Smart / RESTART

and

Smart / RRSAF

General Information

Version 12.2.2

Febuary 2019

Publication SJS-009-18

Relational Architects Intl

This Guide: RAI Publication SJS-009-18

This document applies to:

Smart/RESTART	Version 12 Release 2.2 (Febuary 2019) and
Smart/RRSAF	Version 12 Release 2.2 (Febuary 2019)

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: 201 420-0400
Fax: 201 420-4080
Email sales@relarc.com
Web www.relarc.com

Comments regarding the product, its documentation, and suggested improvements are welcome. A reader comment form is provided at the back of this publication or you can send comments to Comments@relarc.com.

Copyright (c) 1988, 2019 by Relational Architects Intl. (RAI) as an unpublished work – all rights reserved.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 Intl. 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.

z/OS, Db2, IMS, WebSphere MQ and ISPF are software products of International Business Machines Corporation. CA-11 is a trademark of Computer Associates, Inc. Smart/RESTART, Smart/LOGGER, Smart/CAF and Smart/RRSAF are trademarks of Relational Architects Intl.

Ed 19B25

Preface

This publication outlines the features and benefits of Smart/RESTART and Smart/RRSAF. It is intended for analysts, administrators, developers, and managers responsible for evaluating and purchasing software products for the z/OS batch environment.

Smart/RESTART delivers restart capability that is robust, reliable and easy to use. It minimizes the impact of batch job failures without complicated restart logic or cumbersome operational procedures. Smart/RESTART ***guarantees*** that updates to *any* RRS compliant resources (such as those managed by Db2, MQ and IMS) remain synchronized with application storage, sequential file position and VSAM updates. This ensures jobs can resume from a point of consistency (near the point of failure) after abends, recompiles, even system IPLs.

Smart/RRSAF complements Smart/RESTART by letting you run your Db2 applications with regular JCL, so they look and behave like standard z/OS batch jobs. Smart/RRSAF support for RRSAF (Db2 Recoverable Resource Manager Services Attachment Facility) enables your applications to make changes to Db2, IMS, MQ and other RRS compliant resources within a single unit of recovery. Together, Smart/RESTART and Smart/RRSAF comprise the Smart Jobstream Series – an integrated set of products designed to keep your batch workload on schedule and running smoothly. Smart implementation simplifies application development, maintenance and trouble shooting while Smart operations make your nightly batch cycle more predictable, reliable and secure. Your organization benefits from better throughput, improved levels of service and enhanced disaster recovery capability.

Chapter 1 describes the context in which Smart/RESTART operates and discusses the product's concepts and facilities.

Chapter 2 discusses the development of restartable applications in detail. It describes how Smart/RESTART assumes responsibility for the *mechanics* of checkpoint/restart and enumerates the design guidelines that make applications restartable ***in practice.*** Chapter 2 includes a complete and fully annotated example of a restartable Db2/COBOL application and the JCL to run it.

Chapter 3 describes Smart/LOGGER for VSAM, a component of Smart/RESTART that provides automated restart capabilities for applications that access VSAM data as well as RRS compliant resources such as those managed by Db2, IMS and WebSphere MQ. Smart/LOGGER supports VSAM ESDS, KSDS, RRDS and VRRDS files via random, sequential and skip-sequential access.

Chapter 4 describes and illustrates the job management facilities that complement Smart/RESTART's core checkpoint/restart capabilities. These include:

- Job monitoring and prediction
- Job control
- Error recovery and retry

Chapter 5 describes Smart/RRSAF and the benefits of running Db2 batch applications with standard JCL. The chapter also discusses implementation considerations and the value Smart/RRSAF adds to Smart/RESTART in a Db2 environment.

This General Information Manual is augmented by the more extensive Smart Jobstream Series product library which at present includes the following publications:

- Smart Jobstream Series Installation and Customization Guide SJS-001
- Smart/RRSAF User Guide and Reference SJS-002
- Smart/RESTART User Guide SJS-004
- Smart/RESTART Messages Reference SJS-006

Notational conventions

The following notational conventions are used in this 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

Section A

What's New in

Smart/RRSAF and Smart/RESTART

Smart/RRSAF and Smart/RESTART Version 12.2 — Highlights

New Features

Versions 12.2 of Smart/RRSAF and Smart/RESTART provide productivity, scalability and security enhancements designed to improve its role as a high-availability solution for your critical z/OS batch workloads.

Two new members of the DCACNTL library are provided to facilitate the definition of RACF FACILITYs and to grant permission to utilize these facilities to authorized users.

- The SJSJRAC1 member of the DCACNTL library provides prototype JCL with which to issue RACF REDEFINE commands for the protected facilities associated with the Smart Jobstream Series.
- The SJSJRAC2 member of the DCACNTL library provides prototype JCL with which to issue RACF PERMIT commands to allow authorized userIDs to utilize the protected facilities asociated with the Smart Jobstream Series.

The Smart/RRSAF Conversion Facility has been enhanced to support program objects stored in PDSE's, in addition to the support for load modules stored in PDS's.

Altered Features

The profile parameter RRSAF_TERMINATE has been changed to support settings: (AUTO | OFF). RRSAF_TERMINATE previously supported the settings (ALL | NONE | COMMIT | ROLLBACK)

- AUTO is introduced as a synonym for ALL and is the new default. ALL is still accepted as a synonym for AUTO.
- OFF is introduced as a synonym for NONE while NONE is still accepted

- COMMIT and ROLLBACK are both removed as options in the Smart/RRSAF Profile dialog. If the dialog detects COMMIT or ROLLBACK they will be flagged as an error.
- In contrast, Smart/RRSAF will internally convert COMMIT and ROLLBACK to AUTO when either value is detected at at runtime in a pre-existing profile module or RAINPUT parameter file. A warning message will also be issued.

The profile parameter ERROR has had the default changed from RETURN to ABEND.

The Smart/RESTART option IOTRACE is redocumented as a a CSECT flow of control trace. The new TRACE_CF parameter is introduced as a more descriptive keyword. The text of the diagnostic default TRACE_CF on panel SRSTPDD appears as follows:

```
Trace control flow activity to the RFATRACE file
```

This is solely to maintain visual continuity. A control flow trace can now be produced simply by allocating the DDname RFATRACE at runtime. The CSECT level control flow trace is written to the dataset allocated to the RFATRACE file. Both Smart/RRSAF and Smart/RESTART will ignore any IOTRACE or TRACE_CF specification.

Removed Features

- The ASM option has been removed from panel SJS@SLIB
- The TRACE and IOTRACE parameters are both removed from the Smart/RRSAF User Guide and Reference. A control flow trace can now be produced simply by allocating the DDname RFATRACE at runtime. The CSECT level control flow trace is written to the dataset allocated to the RFATRACE file. Smart/RRSAF will ignore any TRACE orIOTRACE specification.
- The set of zIIP_OFFLOAD parameters have been removed from the Smart/RRSAF Profile dialog

Documentation changes - Smart/RRSAF User Guide and Reference

- The set of runtime parameters described in Section 4.7 of the Smart/RRSAF User Guide and Reference pertain to the OS_CHECKPOINT facility described in Section 4.18 of the Guide and Reference. Due to the extremely limited set of circumstances from which restart can successfully occur, the use of OS_CHECKPOINT and its associated parameters are not recommended.
- The Smart/RESTART WLM_ENCLAVE_CREATE description is copied to Smart/RRSAF User Guide and Reference

Documentation changes - Smart/RESTART User Guide

RAI strongly recommends the Smart/RESTART checkpoint datasets be allocated explicitly in the JCL of a restartable jobstep, rather than be allocated dynamically by

Smart/RESTART Version 12.1 — Highlights

Smart/RESTART Version 12.1 provides productivity, scalability and security enhancements designed to improve its role as a high-availability solution for your critical z/OS batch workloads.

Version 12.1 of Smart/RESTART supports all functions available in prior releases of the product plus various functions and enhancements introduced since the advent of Smart/RESTART Version 11.1. In addition, Smart/RESTART Version 12.1 incorporates corrective maintenance for numerous problem reports.

Full support for:

- Enterprise COBOL 6.1.0, IBM Program Product 5655-EC6
- Enterprise PL/I 5.1.0, IBM Program Product 5655-PL5
- Db2 Version 12, IBM Program Product 5650-Db2
- IMS Version 14, IBM Program Product 5751-CS3

Enhanced Functionality

- 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.
- 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.
- 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.
- The Smart/RRSAF Conversion Facility is enhanced to support the replacement of any Db2 attachment stub with any other Db2 attachment. This conversion can occur for selected modules or for an entire library of load modules or program objects.
- New Utility envelope SRSUTIL is introduced to invoke Smart/RESTART Utility services. These services can be invoked as an independent jobstep and/or via dynamic and static calls issued from within a Smart/RESTART application program. Members SRSJUCHK, SRSJUCPY, SRSJULRS and SRSJUMSG of the DCACNTL library contain sample JCL with which to invoke various SRSUTIL services. In addition, the new sample program named SRS8CUTL demonstrates how to invoke SRSUTIL services from within an application program.

Improved diagnostics

- RAI-SNAP is an advanced API service which directs Smart/RESTART to write a formatted dump of its internal control blocks to the destination identified by the DUMPDD parameter.
- Smart/RESTART supports the IBM Debug Tool without restriction to debug LE enabled applications written in COBOL, PL/I and Assembler.

Enhanced security

- The new RACF profile RAI.SRM.CONTROL is introduced to provide more granular control over access to authorized Smart/RESTART resources referenced by Smart/MONITOR
- New profile option EMPTY_FILE_CHECK is introduced to ensure that RSAM files are not empty upon a RESTART run.

Enhanced capacity and reliability

- The Smart/RESTART job identification string is extended from sixteen to thirty two characters
- CKPT_SIZE limit is increased from 32767 to 99999
- Smart/RESTART messages support RSAM files with up to 999,999,999 records
- Smart/RESTART messages support up to 999,999,999 checkpoints
- The new profile option CKPT_HLQ_LONG is introduced to allow the definition of a high level qualifier of up to twenty-two characters in length for dynamically allocated checkpoint datasets. See the Smart/RESTART Reference Manual for a detailed description of this parameter.

1

Smart/RESTART

Concepts and Facilities

Smart/RESTART is a resource manager and syncpoint coordinator that enables your abended batch jobs to resume execution from a checkpoint – near the point of failure. Smart/RESTART can be used in standalone mode or in combination with Db2, WebSphere MQ, IMS and/or any other RRS compliant resource manager.

1.1 What is Smart/RESTART?

Background and Definitions

'Protected resources' are resources so critical to an organization that their integrity must be guaranteed. The organization needs to be able to recover such protected resources in the event they become corrupted due to hardware failure, software failure, human error, catastrophe, etc. Providing integrity requires locking, logging and recovery mechanisms that implement checkpoints (COMMITs) and backouts (ROLLBACKs).

