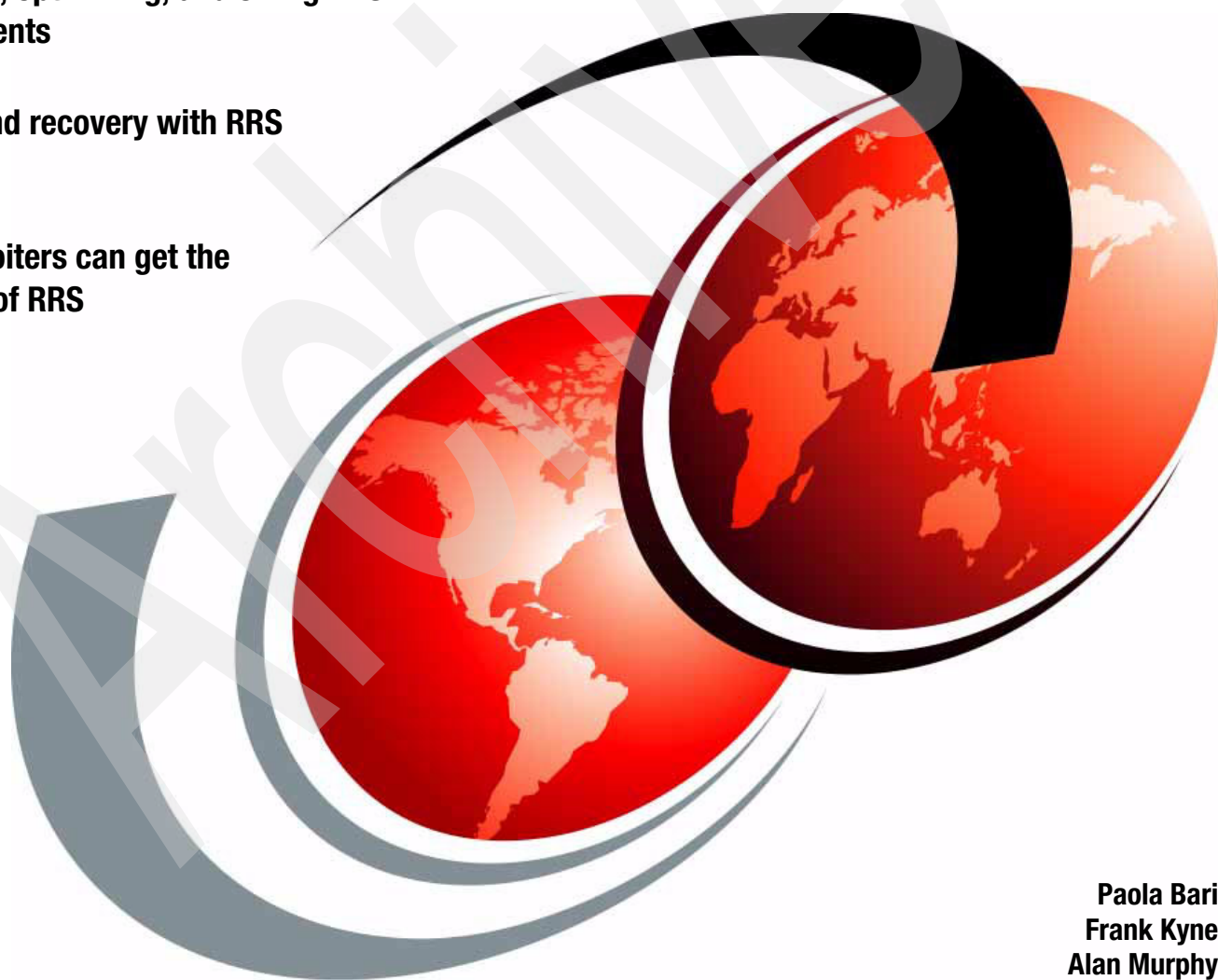


Systems Programmer's Guide to Resource Recovery Services (RRS)

Managing, optimizing, and sizing RRS
environments

Restart and recovery with RRS

How exploiters can get the
most out of RRS



Paola Bari
Frank Kyne
Alan Murphy

Redbooks



International Technical Support Organization

**Systems Programmer's Guide to Resource Recovery
Services (RRS)**

November 2004

Archived

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (November 2004)

This edition applies to Version 1, Release 4 of z/OS (product number 5694-A01, 5655-G52).

