2 RAI_END_OF_JOB CHAR(12) INIT('END_OF_JOB'),
2 RAI_ENVCHECK CHAR(12) INIT('ENVCHECK'),
2 RAI_FILE_INFO CHAR(12) INIT('FILE_INFO'),
2 RAI_FREE CHAR(12) INIT('FREE'),
2 RAI_FSOPEN CHAR(12) INIT('FSOPEN'),
2 RAI_INITIALIZE CHAR(12) INIT('INITIALIZE'),
2 RAI_MESSAGE_LOG CHAR(12) INIT('MESSAGE_LOG'),
2 RAI_OPEN CHAR(12) INIT('OPEN'),
2 RAI_PROFILE CHAR(12) INIT('PROFILE'),
2 RAI_READ CHAR(12) INIT('READ'),
2 RAI_ROLLBACK CHAR(12) INIT('ROLLBACK'),
2 RAI_ROLLIMS CHAR(12) INIT('ROLLIMS'),
2 RAI_SNAP CHAR(12) INIT('SNAP'),
2 RAI_STOP_RUN CHAR(12) INIT('STOP_RUN'),
2 RAI_SQLCA CHAR(12) INIT('SQLCA'),
2 RAI_START CHAR(12) INIT('START'),
2 RAI_TERMINATE CHAR(12) INIT('TERMINATE'),
2 RAI_WRITE CHAR(12) INIT('WRITE'),
2 RAI_RAIAPI CHAR(08) INIT('RAIAPI'),
2 RAI_RESTART CHAR(12) INIT('RESTART'),
2 RAI_ENVLOAD CHAR(12) INIT('ENVLOAD'),
2 RAI_STORAGE_REST CHAR(12) INIT('STORAGE_REST');

/* -- END OF RAI SERVICE NAMES STRUCTURE -----*/

/*-----*/
/* */
/* SMART/RESTART */
/* */
/* FUNCTION COMMUNICATIONS AREA FOR PLI */
/* */
/*-----*/

```

```

DECLARE
  1 FCA_STATIC,                                /* 1 */
  2 FCA_STATUS                                CHAR(4),          /* 2 */
  2 FCA_RETURN_CODE                          BIN FIXED,      /* 3 */
  2 FCA_ERROR_MSG                           CHAR(8),          /* 4 */
  2 FCA_LAST_FUNCTION                        CHAR(12),         /* 5 */
  2 FCA_USER_ABEND_CODE                      BIN FIXED,        /* 6 */
  2 FCA_DUMP_OPTIONS                         PIC'9' INIT(0),   /* 7 */
  2 FCA_TERM_OPTIONS                         PIC'9' INIT(0),   /* 8 */
  2 FCA_INIT_OPTIONS                         PIC'9' INIT(0),   /* 9 */
  2 FCA_INIT_REDEFINE                       PIC'9' INIT(0),   /* 10 */
  2 FCA_RESERVED1                           CHAR(8),          /* 11 */
  2 FCA_CSECT_NAME                           CHAR(8) INIT(' '), /* 12 */
  2 FCA_CSECT_LANG_ID                       CHAR(1),          /* 13 */
  2 FCA_RESERVED2                           CHAR(3),          /* 11 */
  2 FCA_TRS_MODE                             CHAR(1),          /* 14 */
  2 FCA_HOST_REOPENS                        CHAR(1),          /* 15 */
  2 FCA_COMMIT                              CHAR(1),          /* 16 */
  2 FCA_COMPARE_WARNING                     CHAR(1),          /* 17 */
  2 FCA_FUNC_DEFLECTION                     CHAR(1),          /* 18 */
  2 FCA_INIT_STYLE                           CHAR(1),          /* 22 */
  2 FCA_SRSPC_LEVEL                         BIN FIXED,         /* 23 */
  2 FCA_MQ_CONNECT_HAND                     FIXED BIN(31),     /* 19 */
  2 FCA_CKPT_FREQ                           BIN FIXED,         /* 20 */
  2 FCA_CKPT_COUNTER                        BIN FIXED,         /* 21 */
  2 FCA_PGM_RETURN_CODE                     BIN FIXED,         /* 24 */
  2 FCA_RESERVED3                           BIN FIXED,         /* 11 */
  2 FCA_CKPT_PACE                           FIXED BIN(31),     /* 25 */
  2 FCA_CKPT_PACE_COUNTER                   FIXED BIN(31),     /* 26 */
  2 FCA_RESERVED4                           CHAR(8);           /* 11 */
/*-----*/
/*
/*      Notes:
/*
/*      1. Function Communications Area.
/*
/*      2. Request completion status:
/*          ' ' - Function successful
/*          'REST' - Restart is in progress
/*          'ROLL' - Rollback just occurred
/*          'EC00' - External commit just issued was successful
/*          'EC08' - External commit just issued has failed
/*
/*      3. Function return code.
/*      4. Extended error information.
/*      5. Function request.
/*      6. User abend code.
/*
/*      7. User abend dump option:
/*          0 - Do not print dump
/*          1 - Print dump
/*
/*      8. Subprogram terminate request option:
/*          0 - Do not free CSECT Specifications Block (CSB)
/*          1 - Free CSB - No further calls to this subprogram
/*
/*      9. Subprogram initial option for:
/*          0 - Not cleared
/*          1 - Cleared
/*
/*      10. Subprogram initialize API redefinition:
/*          0 - Define subprogram save areas first initialize API call
/*          1 - Redefine subprogram save areas each initialize call
/*
/*      11. RESERVED
/*      12. CSECT (PROCEDURE) NAME
/*      13. CSECT LANGUAGE ID IS EQUAL TO P FOR PL/I
/*      14. TRANSPARENCY MODE MAY BE 'M' OR 'P' FOR A PL/I CSECT
/*      15. PRECOMPILER INSERTED HOST REOPENS NOT USED IN PL/I
/*      16. FCA COMMIT TYPE
/*          P - PHYSICAL COMMIT COMPLETED
/*          L - LOGICAL COMMIT COMPLETED
/*      17. FCA DATA VALIDITY CHECKING RESULTS AT RESTART

```