This publication uses the term *resource manager* to refer to authorized subsystems like Db2 and IMS (which control access to database resources) and WebSphere MQ which manages message queues. The recoverable resources managed by these subsystems are termed *protected resources*.

The *set* of changes that must succeed or fail *together* is called a unit-of-work (UOW). A unit-of-work is also referred to as a Logical Unit of Work (LUW) or Unit of Recovery (UR). Section 1.2 discusses units-of-work in detail.

Introduction to Smart/RESTART

Smart/RESTART functions as a resource manager for program working storage, sequential files and VSAM datasets. As such, Smart/RESTART treats application storage and files as protected resources whose integrity must be guaranteed.

Smart/RESTART provides:

- make them permanent)
- a backout service to *restore* protected resources to their previous state

Batch applications may update Db2 tables, IMS databases, MQ queues and other RRS compliant resources as well as update working storage, make changes to VSAM datasets, and read and write sequential files. As such, Smart/RESTART must function as a syncpoint manager that coordinates changes to *multiple* protected resources. Smart/RESTART must ensure that either *all* changes are made or *no* changes are made. Smart/RESTART guarantees that updates to Db2, IMS MQ and/or other RRS compliant resources stay in synch with your program's working storage, sequential files and VSAM datasets. This 'all or nothing' guarantee applies even if the application program, the system, or a participating resource manager fails. Therefore, your applications can *always* restart from a point of consistency.

In its role as resource manager and syncpoint coordinator, Smart/RESTART provides batch applications with functions similar to those performed by CICS for online transactions. CICS is a resource manager for protected resources like VSAM files, temporary storage queues and transient data queues. In addition, CICS applications benefit from the syncpoint coordinating role that CICS assumes when dealing with *multiple* resource managers. Rather than issuing *separate* commit commands to each resource manager, a CICS program issues a single EXEC CICS SYNCPOINT command. CICS then takes responsibility for committing changes to Db2, IMS and MQ as well as its own protected resources. All changes are thus committed or rolled back *together.*

> ***NOTE:*** *CICS* ***must*** *take responsibility for coordinating checkpoints among the various resource managers. Suppose you coded EXEC CICS SYNCPOINT to make permanent the updates to CICS resources and EXEC SQL COMMIT to harden Db2 changes. With two discrete commands, it is possible one could succeed while the other failed – leaving your application with inconsistent data.*

Smart/RESTART provides batch applications with a robust and powerful ***Smart*** checkpoint whose scope encompasses changes to ***all*** resources your batch program accesses. In addition to the Db2, MQ and IMS resources managed by IBM software, a ***Smart*** checkpoint includes the following resources managed by Smart/RESTART:

- Program storage (in COBOL, the WORKING-STORAGE section)
- Sequential input and output files
- VSAM datasets (KSDS, ESDS, RRDS and VRRDS)

A ***Smart*** checkpoint allows a batch application to resume execution from a unit-of-work boundary. There is never a need to restore a file, table or database to its state *before* the job started nor is it necessary to re-run a failed batch job from the beginning. As such, work that has *already* been performed successfully need *never* be re-processed. This is illustrated in Figure 1.1.

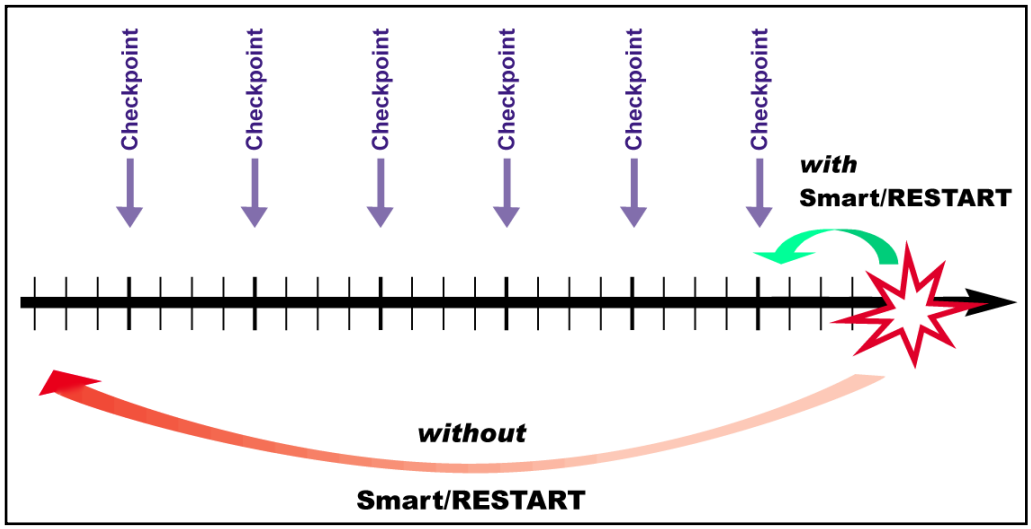


Figure 1.1 Smart/RESTART enables batch jobs to restart from the last checkpoint

Programming with Smart/RESTART

Smart/RESTART eliminates the need for restart logic in your programs. It handles the mechanics of resuming a failed job step. Your batch programs are thus smaller, easier to understand, faster to develop and less prone to bugs.

Smart/RESTART detects when a job is restarted and automatically restores ***all*** protected resources to their state as of the last ***Smart*** checkpoint. Specifically, Smart/RESTART:

- Repositions sequential input and output files
- Restores all or selected areas of program storage
- Backs out uncommitted changes to VSAM datasets

Smart/RESTART can also be configured to automatically handle Db2 resource unavailable conditions and to re-try failing SQL statements when appropriate.

In many cases, restartability can be achieved for a batch program without source code changes. In some cases, re-linking is required to enable a program for restart.

Production Operations with Smart/RESTART

By following the JCL guidelines in this publication and the Smart/RESTART User Guide, a failed batch job step can be restarted by simply re-submitting it. Figure 1.2 enumerates how Smart/RESTART lets you deal with several common failures.

Reason for Batch Job Failure	Response with Smart/RESTART
Job step exceeds CPU time limit	**Re-submit the job step**
Operator cancel	**Re-submit the job step**
x37 Abends (disk space)	**Copy to a new, larger file and re-submit the job step**
Resource manager failure (Db2, IMS or MQ)	**Restore resource manager availability Re-submit the job step**
S0C7 Abend (bad data)	**Fix the data and re-submit the job step**
IPL while the job is running	**Re-submit the job step**

Figure 1.2 Responding to common batch job failures when Smart/RESTART is used

Smart/RESTART allows the failed batch program to be changed and recompiled and its job step resumed from the last ***Smart*** checkpoint. If needed, a job step can even be resumed from a checkpoint *other* than the last one.

Smart/RESTART also provides extensive job management facilities that complement its core checkpoint/restart capabilities. These include:

- Job monitoring (Smart/MONITOR)
- Prediction of job completion (Smart/PREDICTOR)
- Job control
- Error recovery and retry

1.2 Units of Work and Points of Consistency

The goal of Smart/RESTART is to enable you to *resume* a batch application from a point near the failure, rather than re-running it from the beginning. This requires the application to periodically take checkpoints from which processing can resume on restart.

Two concepts are the basis for restartability: the logical unit-of-work and the logical point of consistency.

A unit-of-work represents an application program's changes to resources since the last commit or backout or, for the first UOW, since the beginning of the application. All changes to protected resources made by an application in a unit-of-work must succeed or fail *as a unit.*

A unit-of-work is delimited by a checkpoint. For example, a Db2 batch application requests a checkpoint by issuing an EXEC SQL COMMIT while IMS, WebSphere MQ and Smart/RESTART provide checkpoint services of their own.

The classic example of a unit-of-work is the banking application that transfers funds between your checking and savings accounts. This simple unit-of-work encompasses just *two* changes. Funds are first deducted from the checking account and then added to the savings account. A checkpoint is then requested. If the checkpoint succeeds, the unit-of-work is committed and you have saved for a rainy day.

A new point of consistency is established only after ***both*** changes complete successfully. Should an error prevent adding funds to your savings account, then the deduction from checking must be backed out to re-establish the original point of consistency. In this example, the unit-of-work corresponds to a logical point of consistency. This correspondence is ***crucial*** to correct application design.

To illustrate *incorrect* design, consider a banking application that issues a checkpoint after deducting funds from checking but *before* crediting them to savings. This is not a logical point of consistency since the unit-of-work does not encompass the two changes that comprise the transaction.

The developer is responsible for identifying the point of consistency in application processing and requesting a checkpoint at that time. This is not a task you can delegate to Smart/RESTART since it knows neither your application's requirements nor when application data is in a consistent state.

To reiterate and rephrase a crucial point – ***Proper application design must ensure that a unit-of-work completes at (i.e. results in) a logical point of consistency. Otherwise, the design is incorrect!***

What Smart/RESTART *can* do is establish a point of consistency which is comprehensive in scope. A ***Smart*** checkpoint encompasses ***all*** changes made by an application. Since working storage contents and sequential file position are lost when a batch job fails (or is interrupted), Smart/RESTART saves them at checkpoint-time, restores them at restart-time and ensures they remain in synch with changes to Db2, IMS,VSAM and MQ resources.

Smart/RESTART can do all this *transparently*. In many, if not most cases, application programs need not be aware of Smart/RESTART's existence.

1.3 Features and Benefits of Smart/RESTART

There's less and less time to tolerate failures in z/OS batch production as organizations progress towards `twenty-four by seven' operation. Checkpoint / Restart capability becomes not just insurance but a prerequisite for concurrency. With this in mind, we present some of the features and benefits of Smart/RESTART.

Simplified Application Development: Smart/RESTART eliminates the need for restart logic in your application programs. 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.

Standardized Restart Procedures: The Smart/RESTART architecture supports a variety of programming languages and resource managers, making it easy to adopt as an enterprise-wide restart methodology. Restartable applications can be comprised of multiple load modules within a single job step. Each load module in turn may be comprised of multiple, independently written and compiled source modules.

Minimizes the Impact of Batch Job Failures: Smart/RESTART eliminates the need to rerun abended jobs from the beginning. This ensures processing always proceeds in a forward direction. Smart/RESTART eliminates redundant processing and provides for easier debugging. Backing out changes to protected resources is no longer a potential source of delay. Delays associated with modifying JCL can be eliminated because the *same* JCL can be used in both the initial and restart runs. Your nightly batch cycle becomes more reliable and predictable.

Eliminate Precautionary Backups: In domino fashion, Smart/RESTART reduces the need to back up files and databases – to the extent precautionary copies are made to allow job restart.