© Copyright International Business Machines Corporation 2004. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Trademarks	viii
Preface	ix
The team that wrote this redbook.	ix
Become a published author	x
Comments welcome.	x
Part 1. Resource Recovery Services (RRS) introduction and concepts	1
Chapter 1. Introduction to Resource Recovery Services (RRS)	3
1.1 Transactions	4
1.2 Resource managers and protected resources	5
1.3 The role of Resource Recovery Services (RRS)	6
1.3.1 Who uses RRS	7
Chapter 2. Two-phase commit and RRS	9
2.1 Introduction to two-phase commit	10
2.2 Two-phase commit as supported by legacy resource managers	13
2.2.1 CICS	14
2.2.2 IMS	15
2.2.3 DB2	15
2.3 How RRS works	16
2.3.1 Registration services	16
2.3.2 Context services	17
2.3.3 RRS invocation	18
2.4 How two-phase commit works with RRS	19
2.5 Summary	21
Chapter 3. Distributed RRS	23
3.1 Distributed two-phase commit	24
3.1.1 RRS distributed syncpoint support	25
3.1.2 Multisystem cascaded transactions	26
Part 2. Implementing and managing RRS	29
Chapter 4. Implementing RRS	31
4.1 RRS Implementation overview and planning	32
4.2 Define the logging environment	32
4.2.1 RRS logging group name	33
4.2.2 Log stream characteristics	34
4.2.3 RRS log stream structure sizing	34
4.2.4 Define the RRS log streams	35
4.3 Define the RRS infrastructure	42
4.3.1 WLM definitions	42
4.3.2 RRS subsystem definitions	42
4.3.3 Define RRS procedure	43
4.3.4 RRS automation	43
4.3.5 Define RRS panels to ISPF	43
4.3.6 Define RRS SAF authorization	44

4.3.7 Define RRS component trace	44
Chapter 5. RRS operations.	45
5.1 Starting RRS	46
5.1.1 RRS warm start.	47
5.1.2 RRS cold start.	47
5.2 Stopping RRS	48
5.3 Using RRS panels	48
Chapter 6. RRS performance and availability.	57
6.1 Availability considerations for RRS log streams	58
6.2 Performance considerations of RRS log streams	59
6.2.1 RRS performance monitoring	60
Chapter 7. RRS restart and recovery	61
7.1 RRS restart	62
7.1.1 RRS log takeover	62
7.2 Resource manager restart	63
7.2.1 Resource manager startup sequence	63
7.2.2 Resource Manager restart restrictions	63
7.2.3 Example of resource manager restart within the same RRS logging group	66
7.2.4 Example of resource manager restart outside the same RRS logging group	66
7.2.5 Sample DB2/MQ restart scenario with RRS	66
Part 3. RRS exploiters	71
Chapter 8. WebSphere Application Server for z/OS	73
8.1 Introduction	74
8.2 J2EE terminology	74
8.3 RRS exploitation	75
8.4 Connectors for JDBC, JMS and JCA	76
8.4.1 IMS connectors	76
8.4.2 CICS connectors	78
8.4.3 DB2 connector	79
8.4.4 WebSphere MQ connector	81
8.4.5 Connector summary table	81
8.4.6 RRS versus XA resource adapters	82
8.5 Restart and recovery issues with RRS	82
8.5.1 RRS failure	82
8.5.2 Failure and restart	83
8.5.3 Peer restart and recovery	84
8.6 Example scenarios	87
8.6.1 Application updating CICS and IMS using RRS connectors	88
8.6.2 Application updating CICS and IMS with RRS and XA connectors	89
8.6.3 Application backout updating CICS and IMS with RRS and XA connectors	91
Chapter 9. DB2 for z/OS	93
9.1 DB2 RRS requirements	94
9.1.1 DB2 RRS Attach facility	94
9.1.2 DB2 Stored Procedures	95
9.1.3 DB2 JDBC/SQLJ driver for OS/390	96
9.1.4 DB2 Universal JDBC/SQLJ driver	96
9.2 DB2 restart and recovery with RRS	97
9.2.1 DB2 restart if RRS is unavailable	97