```

/*      18. FCA FUNCTION DEFLECTION FLAG                                */
/*      19. FCA MQ SERIES CONNECTION HANDLE                            */
/*      */                                                            */
/*      Smart/RESTART maintains data areas <20> and <21> whenever a    */
/*      command CKPT_FREQ is used to set the checkpoint pacing rate: */
/*      */                                                            */
/*      20. LOGICAL CKPTS PER PHYSICAL CKPT                           */
/*      21. CKPT CALLS SINCE LAST PHYSICAL CKPT                       */
/*      22. CONTROLS THE WAY IN WHICH PROGRAM STORAGE DESCRIPTION    */
/*      FIELDS FOR AREAS TO BE CHECKPOINTED ARE INTERPRETED WHEN     */
/*      SPECIFIED AS ARGUMENTS OF THE INITIALIZE FUNCTION.           */
/*      N - (NON-INITIALIZED FIELD)                                   */
/*      L - LENGTH - PAIRS OF DATA ITEMS RECOGNIZED AS              */
/*      - <DATA_ITEM>,<LENGTH-FIELD>.                                  */
/*      E - END - PAIRS OF DATA ITEMS RECOGNIZED AS                 */
/*      - <STARTING_FIELD>,<ENDING_FIELD>.                             */
/*      23. PRECOMPILER VERSION, RELEASE AND MODIFICATION LEVEL      */
/*      24. PRECOMPILER PRESERVES THE VALUE OF THE PL/1 SPECIAL      */
/*      REGISTER "RETCODE" DURING THE RAI_TERMINATE API CALL          */
/*      */                                                            */
/*      Smart/RESTART maintains data areas <25> and <26> whenever a    */
/*      command CKPT_PACE is used to set the checkpoint pacing rate: */
/*      */                                                            */
/*      25. Logical checkpoints per physical checkpoint               */
/*      26. Checkpoint calls since last physical checkpoint.         */
/*-----*/
/*--- END OF FCA STRUCTURE -----*/

```

```

/*-----*/
/*      SMART/RESTART                                                */
/*      FILE CONTROL STRUCTURE FOR PLI                               */
/*-----*/

```

DECLARE

```

1 FCS,                                /* 1 */
2 FCS_DDNAME CHAR(8) INIT(' '),      /* 2 */
2 FCS_INTENT CHAR(8) INIT(' '),      /* 3 */
2 FCS_ACCESS CHAR(1) INIT('S'),       /* 4 */
2 FCS_FLAG PIC'9',                   /* 5 */

```