Automated Error Handling: Smart/RESTART reduces timeouts, deadlocks and 'resource unavailable' conditions as a cause of job failure. It can automatically retry the failing unit-of-work (as described in Error Recovery and Retry in Section 1.6).

Checkpoint Pacing: Both the checkpoint frequency and interval may be modified or tuned *during* execution – based on factors such as elapsed time, number of records processed, time of day, heuristic code and/or operator request. You can *predefine* different checkpoint frequencies for separate shifts. For example: Does a batch job hold so many database locks that it causes contention with online applications? Rather than cancel it, you can direct Smart/RESTART to take more frequent checkpoints. Smart/RESTART also allows you to issue checkpoints in applications that do not request them by defining checkpoint criteria *external* to the application.

Monitor and Control Restartable Jobs: TSO users can display real time statistical summaries of file I/O, SQL statements and checkpoints in restartable jobs. Authorized users can view and even modify program storage during execution. Job completion times can be predicted and 'behind schedule' warnings can be issued. Jobs can be suspended, either manually or automatically, in response to certain errors. Of course, jobs can also be resumed. Restartable jobs that need to be cancelled can be terminated when their next checkpoint completes so there is no time wasted doing processing that must not only be discarded, but backed out as well.

Restarts Not Limited to Abends: Batch jobs can be restarted after the failing program is changed and recompiled. Restarts can even be performed after the system is IPL'd.

Job Scheduling: Smart/RESTART enhances products like CA-11 and IBM's OPC (that provide job step restart capability) by enabling jobs to resume from a checkpoint within a job step – something a job scheduler alone cannot do.

Security: All Relational Architects products are well behaved from a security standpoint. Security administrators can control access to authorized Smart/RESTART functions on a granular basis via standard RACF, ACF2 and TopSecret definitions. Smart/RESTART verifies a user's privileges by calling SAF (the System Authorization Facility of z/OS) which in turn invokes the security product installed at your site.

1.4 Method of Operation

In many cases, the use of Smart/RESTART can be enabled without source code changes. Your application programs need not call Smart/RESTART directly or even be aware of Smart/RESTART's existence. Typically, Smart/PRECOMPILER (described in the next section) automates the restart enablement process.

If source changes are possible, you can restart enable your applications using the COBOL COPY text and PL/1 INCLUDEs supplied with Smart/RESTART or invoke its call level and EXEC level APIs (application programming interfaces). The EXEC level API is simple to use and is supported by modules written in COBOL and PL/1. EXEC level API requests are denoted by EXEC SRS (analogous to EXEC SQL and EXEC CICS requests). For example, an EXEC SRS COMMIT statement explicitly requests a ***Smart*** checkpoint.

Smart/RESTART services are also available via calls to module RAIAPI. (For example, in COBOL: CALL 'RAIAPI' USING field1, field2). The call level API is available from any programming language which adheres to z/OS calling conventions.

Smart/RESTART receives control whenever your application program requests a checkpoint, either *explicitly* via the Smart/RESTART call level or EXEC level API or *implicitly* via an EXEC SQL COMMIT request (when Smart/CAF is active).

Either way, Smart/RESTART performs a ***Smart*** checkpoint that records a 'snapshot' of your application's protected resources which include both the working storage, sequential files and VSAM datasets managed by Smart/RESTART as well as any resources managed by Db2, MQ and/or IMS. Smart/RESTART coordinates checkpoint processing among all the Resource Managers involved.

> ***NOTE:*** *Smart/RESTART's checkpoint and rollback services are both coordinated and comprehensive in scope.*

Smart/RESTART does not interfere with Resource Manager processing when an application terminates abnormally. For example, Db2 backs out uncommitted changes when an application thread abends.

Smart/RESTART recognizes the restart environment at restart time and automatically restores working storage and sequential file position. If necessary, uncommitted changes to VSAM files are backed out as well. This reestablishes the point of consistency that prevailed at the last successful checkpoint and enables processing to resume cleanly at a unit-of-work boundary.

1.5 Smart/RESTART Architecture and Components

Figure 1.3 illustrates the Smart/RESTART Run Time Architecture. With the exception of Smart/PRECOMPILER, all components and topics discussed in this section and Section 1.6 appear in the Run-time Architecture diagram

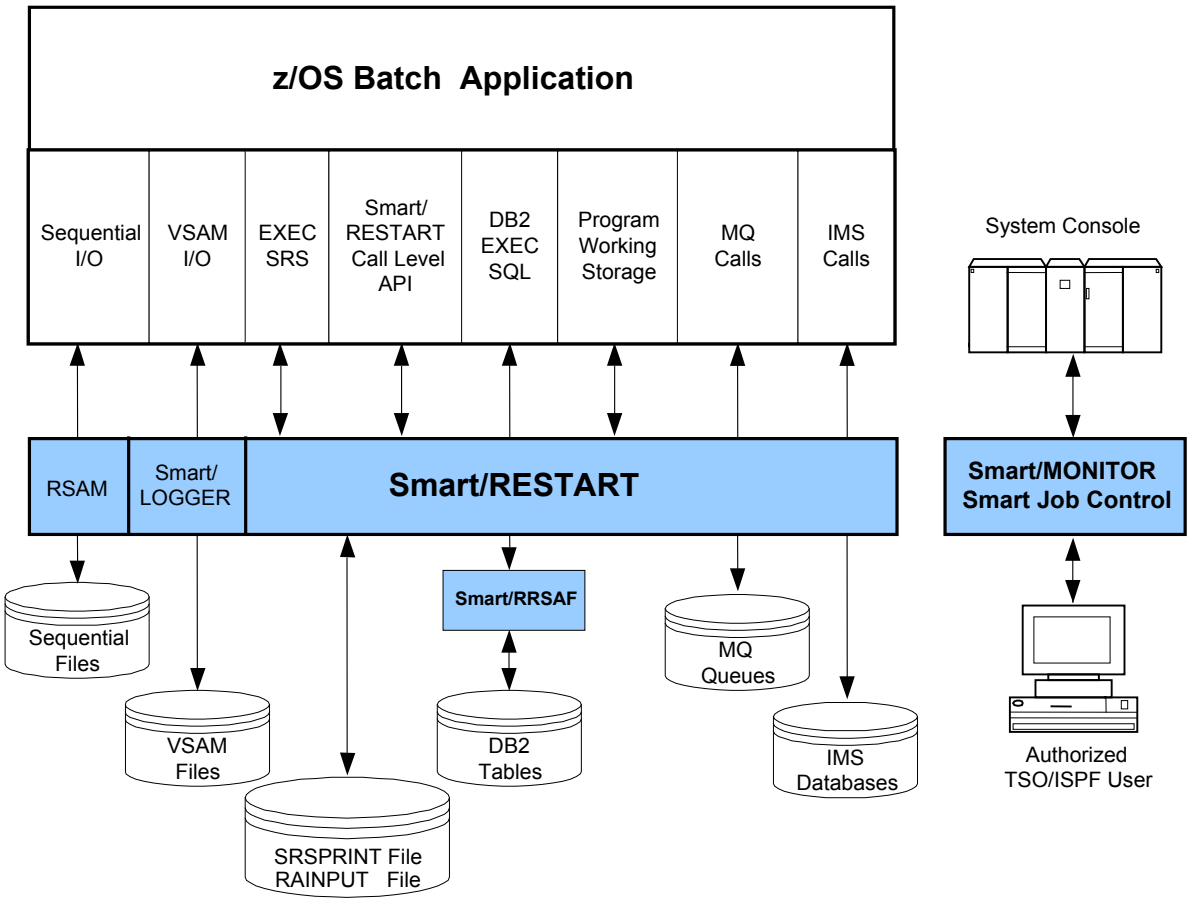


Figure 1.3 Smart/RESTART Run-time Architecture

Smart/PRECOMPILER

Smart/PRECOMPILER can completely automate the restart enablement process. It functions much like the Db2 precompiler and the CICS translator and runs as a job step prior to source code compilation. Smart/PRECOMPILER's main function is to add 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. Additionally, it conducts syntax, validity and consistency checks that help reduce errors and speed development. Smart/PRECOMPILER supports COBOL and PL/1.

Repositionable Sequential Access Method – RSAM

Smart/RESTART allows I/O to sequential files to be performed using verbs *native* to the programming language in which your application is coded. For example, COBOL programmers can code standard SELECT and FD statements and use native OPEN, CLOSE, READ and WRITE verbs. You need not define repositionable sequential files as IMS GSAM databases in order to achieve restartable operation.

Behind the scenes, Smart/RESTART uses its ***Repositionable Sequential Access Method*** (RSAM) to access sequential files that must be repositioned on restart. There is no performance penalty with RSAM. In fact, RSAM's optimized, parallel I/O techniques actually *outperform* native COBOL sequential file access.

By default, all sequential files are accessed via RSAM. However, you can tell Smart/RESTART to use normal QSAM access for files that need not be repositioned. RSAM is also used to sequentially access VSAM datasets.

Smart Attachment Facility

Smart/RRSAF lets you run your Db2 batch applications using regular JCL (that is EXEC PGM=MYPROG) and provides an alternative to running batch Db2 programs under the TSO Terminal Monitor Program. With Smart/RRSAF, Db2 batch job steps look and behave like regular z/OS batch job steps. This allows job step completion codes to accurately reflect the outcome of application processing rather than TSO DSN command processing. Moreover, SMF accounting is recorded under the application program name rather than being lumped together under IKJEFTxx (the TSO Terminal Monitor Program). Smart/RRSAF support for RRSAF (Recoverable Resource Manager Services Attachment Facility) enables your applications to make changes to Db2, IMS and MQ resources within a single unit of recovery. Smart/RRSAF and its many benefits are described in Chapter 5.

Smart/LOGGER for VSAM

Smart/LOGGER is a component of Smart/RESTART that treats VSAM files as protected resources whose integrity must be guaranteed. Smart/LOGGER saves 'before' images of records such that *uncommitted* VSAM inserts, updates and deletes can be backed out if necessary.

In conjunction with Smart/RESTART, Smart/LOGGER guarantees that changes to VSAM data stay in synch with your program's working storage and sequential files as well as with updates to Db2,

MQ and/or IMS resources. Smart/LOGGER supports ESDS, KSDS, RRDS and VRRDS file types via random and skip-sequential access. Chapter 3 has more on Smart/LOGGER for VSAM.

1.6 Smart/RESTART Run-time Smart/MONITOR and Smart/PREDICTOR

Smart/MONITOR is a set of ISPF dialogs that let you monitor, tune and control your restartable applications in real time. Smart/MONITOR maintains information about checkpoint activity, I/O counts by file, execution counts by SQL statement and entry/exit counts by module. With Smart/MONITOR you can also view and modify program storage during execution.