9.2.2 DB2 restart on another system	98
9.3 Sample scenarios for DB2 using RRS	98
9.3.1 Normal commit processing scenario	98
Chapter 10. CICS Transaction Server	103
10.1 CICS RRS requirements	104
10.1.1 Working in CICS	104
10.1.2 Connecting to CICS via EXCI	106
10.2 CICS restart and recovery with RRS	109
10.2.1 RRS failure	109
10.2.2 CICS restart	110
10.2.3 Operator commands	110
10.2.4 CICS example	111
Chapter 11. IMS	115
11.1 How IMS/ESA exploits RRS	116
11.2 Connecting to IMS/ESA	116
11.2.1 ODBA	116
11.2.2 APPC/IMS	117
11.2.3 OTMA	121
11.3 IMS/ESA restart and recovery with RRS	123
11.3.1 RRS failure while IMS is active	124
11.3.2 IMS restart when RRS is not available	124
11.3.3 IMS restart when RRS has been cold-started	125
11.3.4 IMS restart on a different system	125
11.4 IMS/ESA sample scenario using RRS	126
Chapter 12. WebSphere MQ for z/OS	129
12.1 WebSphere MQ RRS requirements	130
12.1.1 WebSphere MQ and DB2 stored procedures	130
12.1.2 WebSphere MQ JMS interface	130
12.2 WebSphere MQ restart and recovery issue with RRS	131
12.2.1 RRS failure when MQ is running	131
12.2.2 WebSphere MQ restart if RRS is unavailable	132
12.2.3 WebSphere MQ restart on another system	132
12.3 Sample scenarios for WebSphere MQ using RRS	133
12.3.1 Normal commit processing scenario	133
Chapter 13. APPC/MVS	137
13.1 APPC/MVS RRS requirements	138
13.1.1 Transaction flow using APPC/MVS protected conversations	139
13.1.2 APPC/MVS system requirements for protected conversations	139
13.1.3 Managing APPC/MVS resources for protected conversations	139
13.2 APPC/MVS application restart and recovery with RRS	139
13.2.1 RRS failure while the APPC/MVS application is active	140
13.3 APPC/MVS sample scenario with RRS	140
Chapter 14. DFSMStvs	143
14.1 DFSMStvs features that exploit RRS	144
14.1.1 Resource recovery participants	145
14.1.2 Commit flow with DFSMStvs	146
14.1.3 Backout flow with DFSMStvs	147
14.1.4 Handling of undo records when in-doubt with DFSMStvs	148
14.1.5 Handling long-running units of recovery with DFSMStvs	149

14.2 TVS restart and recovery with RRS	149
14.2.1 RRS failure	150
14.2.2 DFSMStvs restart	153
14.2.3 DFSMStvs peer restart	153
14.2.4 Operator commands	154
14.3 DFSMStvs examples	157
Related publications	163
IBM Redbooks	163
Other publications	163
How to get IBM Redbooks	163
Index	165

Notices

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

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

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

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

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

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

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

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

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


This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

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

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Redbooks (logo) ™

z/OS®

zSeries®

CICS®

Distributed Relational Database
Architecture™

DB2®

DRDA®

IBM®

IMS™

IMS/ESA®

Language Environment®

MQSeries®

MVS™

OS/390®

Parallel Sysplex®

Redbooks™

RACF®

RMF™

S/390®

VTAM®

WebSphere®

The following terms are trademarks of other companies:

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates.

EJB, Java, JDBC, J2EE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

Preface

This IBM® Redbook gives you a broad understanding of the Resource Recovery Services (RRS) environment. RRS provides a global syncpoint manager that any resource manager on z/OS® can exploit. It enables transactions to update protected resources managed by many resource managers.

RRS is increasingly becoming a prerequisite for new resource managers, and for new capabilities in existing resource managers. Rather than having to implement their own two-phase commit protocol, these products can use the support provided by RRS.