```

/* File characteristics area which */
/* overrides JCL when specified */

```

```

2 FCS_LRECL BIN FIXED INIT(0),        /* 6 */
2 FCS_BLKSIZE BIN FIXED INIT(0),      /* 7 */
2 FCS_RECFCM CHAR(3) INIT(' '),       /* 8 */
2 FCS_ADV PIC'9' INIT(0),             /* 9 */
2 FCS_DIR PIC'9' INIT(0),             /* 10 */
2 FCS_AUNIT PIC'9' INIT(0),           /* 11 */
2 FCS_CMIX PIC'9' INIT(0),            /* 12 */
2 FCS_RESRV1 PIC'9' INIT(0),          /* 13 */
2 FCS_ADVANC BIN FIXED INIT(0),       /* 14 */
2 FCS_CCSEQ BIN FIXED INIT(0),        /* 15 */
2 FCS_VLSEQ BIN FIXED INIT(0),        /* 16 */

```

```

/* FCS Version Release Modification Level identifier */

```

```

2 FCS_LEVEL BIN FIXED INIT(720),      /* 17 */
2 FCS_RESRV2 CHAR(2) INIT(' '),       /* 18 */

```

```

/* Record identifier for use with "START" service only */

```

```

2 FCS_RECID CHAR(8) INIT(' '),        /* 19 */

```

```

/* Dynamic allocation information area.  No dynamic */
/* allocation attempted if the data set name is blank */

2 FCS_DSNAME  CHAR(44)      INIT(' '), /* 20 */
2 FCS_MEMBER  CHAR(8)       INIT(' '), /* 21 */
2 FCS_UNIT    CHAR(8)       INIT(' '), /* 22 */
2 FCS_VOLSER   CHAR(6)      INIT(' '), /* 23 */
2 FCS_PRIM    BIN FIXED    INIT(0), /* 24 */
2 FCS_SEC     BIN FIXED    INIT(0), /* 25 */

/* Tape related fields */

2 FCS_RETPD   BIN FIXED    INIT(0), /* 26 */
2 FCS_DSSEQ   BIN FIXED    INIT(0), /* 27 */
2 FCS_LABEL   CHAR(1)      INIT(' '), /* 28 */
2 FCS_ALLUN   CHAR(1)      INIT(' '), /* 29 */
2 FCS_DISPS   CHAR(1)      INIT(' '), /* 30 */
2 FCS_DISP    CHAR(1)      INIT(' '), /* 31 */
2 FCS_DISPA   CHAR(1)      INIT(' '), /* 32 */
2 FCS_REPO_STYLE CHAR(1)  INIT(' '), /* 33 */

2 FCS_EBLKSIZE8 , /* 34 */
3 FCS_EBLKSIZE_HIGH BIN FIXED (31) INIT(0), /* 35 */
3 FCS_EBLKSIZE     BIN FIXED (31) INIT(0), /* 36 */

2 FCS_RESRV3  CHAR(8)      INIT(' '); /* 18 */

/*-----*/
/*
/* Remarks:
/*
/* 1. File control structure
/* 2. file DDname
/* 3. intent - 'input' or 'output'
/* 4. access method indicator - v=vsam s=bsam
/* 5. FCS status code:
/* 0 - read/write performed
/* 1 - end of data reached
/* 2 - i/o error occurred
/* 3 - not allocated
/* 6. logical record length (LRECL)
/* 7. block size (BLKSIZE)
/* 8. record format (RECFM):
/* F - fixed
/* FB - fixed blocked
/* V - variable
/* VB - variable blocked
/* U - undefined
/* 9 - 12. flag bytes for write advancing option
/* 13. reserved
/* 14. amount for write advancing
/* 15. concatenation sequence number for dsn
/* 16. volume sequence number for current dataset
/* 17. Identify FCS Version Release Modification Level:
/* 720 - Smart/RESTART Version 7.2.0.
/* 712 - Smart/RESTART Version 7.1.2.
/* 18. reserved
/* 19. combination of TTRZ or RBA and BUFX and VOLSEQ in
/* order to uniquely position to a logical record
/* 20. data set name
/* 21. member name
/* 22. unit
/* 23. volume serial number
/* 24. primary allocation
/* 25. secondary allocation
/* 26. retention period (tape label)
/* 27. data set sequence number (tape label)
/* 28. label type:
/* '1' - NL
/* '2' - SL
/* '3' - NSL
/* '4' - BLP
/* 29. Allocation units
/* 'C' - cylinders
/*

```