Smart/PREDICTOR estimates job completion times and can issue early warnings that allow you to take remedial action to help meet production schedules. Smart/MONITOR and Smart/PREDICTOR are described in Chapter 4.

Job Control

Smart/RESTART's job control facilities enable MVS operators and authorized TSO users to vary options such as checkpoint frequency and checkpoint interval while a restartable application is running. This lets you adjust execution speed and resource contention in accordance with changing conditions – without having to cancel the job. Authorized personnel can also quiesce and suspend jobs. They can even drain the system of restartable jobs with a single command.

Error Recovery and Retry

Smart/RESTART provides a recovery/retry facility that can automate the resolution of problems attributable to user specified conditions such as contention and bad data. For example, automated recovery/retry can reduce or eliminate timeouts, deadlocks and resource unavailable conditions as a cause of batch application failure.

The ***Smart*** ROLLBACK service reestablishes the point of application consistency that prevailed at the last successful checkpoint to allow the failing operation and current unit-of-work to be retried. The scope of the ***Smart*** ROLLBACK service includes both the working storage, sequential files and VSAM datasets managed by Smart/RESTART as well as any protected resources managed by Db2, MQ and/or IMS.

Alternatively, Smart/RESTART can suspend the job while the problem is resolved manually. For more on error recovery and retry, see Chapters 42 and 43 of the Smart/RESTART User Guide.

Run-time Requirements

For a job step to be restartable, Smart/RESTART requires an external checkpoint medium in which to record snapshots of your application's working storage and sequential file position. The checkpoint medium can be either a disk dataset, a Db2 table or a tape file. Relational Architects recommends the use of VSAM Linear Datasets to minimize resource contention and optimize performance. Smart/RESTART can dynamically allocate these disk datasets or you can define them in your JCL.

Additionally, Smart/RESTART sends output messages to the SRSPRINT file and reads input parameters from an optional RAINPUT file. Figure 2.4 in Chapter 2 illustrates sample Smart/RESTART JCL.

1.7 Advanced Topics

Object Transparency

Smart/RESTART's Object Transparency Mode provides restart support for applications which can not be modified (such as software supplied as object code only). These programs can be endowed with restart capability – without changing source code or recompiling. Object Transparency mode is restricted to pure COBOL and COBOL/Db2 applications whose COMMIT logic and vital working storage are defined in the main module. Object Transparency is invoked by changing the JCL EXEC statement to invoke an SRS front-end program.

Point-in-time recovery

Normally batch jobs are restarted from the *last* checkpoint prior to the point of failure. Point-in-time recovery allows you to restart an application from *any* checkpoint for which checkpoint records exist. Run-time options and your choice of checkpoint medium govern the number of checkpoints from which restart is possible.

Customization

Smart/RESTART provides interfaces to user written exit routines that let you customize and extend Smart/RESTART processing. Discrete exits can be invoked at job step initialization, checkpoint time, job step termination and when RSAM reads a repositionable file.

For example, site written exits called at program initialization and termination time might be used to turn performance-monitoring on and off whereas a checkpoint exit might obtain Db2 locking data to determine whether Smart/RESTART should honor the application's checkpoint request.

1.8 Installation Prerequisites

The Smart Jobstream Series has the following software prerequisites:

- z/OS – all releases whose IBM support status is current
- ISPF – all releases whose IBM support status is current
- TSO/E REXX
- Smart/PRECOMPILER – all releases of TSO/E whose IBM support status is current
- All releases of COBOL, PL/1, C and Assembler – all releases whose IBM support status is current
- If your application accesses *just* Db2 *or* IMS *or* WebSphere MQ resources – any releases whose IBM support status is current
- Any release of Db2 and IMS supported by IBM can be used by applications that access both Db2 and IMS resources.

The following are *minimum* release levels for prerequisite software when RRS (the Recoverable Resource Manager Services component of z/OS) is used to co-ordinate changes across *multiple* resource managers. Subsequent releases may also be used.

- When Db2 is one of several resource managers participating in a unit-of-work, then RRSAF must be used.
- WebSphere MQ is required when MQ is one of several resource managers participating in a unit-of-work.
- Use of Db2, WebSphere MQ and IMS in the *same* application requires the RRS enabled versions of these products in a z/OS environment.

Smart/RESTART's Repositionable Sequential Access Method requires the Relational Architects Server Address Space to be active in order to support native I/O verbs such as COBOL's READ, WRITE, OPEN and CLOSE.

1.9 Related Publications

This General Information Manual is augmented by the more extensive Smart Jobstream Series product library which at present includes the following publications:

Smart Jobstream Series Installation and Customization Guide	SJS-001
Smart/RRSAF User Guide and Reference	SJS-002
Smart/RESTART User Guide	SJS-004
Smart/RESTART Messages Reference	SJS-006

2

Developing Restartable Applications

Smart/RESTART assumes responsibility for the *mechanics* of checkpoint/restart. This includes saving and restoring program working storage, repositioning sequential files at restart time and logging updates to VSAM datasets so uncommitted changes can be backed out if the unit-of-work fails to complete successfully. Smart/RESTART also assumes responsibility for keeping working storage, sequential file position and changes to VSAM datasets synchronized with Db2, MQ and IMS activity.

However, in order for your applications to be restartable ***in practice***, you must do the following:

- Divide the processing into units-of-work and request checkpoints when they are complete. These checkpoints represent ***'points of consistency'*** from which processing can *resume* upon restart.
- On restart, the application's flow of control must be such that it ***resumes*** execution at a unit-of-work boundary (see Figure 2.1).
- *Pre-allocate* sequential files that need to be repositioned on restart in a separate jobstep. Then allocate the repositionable files within the restartable jobstep with DISP=SHR or DISP=OLD. This allows you to resubmit the *same* JCL should restart be necessary.

The topics addressed in this chapter are covered in more detail in Chapter 2 of the Smart/RESTART User Guide.

2.1 Pseudo-Code Example

Figure 2.1 illustrates a *pseudo-coded* restartable application while Section 2.2 has a complete Db2 / COBOL program.

```
Do Initialization
   (re)open sequential files and VSAM datasets
End

Do while (more input)
   Do Process one or more transactions
      Read from input file
      Update protected resources
      Write to output file
   End

   If Unit-of-Work completes successfully
   Then
      Commit Unit-of-work
   Else
      Roll back Unit-of-work
   Endif
End
```

Figure 2.1 Pseudo-Coded Restartable Application

The innermost Do-End block defines the processing that comprises a unit-of-work (which can consist of one or more transactions). A commit is issued whenever a unit-of-work completes successfully whereas a rollback is requested if it fails.

The code is not aware of Smart/RESTART. Nonetheless, Smart/RESTART receives control on commit and rollback. For a commit request, Smart/RESTART saves an execution-time 'snapshot' of your application's working storage, VSAM updates and sequential file position that is *guaranteed* to be in synch with the unit-of-work maintained by Db2, MQ and/or IMS. Smart/RESTART also gets control at initialization time and determines if this is a new run or a restart run.

No special restart logic is needed in the application program because during a restart run Smart/RESTART automatically:

Repositions sequential files to their position at the last checkpoint
Restores working storage contents to their values as of the last checkpoint
Backs out uncommitted changes to VSAM files (if not backed out at the time of failure)
Synchronizes the above with any Db2, MQ and/or IMS, resources the application may have accessed
Signals a restart is in progress

2.2 Sample Db2/COBOL Application

This section presents a complete COBOL/Db2 batch program that illustrates how restartable applications can be developed without *any* reference to Smart/RESTART in the code. Programs such as this can be made restartable by Smart/PRECOMPILER which adds the required data structures and procedural code to restart-enable the program.

The sample program reads a sequential file of transactions and inserts each one into a Db2 table. In this program, ten transactions (somewhat arbitrarily) comprise a unit-of-work, after which a checkpoint is requested. If an error is detected, the current unit-of-work is backed out, an error message is issued and the program ends.

The only resource manager this program is aware of is Db2. As such, the program issues EXEC SQL COMMIT and EXEC SQL ROLLBACK requests. When Smart/RRSAF is used, EXEC SQL COMMIT requests a ***Smart*** checkpoint whose scope includes sequential file position as well as WORKING-STORAGE.

The sample program has no dedicated restart logic and none is required. Smart/RESTART *automatically* repositions the input file and restores WORKING-STORAGE on a restart run so that processing continues from the last ***Smart*** checkpoint. Smart/RESTART's Repositionable Sequential Access Method transparently manages I/O to files whose position must be reestablished at restart-time. Thus, the program performs I/O using native COBOL READ and WRITE verbs. The ability to restart enable an application *without changes to source code* is one of Smart/RESTART's major benefits.

The bold, italic numbers within parentheses correspond to the numbered annotations which follow the program.

```
ID DIVISION.
PROGRAM-ID.  PROGRAM1.
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  HOST-VARIABLES.
    05  HOST-VARIABLE1            PIC S9(5) COMP VALUE 0.
    05  HOST-VARIABLE2            PIC S9(5) COMP VALUE 0.
    05  HOST-VARIABLE3            PIC S9(5) COMP VALUE 0.
EXEC SQL INCLUDE SQLCA END-EXEC.
01  VITAL-STORAGE.
    05  TRANSACTION-COUNT         PIC S9(5) COMP VALUE 0.
01  NON-CRITICAL-FIELDS.
    05  PROCESSING-FLAG           PIC X(1) VALUE SPACE.
        88  SUCCESSFUL-PROCESSING VALUES ARE 'S' 'E'.
        88  END-OF-FILE           VALUE 'E'.
        88  IO-ERROR              VALUE 'I'.
        88  INSERT-ERROR          VALUE 'P'.
        88  PROCESSING-ERROR      VALUES ARE 'I' 'P'.
    05  INPUT-FILE-STATUS         PIC 9(2) VALUE ZEROS.
PROCEDURE DIVISION.
   OPEN INPUT INPUT-FILE                                    <=== (1)
    IF INPUT-FILE-STATUS > 0 THEN                           <=== (2)
        DISPLAY 'ERROR OPENING INPUT FILE=' INPUT-FILE-STATUS
        GOBACK.
    PERFORM A100-PROCESS-UNIT-OF-WORK THRU A100-EXIT        <=== (3)
        UNTIL END-OF-FILE  OR PROCESSING-ERROR.

    IF SUCCESSFUL-PROCESSING DISPLAY 'SUCCESSFULL END'.
    IF IO-ERROR DISPLAY 'I/O ERROR READING INPUT FILE'.
    IF INSERT-ERROR DISPLAY 'INSERT ERROR TO Db2 TABLE'.
    GOBACK.

A100-PROCESS-UNIT-OF-WORK.                                  <=== (4)
   MOVE 0 TO TRANSACTION-COUNT.
   PERFORM B200-PROCESS-ONE-TRANSACTION THRU B200-EXIT
      UNTIL TRANSACTION-COUNT = 10 OR
      END-OF-FILE  OR  PROCESSING-ERROR.
    IF SUCCESSFUL-PROCESSING
        EXEC SQL COMMIT END-EXEC                            <=== (5)
    ELSE
       EXEC SQL ROLLBACK END-EXEC.                          <=== (6)
A100-EXIT. EXIT.

B200-PROCESS-ONE-TRANSACTION.
    MOVE SPACES TO PROCESSING-FLAG.
    READ INPUT-FILE INTO HOST-VARIABLES AT END
       MOVE ‘E’ TO PROCESSING-FLAG
       GO TO B200-EXIT.
    IF INPUT-FILE-STATUS NOT = 0 THEN
       MOVE 'I' TO PROCESSING-FLAG
       GO TO B200-EXIT.
    EXEC SQL INSERT INTO SOME.TABLE VALUES
       (:HOST-VARIABLE1, :HOST-VARIABLE2, :HOST-VARIABLE3) END-EXEC.
    IF SQLCODE < 0 THEN
        MOVE 'P' TO PROCESSING-FLAG
    ELSE
        MOVE 'S' TO PROCESSING-FLAG
        ADD 1 TO TRANSACTION-COUNT.
B200-EXIT. EXIT.
```

Figure 2.2 Sample Db2 / COBOL Batch Program

(1) In the restart environment, Smart/RESTART automatically repositions sequential input and output files when they are reopened.