Since older transaction managers like CICS® already offered many of the benefits of RRS for processing their own data, not many people rushed to exploit RRS when it was first introduced. However, as more transaction managers have become RRS resource managers, and as the complexity of the exchanges of transactional data increases, more and more systems and application programmers will need to use RRS.

This redbook provides information that will help you install, tailor, and manage the RRS environment. It covers RRS exploiters, helping you to understand the connections between RRS and its exploiters and how they work together, as well as how the installation should behave in recovery and restart situations.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Paola Bari is an Advisory Programmer at the International Technical Support Organization, Poughkeepsie Center. She has 20 years of experience as a Systems Programmer in OS/390®, z/OS and Parallel Sysplex®, and several years of experience in WebSphere® MQSeries® and WebSphere Application Server.

Frank Kyne is a Senior IT Specialist at the International Support Organization, Poughkeepsie Center. He writes extensively and teaches IBM classes worldwide in all areas of Parallel Sysplex. Before joining the ITSO, Frank worked in IBM Global Services in Ireland as an MVS™ Systems Programmer.

Alan Murphy is a Senior IT Specialist working for IBM Global Services in Ireland. He has 18 years of systems programming experience on the zSeries® and S/390® platforms. He is co-author of a number of previous IBM Redbooks™ on Parallel Sysplex.

Thanks to the following people for their contributions to this project:

Stephen Anania
IBM Poughkeepsie

Juliet Candee
IBM Poughkeepsie

Mitch Johnson
IBM zSeries Services

Rick Kready
IBM Poughkeepsie

Matthew Sykes
IBM Poughkeepsie

Tamas Vilaghy
IBM International Technical Support Organization, Poughkeepsie Center

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

ibm.com/redbooks

- Send your comments in an Internet note to:

redbook@us.ibm.com

- Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Resource Recovery Services (RRS) introduction and concepts

In this part we introduce Resource Recovery Services (RRS) and discuss concepts such as transactions, resource managers, and two-phase commit, that you need to understand before implementing RRS.

Archived



Introduction to Resource Recovery Services (RRS)

In this chapter, we introduce concepts required to understand Resource Recovery Services (RRS). We also describe what RRS is, and why it is needed in a transactional environment.

This chapter covers the following topics:

- ▶ “Resource managers and protected resources” on page 5
- ▶ “The role of Resource Recovery Services (RRS)” on page 6

1.1 Transactions

How times have changed. Not that long ago tie-dyed jeans were in fashion—and a “complicated” application was a batch job that used both IMS™ and VSAM within the same job! Today, a reasonably complex application is one that verifies a user’s identity using digital certificates; links to a WebSphere application that integrates CICS transactions, IMS transactions, and DB2® stored procedures spread across four separate enterprises; and sends multimedia files and a digital receipt back to the user at his Internet screen. And all of this within a single unit of recovery!

Sound outlandish? Well consider that the user is a customer sitting at home using the Internet to book a vacation. Having selected the flights, hotel, and car, the user decides to book it. This will initiate a transaction to confirm the user’s identity, update the airline’s system with the reservation, update the hotel’s system to book the room, update the car rental company’s system to book the car, debit the user’s bank account with the cost of the package, and finally send the user a multimedia file containing information about the package, along with a receipt for the transaction. Figure 1-1 illustrates this transaction.