```

/*      'T' - tracks                                     */
/* 30. Initial disposition status:                       */
/*      '1' - old                                       */
/*      '2' - mod                                       */
/*      '3' - new                                       */
/*      '4' - shr                                       */
/* 31. Normal disposition:                               */
/*      '1' - uncatlg                                   */
/*      '2' - catlg                                    */
/*      '3' - delete                                   */
/*      '4' - keep                                     */
/* 32. Abnormal disposition:                             */
/*      '1' - uncatlg                                   */
/*      '2' - catlg                                    */
/*      '3' - delete                                   */
/*      '4' - keep                                     */
/* 33. File repositioning style                         */
/*      'A' - After committed record                   */
/*      'T' - At committed record - reread committed record */
/*      'N' - Non-repositionable file                  #aa */
/*      Note: This feedback field is preset by Smart/RESTART */
/*              OPEN or FILE_INFO call level service      */
/* 34. Block size in 8 bytes                             */
/* 35. Hi-order half of Blocksize                       */
/* 36. Block size (Larger Block Interface)              */
/*-----*/
/* END OF FCS STRUCTURE -----*/

```


Smart/RESTART's DB2

Checkpoint Synchronization Table

Figure C.1 illustrates the SQL DDL with which the DB2 CKPT_SYNCH table is created and shows the columns which comprise the table. The numbers in parentheses to the right of each column correspond to the numbered, annotating paragraphs which follow the figure.

```
create table DCAvrM.ckpt_synch
(ckpt_jobkey          char(32)      not null,  (1)
 ckpt_skey_rrn        integer       not null,  (2)
 ckpt_rrn             integer       not null,  (3)
 ckpt_number          char(7)       not null,  (4)
 ckpt_keytime         timestamp     not null,  (5)
 ckpt_freq            int           not null,  (6)
 ckpt_interval        int           not null,  (7)
 ckpt_start_date      date          not null,  (8)
 ckpt_start_time      time          not null,  (9)
 ckpt_ckpt_date       date          not null,  (10)
 ckpt_ckpt_time       time          not null,  (11)
 ckpt_term_code       char(8)       not null,  (12)
 ckpt_ckpt_dsn        char(44)      not null,  (13)
 ckpt_jobname         char(8)       not null,  (14)
 ckpt_jobnum          char(8)       not null,  (15)
 ckpt_enddate         char(8)       not null,  (16)
 ckpt_endtime         char(8)       not null,  (17)
 ckpt_fourk_datablk   varchar(3883) not null,  (18)
 primary key(ckpt_jobkey, ckpt_skey_rrn) )    (19)
in DCADBvrM.DCATSvrM;
```

Figure C.1 Column structure of the CKPT_SYNCH table

DCAvrn denotes the owner of the CKPT_SYNCH table. Typically the authorization ID starts with "DCA" and is followed by a string of numbers that indicate the version, release and modification level of Smart/RESTART.

1. The **CKPT_JOBKEY** column is the Smart/RESTART job identification string (Job ID). There is one Job ID for every restartable job step. Use the Smart/RESTART profile parameter JOBID to specify this explicitly. Use the Smart/RESTART profile parameter AUTO_JOBID to control how Smart/RESTART generates the Job ID. Additionally, see the discussion of the VERIFY_JOBID parameter in the Smart/RESTART Reference for additional information.
2. For restartable job steps whose main (or bulk) checkpoint medium is DB2, the **CKPT_SKEY_RRN** column contains the **Relative Record Number** of the 4K storage block in the CKPT_FOURK_DATA_BLK column. For job steps that use VSAM, BDAM or TAPE as checkpoint media, the CKPT_SKEY_RRN value is -1.
3. The **CKPT_RRN** column contains the Relative Record Number of the first record of the current checkpoint blockset in the main checkpoint file. It is used with all checkpoint media.
4. The **CKPT_NUMBER** column identifies the absolute checkpoint ID number across all run(s).
5. The **CKPT_KEYTIME** column reflects the time when this row was inserted or updated within the CKPT_SYNCH table.
6. The **CKPT_FREQ** column is the current or last known value of the Smart/RESTART checkpoint pacing parameters CKPT_PACE or CKPT_FREQ.
7. The **CKPT_INTERVAL** column is the current or last known value of the Smart/RESTART checkpoint interval parameter CKPT_TIME.
8. The **CKPT_START_DATE** column is the date of the initial run of the job step.
9. The **CKPT_START_TIME** column is the starting time of the initial run of the job step.
10. The **CKPT_CKPT_DATE** column is the date of the last checkpoint.
11. The **CKPT_CKPT_TIME** column is the time of the last checkpoint.
12. The **CKPT_TERM_CODE** column contains one of the following status code values

TBLINIT	Smart/RESTART has formatted the checkpoint file for this run, but no physical checkpoint has yet been issued.
RUNNING	At least one physical checkpoint has completed for this run and the job step is still running.

NOTE: The status *RUNNING* may persist even when the jobstep abends because an unavailable resource condition may prevent Smart/RESTART from updating the main synchronization row.

COMPLETE	The jobstep completed successfully
EOJxxxxx	The jobstep termination code is set by an application using an optional parameter of the END_OF_JOB call level interface. See the Smart/RESTART Reference manual for a detailed description of the END_OF_JOB API service.
SSS	The jobstep terminated abnormally with system abend code sss.
UUUU	The jobstep terminated abnormally with user abend code uuuu.
NOTE: The Smart/MONITOR ISPF dialog displays the column CKPT_TERM_CODE below the underlined heading titled 'Status'. A portion of that screen display appears below.	
SUSPEND	The restartable application jobstep is suspended (made non-dispatchable) at the unit of work boundary associated with the next physical checkpoint event. Smart/RESTART resets the column value to 'RUNNING' once the program is re-dispatched and application processing reaches the next physical checkpoint.