(2) Smart/RESTART's Repositionable Sequential Access Method reports the outcome of I/O operations through the standard FILE STATUS data item defined in the FILE CONTROL section.

(3) 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 always begins at a unit-of-work boundary.

(4) A unit-of-work is ten transactions. It ends either when all ten transactions have been processed, there is no more input, or an error is detected.

(5) In response to the EXEC SQL COMMIT request, Smart/RESTART performs a comprehensive, ***Smart*** checkpoint. Smart/RESTART intercepts the COMMIT and saves a 'snapshot' of application storage and sequential file position before passing it to Db2. Only by encompassing Db2 changes, working-storage and file positioning can the ***Smart*** checkpoint establish a *true* point of consistency from which successful restart is possible.

(6) In response to the EXEC SQL ROLLBACK request Smart/RESTART performs a comprehensive, ***Smart*** rollback. In addition to passing the rollback request to Db2, Smart/RESTART also restores working storage and repositions the input sequential file to their state as of the last ***Smart*** checkpoint.

Although a program does not have to be aware of Smart/RESTART to be restartable, there are times when processing specific to an initial run or a restart run may be required. Smart/RESTART sets the COBOL special register RETURN-CODE to a value of +2001 to identify a restart run. Code such as the following can be inserted at the beginning of PROCEDURE DIVISION.

```
IF RETURN-CODE = +2001 THEN
    DISPLAY 'THIS IS A RESTART RUN OF PROGRAM1'
ELSE
    DISPLAY 'THIS IS THE INITIAL RUN OF PROGRAM1'.
```

Alternatively, COBOL programmers can refer to the condition names FCA-RESTART-RUN and FCA-ORIGINAL-RUN to distinguish between a restart and initial run. These two condition names are defined in the Function Communications Area (FCA), a data structure that is automatically inserted by Smart/PRECOMPILER. Assembler, C and PL/1 programs can also test the status field in the FCA to distinguish a restart run from an initial run.

To keep this example simple, program PROGRAM1 stops processing when an error is encountered. Alternatively, the application could note occasional errors and continue processing without effecting restartability. For example, if error messages are written to a sequential output file, then the error file is simply another one that Smart/RESTART automatically repositions.

While this program ends normally when it detects a processing error, many developers choose to issue a user abend when an errors occurs. Smart/RESTART supports either strategy. Smart/RESTART gets control after an abend and performs a ***Smart*** rollback that keeps all resources in a consistent state.

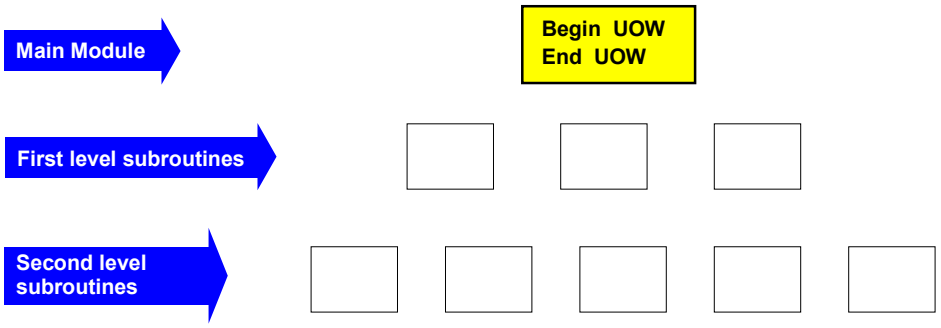
2.3 Multi-module Applications

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 precompiled 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.*** 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.

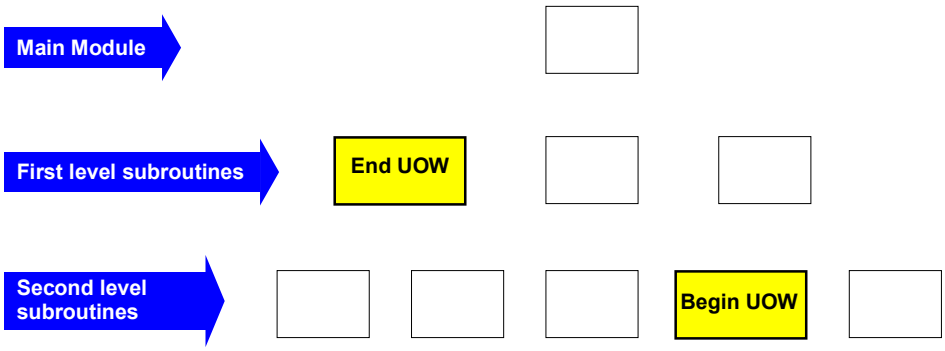
In contrast, Relational Architects *strongly recommends* that you define the Logical Unit of Work in the application's main module as illustrated below. This promotes clarity as to where the unit-of-work begins and ends, and allows developers to state with confidence that an application is logically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

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.

RIGHT WAY to define the unit-of-work



WRONG WAY to define the unit-of-work



2.4 Checkpoint Pacing and Performance

The ***Smart*** checkpoint service, although highly efficient, has a measurable CPU and I/O cost that can delay a job. However, checkpoints also release locks which promotes concurrency. The checkpoint frequency you choose represents a tradeoff between concurrency (more frequent checkpoints) and execution speed (fewer checkpoints).

For example, a batch application contending with online transactions for access to protected resources should take checkpoints more often than an application with exclusive access, whereas a batch job running behind schedule should take very few checkpoints to complete as soon as possible.

The sample program in Figure 2.2 requests a checkpoint after ten transactions, regardless of run-time considerations. Smart/RESTART's checkpoint pacing feature lets you control checkpoint frequency *during execution* by governing whether an application's checkpoint request is actually processed. Each checkpoint an application issues is in fact a *logical* request which, when honored, results in a *physical* checkpoint. A logical checkpoint that Smart/RESTART *does not* honor is effectively a no-op. *Note that restart always occurs from the point of consistency associated with a* ***physical*** *checkpoint.*

You can ***'pace'*** checkpointing dynamically at run time by varying the number of logical checkpoint requests Smart/RESTART must receive before it physically issues a ***Smart*** checkpoint. The decision to issue a physical checkpoint can be based on factors such as elapsed time, the number of logical checkpoint requests received, time of day, heuristic code and/or operator request. Smart/RESTART also lets you *predefine* different checkpoint frequencies for separate shifts.

2.5 Smart/RESTART Modes of Operation

Smart/RESTART supports three distinct modes of restart enablement and operation.

Precompiler Transparency Mode

In Precompiler Transparency mode, the existence of Smart/RESTART is transparent to the application (as was illustrated in the sample COBOL/Db2 program in Figure 2.2). Smart/PRECOMPILER modifies your source program to include the data structures and procedural code needed to enable restartable operation. Smart/PRECOMPILER runs as a discrete jobstep in your program preparation jobstreams and supports COBOL and PL/1 programs that access Db2, MQ and/or IMS resources.

Smart/PRECOMPILER automatically enables your applications for restart, provided the guidelines at the beginning of this chapter are followed. Relational Architects recommends that the main module of each restartable application be processed by Smart/PRECOMPILER. We recommend this whenever program source code is available and program preparation is possible. Figure 2.4 illustrates the JCL used to run a restartable application in Precompiler Transparency mode.

Manual Mode

In this mode, Smart/RESTART services are requested explicitly via COBOL COPY text and PL/1 INCLUDEs or by using a call level or EXEC level API. EXEC SRS statements denote requests which Smart/PRECOMPILER *translates* into compilable code while calls to module RAIAPI request Smart/RESTART services through the call level API.

Manual mode provides additional services, not available in Precompiler Transparency mode, such as dynamic file allocation and the ability to change the checkpoint frequency from within the application.

The EXEC level API is supported for COBOL and PL/1 programs while the call level API is supported by any programming language which adheres to z/OS calling conventions. Chapter 2 of the Smart/RESTART User Guide discusses restart enablement via COBOL COPY text and PL/1 INCLUDEs while Chapters 10 and 11 discuss the EXEC level API and the call level interface respectively. Figure 2.4 illustrates the JCL used with both Manual and Precompiler Transparency modes.

> ***NOTE:*** *Multi-module applications may employ more than one mode of Smart/RESTART support. For example, an application might be a composite of a main module using Precompiler Transparency Mode and several subprograms that explicitly request Smart/RESTART services through its EXEC and call level API's.*

Object Transparency Mode

Smart/RESTART's Object Transparency mode provides robust and efficient restart support for applications which *cannot* be modified, such as software supplied as object code only. Object Transparency mode is restricted to COBOL/Db2 applications whose COMMIT logic and vital working storage are defined in the main module. Such programs can be endowed with restart support *without the need to* change source code or recompile. JCL for use with Object Transparency mode appears in Figure 2.5. Chapter 12 of the Smart/RESTART User Guide describes Object Transparency in detail.

2.6 Sample Restartable JCL

Figure 2.4 illustrates the JCL used to run the restartable COBOL/Db2 sample program in Figure 2.2. This program uses ***Precompiler Transparency*** mode. The same JCL also applies to applications that enable restart in ***Manual*** mode via COPY or INCLUDE text or by explicitly issuing calls and/or EXEC level requests to Smart/RESTART. The following notes relate to the numbered statements in the figure.

```
//STEP1  EXEC PGM=PROGRAM1                                  (1)
//STEPLIB  DD DSN=smart.restart.DCALOAD,DISP=OLD             (2)
//SRSCHECK DD DSN=smart.restart.checkpt.file,DISP=OLD        (3)
//INFILE   DD DSN=your.input.file,DISP=OLD                   (4)
//RAINPUT  DD *                                             (5)
  CKPT_FREQ(50)
//SYSOUT   DD SYSOUT=*
//SRSPRINT DD SYSOUT=*
```

Figure 2.4 JCL to run in Manual or Precompiler Transparency Modes

(1) The program is invoked directly via a JCL EXEC statement. This means jobstep completion codes accurately reflect the outcome of processing and dataset dispositions behave as expected and intended.

(2) The STEPLIB DD statement defines the Smart/RESTART load library and executable code.

(3) The SRSCHECK DD statement defines the Smart/RESTART checkpoint dataset. ***Preallocating*** the checkpoint dataset allows you to resubmit the job for restart *without changing the JCL*.

(4) The INFILE DD statement defines a repositionable, sequential input file

(5) The RAINPUT DD statement supplies *optional* run-time parameters to Smart/RESTART. The CKPT_FREQ(50) parameter tells Smart/RESTART to externalize one physical checkpoint for every 50 logical checkpoint requests issued by the application.

> ***NOTE:*** *With Smart/RRSAF, the Db2 plan name by default is the same as the program name while the default Db2 subsystem is defined at installation time. Alternatively, the Db2 subsystem and application plan names can be specified explicitly as RAINPUT parameters.*

Figure 2.5 illustrates JCL to run the restartable COBOL/Db2 program from Figure 2.2 in ***Object Transparency*** mode. The JCL EXEC statement invokes an SRS front-end program which is passed the name of the restartable application (PROGRAM1) as a parameter.

```
//STEP1  EXEC  PGM=SRS,PARM='PROGRAM1'
//STEPLIB  DD DSN=smart.restart.DCALOAD,DISP=OLD
//SRSCHECK DD DSN=smart.restart.checkpt.file,DISP=OLD
//INFILE   DD DSN=your.input.file,DISP=OLD
//RAINPUT  DD *
  CKPT_FREQ(50)
//SYSOUT   DD SYSOUT=*
//SRSPRINT DD SYSOUT=*
```

Figure 2.5 Object Transparency Mode JCL

Although job step completion codes accurately reflect the outcome of processing Object Transparent applications and dataset dispositions behave as expected and intended, SMF charge-back accounting for all Object Transparent applications is recorded under the name of the SRS front-end program.

The JCL in Figures 2.4 and 2.5 assume the use of Smart/RRSAF (another product in the Smart Jobstream Series) which is discussed in Chapter 5.

Both JCL examples also assume that the Smart/RESTART load library is in the system linklist. If this is not the case, then a STEPLIB DD statement is required that references the Smart/RESTART load library.

Both JCL examples show the RAINPUT DD statement supplying runtime data *instream* but RAINPUT can also reference a sequential file or PDS member. When no runtime overrides are required, the RAINPUT DD statement can be omitted.

Although not illustrated in the COBOL/Db2 sample program in Figure 2.2, restartable programs can accept parameters. In Precompiler Transparency and Manual modes, parameters are passed in the conventional way, as in

```
//STEP1  EXEC PGM=PROGRAM1,PARM='sample parameters'
```

In Object Transparency mode, parameters for the restartable application are passed to the SRS front-end program *following* the name of the restartable application load module, like so:

```
//STEP1  EXEC  PGM=SRS,PARM='PROGRAM1,sample parameters'
```

2.7 Identifying Vital Working Storage to Smart/RESTART

Typically, it is not necessary to save a program's working storage in its entirety at commit time. Instead, just the flags, counters, key values and other fields crucial to processing need be saved at checkpoint time and restored at restart time.

Developers typically need not identify specific work areas to save and restore. However, when working storage is very large, performance can be improved if vital work areas are identified and grouped together and Smart/RESTART is directed to save just these areas.

> ***NOTE:*** *There is no limit to the* ***amount*** *of application storage which can be checkpointed, nor the* ***number*** *of discrete work areas which can be saved and restored.*

In Precompiler Transparency Mode, Smart/PRECOMPILER generates code to save and restore work areas based on installation time specifications. The default is *all* working storage but installers can identify by name fields and structures that Smart/PRECOMPILER should automatically save and restore.

In Manual mode, you can use the EXEC SRS BEGIN SAVE and EXEC SRS END SAVE statements to delimit the beginning and end of vital work areas as illustrated in Figure 2.6. Equivalent functionality is also available via the call level API as well as via COBOL COPY text and PL/1 INCLUDEs. In Object Transparency mode, run time parameters are available to control the storage saved at checkpoint time.

```
 WORKING-STORAGE SECTION.

 EXEC SRS BEGIN SAVE END-EXEC.
 Ø1  WS-VITAL-STORAGE.
     Ø2  WS-UOW-COUNT             PIC S9(4) COMP VALUE Ø.
     Ø2  WS-TRANSACTION-COUNT     PIC S9(4) COMP VALUE Ø.
 EXEC SRS END SAVE END-EXEC.

 Ø1  WS-NONVITAL-STORAGE.
     Ø2  WS-VARIABLE1             PIC X VALUE "A".
     Ø2  WS-VARIABLE2             PIC X VALUE "B".
     Ø2  WS-VARIABLE3             PIC X VALUE "C".
...
```

Figure 2.6 Identifying COBOL storage to save and restore in Manual Mode

3

Smart / LOGGER for VSAM

Smart/LOGGER is a component of Smart/RESTART that treats VSAM files as protected resources whose integrity must be guaranteed. Smart/LOGGER saves 'before' images of records such that uncommitted VSAM inserts, updates and deletes can be backed out if necessary.

In conjunction with Smart/RESTART, Smart/LOGGER assures that changes to VSAM data stay in synch with your program's working storage and sequential files as well as with updates to Db2, IMS and/or MQSeries. Smart/LOGGER supports ESDS, KSDS, RRDS and VRRDS file types via random and skip-sequential access.

> ***NOTE:*** *Sequentially accessed VSAM files need only be repositioned, not logged. As such, they are supported through RSAM (Smart/RESTART's Repositionable Sequential Access Method.)*

Smart/LOGGER is independent of CICS but is tightly coupled with and controlled by Smart/RESTART. Additional information on Smart/LOGGER appears in Chapter 60 of the Smart/RESTART User Guide.

3.1 Key features of Smart / LOGGER

Smart/LOGGER journals changes in a dynamically allocated VSAM LDS (linear dataset) using the most efficient access method available in the z/OS environment. Smart logging occurs at speeds comparable to virtual storage paging. This provides a significant advantage over IMS or CICS logging.

Smart/LOGGER dynamically backs out pending changes to VSAM data when your application

- requests a ROLLBACK
- abends
- is cancelled by the operator

***NOTE**: RAI strongly recommends that restartable jobsteps be cancelled through the use a Smart/RESTART CANCEL, FORCE or QUIESCE command because use of MVS CANCEL or FORCE commands can corrupt data or create inconsistent data. In contrast, the use of the Smart/RESTART's CANCEL, FORCE and QUIESCE commands ensure data integrity is preserved.*

The Smart/RESTART CANCEL and QUIESCE commands avoid the need to discard successfully processed work and eliminate the costly and time consuming rollback of an inflight unit-of-work while the Smart/RESTART FORCE command immediately triggers an abend such that any processing performed during the current unit-of-work is discarded and must be backed out. In contrast to FORCE, the QUIESCE and CANCEL commands gracefully cancel an application at a unit-of-work boundary – with no wasted processing

Smart/LOGGER is also invoked at the start of a restart run to insure that uncommitted changes to VSAM data are backed out.

Smart/LOGGER requires no changes to application code or JCL. It operates with COBOL, PL/1, Assembler, C and other languages that adhere to z/OS linkage conventions.

3.2 How it Works

Smart/LOGGER receives control before every I/O request directed to a recoverable VSAM file and acts as follows:

- For an insert request, a record key or RRN is logged to enable a delete operation to undo the insert, if necessary.
- For a delete, an image of the existing record is logged to permit an insert operation to reverse the delete.
- For an update, a before image is logged so the update can be reversed and the original record restored.

All logging is performed before changes are applied to the dataset (the Smart/LOGGER write-ahead feature). This guarantees that backout can occur in the event an application terminates abnormally.

You can control which VSAM files are recoverable via RAINPUT parameters. By default, Smart/LOGGER treats all VSAM files as recoverable.

4

Smart / MONITOR

Smart/MONITOR (a component of Smart/RESTART) is an ISPF dialog that helps you monitor, tune and control your restartable applications in real-time. Smart/MONITOR maintains information about checkpoint activity, I/O counts by file, execution counts by SQL statement and entry/exit counts by module. Smart/MONITOR can also adjust various aspects of currently executing jobs and even modify data 'on the fly'.

A companion facility, Smart/PREDICTOR, estimates job step completion times and issues warnings when it senses a job may run too long. Part IV of the Smart/RESTART User Guide provides detailed information on Smart/MONITOR and Smart/PREDICTOR.

4.1 Smart / MONITOR Panels

Smart/MONITOR presents a hierarchy of ISPF panels that display job information in progressively greater detail. The first panel displayed is the Job Summary illustrated in Figure 4.1. The Job Summary lists restartable applications that are *currently* executing as well as applications that have failed and are pending restart. You can customize and filter the list of jobs displayed on the Job Summary panel.

```
S/MONITOR ------------------ Smart/RESTART Job Summary ------- Row 1 to 7 of 7
Command ===>                                                   Scroll ===> PAGE
-----------------------------------------------------03/03/2009 11:50 RAI042 SRM
Checkpoint Synchronization Db2 Table Owner ID: RAITST   Db2 Subsystem ID: Db2T
-----------------------------------------------------SORT = St-Date/D, St-Time/D
    ALL           ALL      ALL        ALL      ---------- Checkpoint ----------
Cmd Jobname  Job# Status   St-Date    St-Time  Date       Time      ID     Freq

... JOBSAMP4 9464 SOC7     2009-03-03 11.17.19 2009-03-03 11.17.21 0000003 0001
... RAI7SRMR 9364 RUNNING  2009-03-03 11.01.10 2009-03-03 11.49.11 0000106 0005
... JOBSAMP2 7234 S222     2009-03-03 18.42.20 2009-03-03 18.44.11 0000006 0005
... RAI8SRMR 6556 RUNNING  2009-03-03 17.57.17 2009-03-03 20.04.08 0000026 0100
... RAI7VSTR 6644 SOC7     2009-03-03 16.25.38 2009-03-03 16.25.41 0000003 0001
... RAIJOB7  4269 TBLINIT  2009-03-03 12.39.22 2009-03-03 12.39.22 0000000 0001
... RAI321   4264 S13E     2009-03-03 12.35.02 2009-03-03 12.35.29 0000003 0001
```

Figure 4.1 Job Summary Panel

Smart/MONITOR lets you drill down from the Summary for additional job details. For example, Figure 4.2 shows the Repositionable File I/O Summary which reports I/O statistics and completion percentages for files managed by RSAM - Smart/RESTART's Repositionable Sequential Access Method. You can also enter a line command for abended jobs to invoke the Abend Recovery dialog that helps you analyze the failure and select a recovery strategy.