```

Smart/MONITOR v.r.m           Smart/RESTART Job Queues
Command ==>
-----
Checkpoint Synchronization DB2 Table Owner ID: DCAvrn
-----
      ALL           ALL
Cmd Jobname  Job# Status
... RAI7SRS   3770 SOC7
... RAI6SRS   3068 COMPLETE

```

13. The CKPT_CKPT_DSN column contains the dataset name of the Smart/RESTART checkpoint file when the BULK_CKPT parameter specifies VSAM, BDAM, or TAPE. When the BULK_CKPT parameter specifies DB2, this column contains the literal: RAI.CKPTFILE.IS.A.DB2.TABLE.
14. The CKPT_JOBNAME column contains the name of the current restartable application job. On a restart run, it is possible for the jobname of the current job to differ from the jobname of the initial run.
15. The CKPT_JOBNUM column contains the JES Job ID of the restartable application job. This is the JES Job ID of the current job which may be either the initial run or a restart run.
16. The CKPT_ENDDATE column contains the date when the restartable job step completed successfully. The date is stored as a four digit year, followed by a period, followed by a three digit Julian date. For example, YYYY.DDD.

17. The CKPT_ENDTIME column reflects the time when the restartable job step completed successfully. The time is stored as HHMMSSSTH. That is, two digits for the hour, minute and second, followed by single digits for tenths and hundreds of a second.
18. The CKPT_FOURK_DATABLK column contains BULK checkpoint information when BULK_CKPT parameter specifies DB2. This column is not used when the checkpoint medium is a file. This is the information required for restart to occur such as application storage areas, file repositioning data, etc.

C.1 Deleting Rows from the Checkpoint Synchronization Table

The deletion of rows from the ckpt_synch table is determined both by the install time DCR (Delete Checkpoint Row) setting and the job identification. When DCR is set to ON, Smart/RESTART deletes the main synchronization row when the restartable job step completes normally. See Chapter 14 of the Smart/RESTART Reference manual for more on the DCR parameter. When DCR is set to OFF, the deletion of rows is governed by the job identification in use. If the Job ID is automatically generated, the row will not be deleted and will be re-used by later instances of the same restartable job step. If the Job ID is based on a timestamp, then every restartable job step creates a new row in this table. Smart/MONITOR can be used to clean up old rows via the retention period option. Alternatively, you can run your own SQL against this table to delete or archive old rows.