Figure 4.2 illustrates how Smart/RESTART dynamically determines the number of input file records so that Smart/PREDICTOR (described in Section 4.4) can factor current input file position into its completion time estimate for the job.

Figure 4.3 illustrates the SQL Access Summary which provides statistics about SQL statements executed by all the Db2 packages and DBRMs bound within the application plan.

```
-------------- Smart/RESTART - Repositionable File I/O Summary Row 1 to 3 of 3
Command ===>                                                  Scroll ===> PAGE
SRM061I B - Browse, E - Edit most recent record.
Job/Jobstep Information
   Job Name       ===> RAIJOB7
   Job Number     ===> 4269              (Assigned by Job Entry Subsystem   )
   Job type run   ===> Restart           (Initial/Restart run               )
   Step Procstep  ===> SRMIVP            (Step / Procstep name              )
   Step Status    ===> Abended / S222    (Jobstep Status                    )
   Program        ===> RAIPGM7           (Application Program Name          )

S File     File   Records   Records   Total   Percent   Blocks    Total  Block
  Name     Type   Processed Remaining Records Processed Processed Blocks Size

. INFILE   INPUT       6070      1430    7500     80.93       303    375  1600
. OUTFILE  OUTPUT      6070         -       -         -         -      -  6160
. OUTFILE2 OUTPUT        77         -       -         -         -      -  3200
```

Figure 4.2 Repositionable File I/O Summary

```
---------------------- Smart/RESTART SQL Access Summary ----- Row 1 to 5 of 5
Command ===>                                                  Scroll ===> PAGE

Job/Jobstep Information
   Job Name       ===> RAIJOB4
   Job Number     ===> 8765              (Assigned by Job Entry Subsystem   )
   Step/Procstep  ===> SRMIVP
   Job status     ===> Suspended
   Plan Name      ===> SRSSAMPQ          (Db2 Application Plan              )
   Current Module ===> SRSDBRM           (Current package or DBRM           )
   Statement type ===> COMMIT            (Current SQL statement type        )
   LUW number     ===> 210

S Module   Stmt  Stmt                    Total
  Name     #     Type                    Amount

. SRSSAMPQ  148  FETCH.................  492
. SRSSAMPQ  139  CLOSE.................   98
. SRSSAMPQ  130  OPEN..................   99
. SRSDBRM   247  UPDATE................   20
. SRSDBRM   425  COMMIT................   20
```

Figure 4.3 SQL Access Summary

You can also 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.

Smart/MONITOR also lets you browse and edit a restartable application's working storage as well as the contents of its repositionable file buffers. This enables authorized users to ***repair bad data in real-time.*** For example, the Record Detail panel in Figure 4.4 illustrates how you can view and even modify individual records within sequential files.

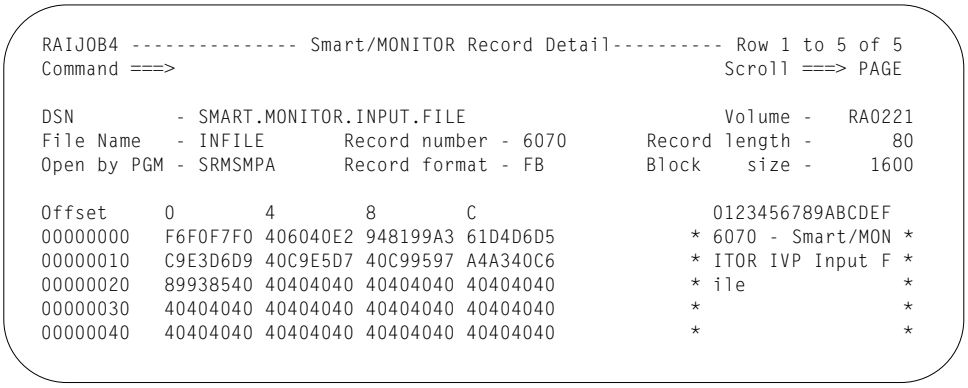


Figure 4.4 Run-time File Record Display and Modification

NOTE: *Relational Architects recommends you suspend execution (as described in the next Section)* ***prior*** *to changing an application's storage or buffer contents.*

4.2 Controlling Restartable Jobs

Smart/RESTART lets you dynamically influence the run-time behavior of restartable applications. For example, you can adjust checkpointing frequency as described in Section 2.4 to increase execution speed (by taking fewer checkpoints) or improve concurrency (by checkpointing more frequently). You can also suspend and resume execution of your restartable applications at any time. The Quiesce command lets you *gracefully* cancel an application at a unit of work boundary so there is no wasted processing or time consuming backout.

You can perform these actions and many others via Smart/MONITOR or by issuing MVS MODIFY commands from the console.

4.3 Smart / PREDICTOR

The Smart/PREDICTOR component of Smart/RESTART estimates job step completion times and issues early warnings when a job is in danger of running too long. These warnings let operators take remedial action to help meet production schedules.

Smart/PREDICTOR calculates a *target* completion time by adding the *estimated* run time (supplied as an RAINPUT parameter) to the *actual* job submission time. When Smart/PREDICTOR recognizes that a job will run past its target completion time, it issues WTO's that appear on the console and in the JES listing.

The Run Time Estimator, shown in Figure 4.5, displays forecasting statistics and a graph of the progress of the job. In this example, the *target* completion time is 09:00 while the *projected* completion time is 08:42 – ahead of schedule!

The top line of the progress graph shows the completion percentage for the *current* run while the second line shows the *total* percentage of completion since the application was first submitted. Since Figure 4.5 illustrates an initial run, both progress lines show the same completion percentage.

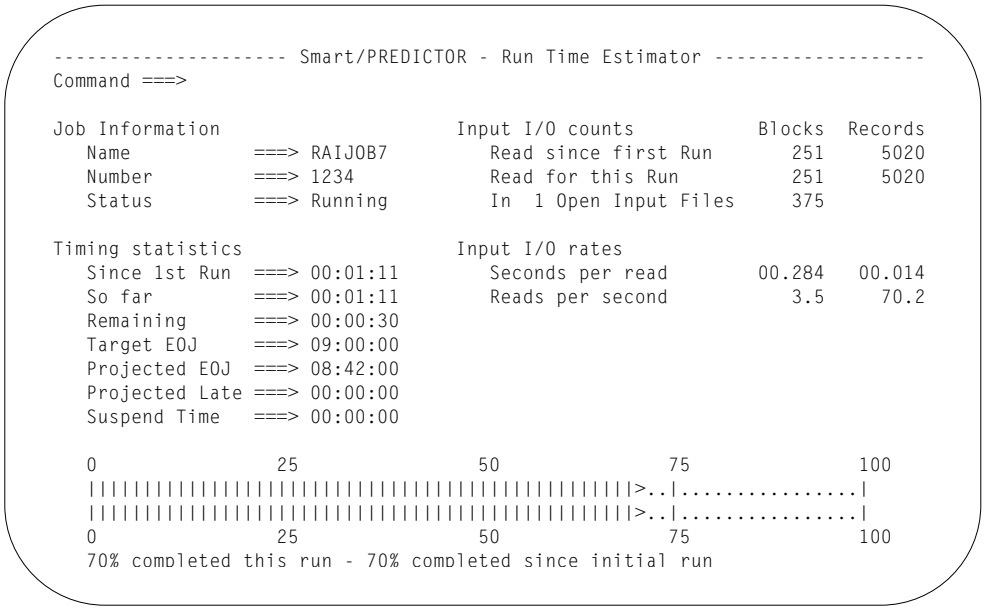


Figure 4.5 Smart/PREDICTOR Run Time Estimator

4.4 Smart / MONITOR Operation

Smart/MONITOR supports ***basic*** and ***extended*** modes of operation. Basic mode gets information from the checkpoint medium so you can examine I/O activity and working storage as of the most recent checkpoint. Extended mode goes cross memory to support *real-time* modification of data as well as real-time displays of such statistics as SQL activity and repositionable file I/O.

Smart/MONITOR lets you dynamically switch among Db2 subsystems so you can monitor restartable applications running on *multiple* Db2 subsystems during a single session.

> ***NOTE:*** *Security administrators can control access to authorized Smart/MONITOR functions on a granular basis via standard OS/390 SAF interfaces.*

5

Smart / RRSAF

Smart/RRSAF provides an alternative to running batch Db2 programs under the TSO Terminal Monitor Program and the DSN RUN command. Smart/RRSAF lets you run your Db2 batch applications using regular JCL, so they ***look*** and ***behave*** like regular z/OS job steps.

Unlike TSO Attach, Smart/RRSAF fully supports RRSAF (the Db2 Recoverable Resource Manager Services Attachment Facility) and does so without the need for user code to manage an *explicit* Db2 connection. RRSAF is IBM's strategic Db2 attachment that provides 2-phase commit support for multi-site updates and distributed transactions that alter multiple recoverable resources such as Db2 tables, MQ messages and IMS databases within a single unit of recovery.

Smart/RRSAF, like Smart/RESTART, is a member of the ***Smart Jobstream Series*** from Relational Architects. Smart/RRSAF can be licensed and used independently of Smart/RESTART, but there are benefits to using the two products together (as described in Section 5.6). Unlike Smart/RESTART, Smart/RRSAF works *only* with Db2 applications.

Additional information on Smart/RRSAF appears in the Smart/RRSAF User Guide and Reference (Publication SJS-002).

5.1 Smart/RRSAF History and Benefits

Smart/RRSAF was originally developed (independently of Smart/RESTART) because abends in Db2 batch programs are masked by the TSO Terminal Monitor Program (TMP). Since the job step ***appears*** to end normally, neither multi-step jobs nor dependent job scheduling packages work correctly. Nor does the abnormal disposition of datasets.

Smart/RRSAF solves these problems by setting job step completion codes which accurately reflect the outcome of Db2 batch application processing. This allows Db2 batch programs to work properly in multi-step jobstreams and to interoperate with production control systems like CA-11 and IBM's OPC. The abnormal disposition of datasets also works the way you expect and intend.

IBM eventually addressed the shortcomings of Db2 batch run under TSO. However, it remains much easier to diagnose and correct program failures in the Smart/RRSAF environment than with the TSO TMP. An abend in your application program causes abnormal job step termination – directly. With Smart/RRSAF, there are no other active tasks or programs to confuse or obscure what happened.

In contrast, TSO's alternate entry points IKJEFT1A and IKJEFT1B always indicate an 04C abend at the job step level, *regardless* of the type of program failure. Although the system or user abend is returned as a reason code, this information is often lost or overlooked. Meanwhile, the reason code associated with the *original* abend is gone. With Smart/RRSAF, however, this information appears in the JES listing – just as issued.

Smart/RRSAF provides additional benefits over running Db2 batch applications under the TSO Terminal Monitor Program, which include the following:

- With Smart/RRSAF a single unit of recovery can encompass changes to Db2, MQ and IMS resources. Although IBM has identified RRSAF as its strategic interface for Db2 batch applications running in z/OS, TSO Attach (the TSO TMP, DSN command and RUN subcommand) does not support RRSAF.
- Smart/RRSAF's transparent support for both RRSAF and CAF permits applications to utilize either Db2 attachment without source changes. Smart/RRSAF lets you toggle between RRSAF and CAF – on a jobstep basis or globally as a default.
- Smart/RRSAF affords all the benefits of RRSAF without the need for application code and recovery routines to manage an *explicit* Db2 connection.
- Smart/RRSAF lets you use a single Db2 attachment for both batch applications and Db2 stored procedures.
- Smart/RRSAF recognizes whether a Db2 stored procedure is executing in a Db2 or WLM established stored procedure address space and automatically uses RRSAF or CAF as appropriate.
- Smart/RRSAF typically presents more information in a more meaningful format than does TSO/DSN.
- Smart/RRSAF records SMF job step accounting under the application program name. In contrast, ***all*** Db2 batch applications run under TSO have their charge back statistics lumped together under the TSO Terminal Monitor Program (IKJEFTxx).
- You can migrate from TSO Attach to Smart/RRSAF without source changes as described in Section 5.4 below.
- Smart/RRSAF provides exits (as described in Section 5.3) with which sites can customize processing.
- The profile definition dialogs supplied with Smart/RRSAF let users customize the product for operation with specific Db2 subsystems or logical environments *within* a subsystem. This often eliminates the need for RAINPUT control statements altogether.

- JCL for Smart/RRSAF is easier to setup and revise than for TSO/DSN because there are fewer control statements to review and change. Moreover, IT professionals who deal with JCL (developers, quality assurance, technical support, and operations professionals) find Smart/RRSAF jobstreams more familiar and intuitive than analogous jobstreams set up for TSO/DSN.
- The standard MVS search order is *always* used with Smart/RRSAF. This avoids the confusion that may occur with the DSN LIBRARY parameter.
- Smart/RRSAF provides flexible diagnostics for SQL error and warning conditions.
- Smart/RRSAF provides flexibility in dealing with Db2 connection failures. For example, Smart/RRSAF can retry the connection request when Db2 is inactive. In addition, you can direct Smart/RRSAF to ABEND when a Db2 connection failure occurs or to return control to your application with a negative SQL code.
- Short running batch Db2 applications exhibit significant performance improvements because Smart/RRSAF job steps initialize *faster* than TSO address spaces. However, these improvements become less significant for longer running applications.
- OS/VS COBOL modules can call their subroutines dynamically, without the need for the VS COBOL II run-time library. The version of module DSNHADDR supplied with Smart/RRSAF makes this possible.
- Smart/RRSAF (when used independently of Smart/RESTART) supports the use of MVS Checkpoint/Restart whereas TSO/DSN does not. This is due to the single-tasking structure of Smart/RRSAF.
- High volume applications that can exploit Smart/RRSAF's *parallel thread* support can achieve dramatically reduced elapsed times.

Smart/RRSAF also includes a Smart/Dialog Attachment Facility for use with Db2/ISPF based applications. ISPF applications implemented with the Smart/Dialog Attach Facility exhibit improved flexibility, performance and ease of maintenance as compared to Db2/ISPF applications implemented with the DSN command. Dialog developers can also use the ISPF SELECT service freely, without the restrictions imposed by the DSN command.

5.2 Smart/RRSAF JCL

Figure 5.1 shows sample JCL to run a Db2 batch application with Smart/RRSAF using regular JCL. Program parameters are passed in the normal way on a standard JCL EXEC statement.

```
//STEP1    EXEC  PGM=YOURPGM,PARM='sample parameters'
//STEPLIB    DD  DSN=user.runlib.load,DISP=SHR
//           DD  DSN=smart.rrsaf.DCALOAD,DISP=SHR
//RAINPUT    DD  DSN=input.parm.file,DISP=SHR
```

Figure 5.1 JCL to run Db2 batch programs with Smart/RRSAF

With Smart/RRSAF, the default Db2 plan name is the same as the program name whereas the default Db2 subsystem is defined at installation time. More than one copy of Smart/RRSAF can be installed, each with its own set of defaults. Alternatively, the Db2 subsystem and application plan names can be specified explicitly through an optional RAINPUT file. The Db2 subsystem and application plan names can also be changed at run-time by Smart/RRSAF exits (see Section 5.3).

In contrast, Figure 5.2 shows JCL to run the *same* Db2 batch application under the TSO Terminal Monitor Program by executing the DSN command and RUN sub-command.

```
//STEP1    EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSIN  DD  *
  DSN SYSTEM(Db2T)
      RUN  PROGRAM(YOURPGM) PLAN(YOURPLAN) -
           PARMS('sample parameters') -
           LIBRARY('user.runlib.load')
//SYSTSPRT DD  SYSOUT=*
```

Figure 5.2 JCL to run Db2 batch programs under TSO

5.3 Smart/RRSAF Exits

Smart/RRSAF provides optional *connection* and *plan name* exits that allow you to customize your environment. Smart/RRSAF calls any site written connection exit before it connects to Db2 whereas the plan name exit is invoked before Smart/RRSAF creates a thread for the Db2 application plan.

For example, a site written plan name exit can allow multiple environments (such as development and system test) to *co-exist* within the same Db2 subsystem by inspecting and modifying the name of the application plan used to create a Db2 thread.

Smart/RRSAF exits also let you standardize on a single plan name when products vary their plan name with each version and release. Moreover, Smart/RRSAF exits can be used to enforce site security standards *before* a connection to Db2 is established.

When multiple Db2 versions run in a single z/OS image, Smart/RRSAF exits can dynamically allocate the appropriate Db2 load libraries at run-time. This avoids the need to STEPLIB to version-specific load libraries.

Appendix D of the Smart/RRSAF User Guide describes how to write connection and plan name exit routines. The Smart/RRSAF product includes sample exits in source form.

5.4 Implementing Smart/RRSAF

Program preparation for the Smart/RRSAF environment entails the same preprocess, compile, link edit and bind steps required for Db2 programs that run in TSO, CICS and IMS environments.

No source code changes to your Db2 batch programs are required to implement Smart/RRSAF. Existing Db2 batch applications which previously ran under TSO/DSN can be migrated to the Smart/RRSAF environment without program preparation, although their execution JCL must be modified.

Applications that call subroutines dynamically (such as COBOL modules compiled with the DYNAM option) will work 'as is' with Smart/RRSAF. In contrast, programs with statically linked subroutines must be re-linked to enable Smart/RRSAF operation.

Smart/RRSAF includes an ISPF dialog to convert Db2 load modules from TSO/DSN operation to Smart/RRSAF. The Smart/RRSAF Conversion Facility tailors link-edit JCL (for a single application or an entire load module library) that replaces TSO Attach with Smart/RRSAF.

The messages issued by Smart/RRSAF can be directed to a file, or as illustrated in Figure 5.3, to the JES listing.

```
DCA010I - Smart / RRSAF has connected to Db2 subsystem Db2T
DCA011I - Thread created for application plan YOURPLAN
DCA014I - Thread for application plan YOURPLAN has terminated NORMALLY
DCA015I - Smart / RRSAF has disconnected from Db2 subsystem Db2T
DCA016I - Smart / RRSAF processing completed
IEF142I YOURJOB2 STEP1 - STEP WAS EXECUTED - COND CODE 0000
```

Figure 5.3 Messages from Smart/RRSAF

5.5 SQL Monitoring

To produce Db2 applications with superior performance, developers need timely and comprehensive performance statistics. SQL statement statistics are of particular interest in tuning Db2 applications. Smart/RRSAF optionally produces such statistics through a monitoring facility that can be turned on and off dynamically while a job step is running.

SQL performance statistics can be directed to a sequential file for batch reporting as illustrated in Figure 5.4. Alternatively, Smart/RRSAF can maintain SQL statistics in memory where Smart/MONITOR (a component of Smart/RESTART) can access them.

```
Job Name YOURJOB2                    SQL/Monitor          Start: 03/01/2009-20:26:56 GMT
Job Num  4133                                             End  : 03/01/2009-20:30:06 GMT
               SQL Counts grouped by Module: SRSDBRM , SQL statement ID

< SQL statement >  Request <      Total time, sec      > <     Average time, msec    >
Type            ID   Number       TCB       SRB   Elapsed       TCB       SRB   Elapsed
-----------------  -------  --------  --------  --------  --------  --------  --------
Insert       00170      802     2.868     0.009    55.249     3.576     0.010    68.889
Commit       00462        5     0.008     0.000     0.282     1.675     0.048    56.429
Select       00422     1212    11.414     0.001    12.838     9.417     0.001    10.592
Update       00267      408   104.546     0.034   115.997   256.239     0.083   284.307
Update       00286        4     0.886     0.000     1.012   221.524     0.046   253.050
Select       00321        2     0.024     0.000     0.025    11.955     0.000    12.306
=================  =======  ========  ========  ========  ========  ========  ========
SRSDBRM  total         2436   119.767     0.045   185.580    49.165     0.018    76.182
```

Figure 5.4 Report of execution counts by module and SQL statement ID

5.6 Smart/RRSAF and Smart/RESTART

Smart/RRSAF significantly enhances Smart/RESTART. Together, they provide valuable monitoring, diagnostic and exception handling facilities that are not available with TSO/DSN and other Db2 attachment packages.

For example, Smart/RRSAF intercepts every SQL request your application issues. This enables Smart/RESTART to *automatically* recover from Db2 timeouts, deadlocks and resource unavailable conditions and *retry* the unit-of-work. Used together, Smart/RRSAF and Smart/RESTART can reduce or eliminate Db2 contention as a cause of batch application failure.