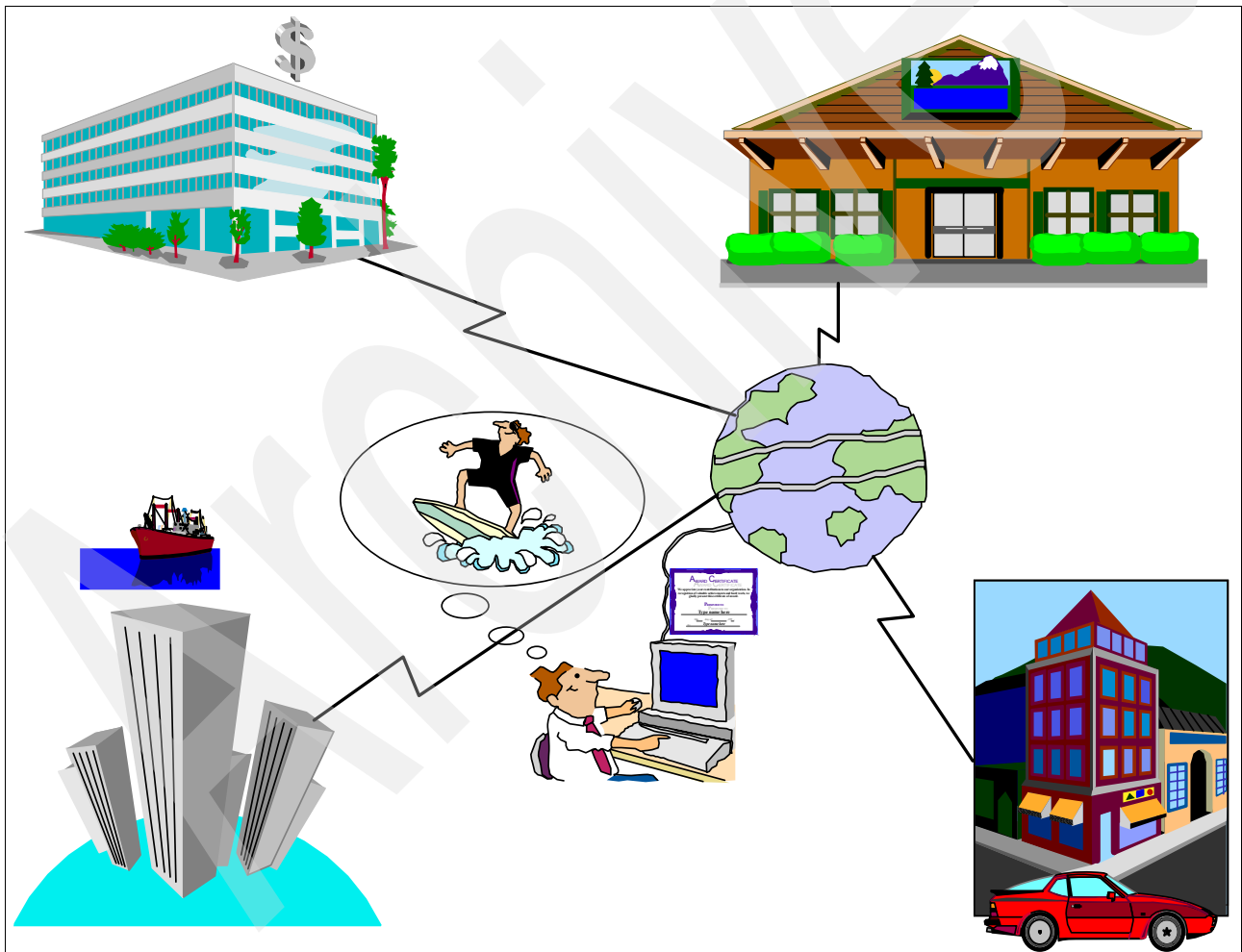


Figure 1-1 Overview of the sample vacation booking transaction

The most important point of this transaction is that all processing must be handled as one atomic transaction. The customer will not be satisfied if the hotel and car are booked and the money withdrawn—but the flights are not booked! So either *every* part of the transaction must

occur, or *none* of it should occur. Meeting the challenge of this new paradigm in a S/390 environment is the focus of this IBM Redbook.

The constituent parts of a modern transaction are changing and growing in number. The traditional transaction managers CICS and IMS are still important and will continue to play a critical role in the new environment. However, there will also be many new components, such as WebSphere MQ, Websphere Application Server and many more.

In this redbook, we describe what options are available in building transactions that more accurately reflect your actual business processes and the interactions between your company and those it does business with. First, however, we need to define the term *transaction* as used here.

Definition of a transaction:

We define a *transaction* as consisting of all activity from the start of a task until the initiating program requests that all changes are either committed or backed out. This differs from the way many people use the term “transaction”, where there may be a number of independent commits or backouts made within the one task. In this publication we will use the term “work request” when referring to an interaction that may consist of more than one unit of recovery.

To be a transaction, a work request must have the following properties:

Atomic	A transaction is known as <i>atomic</i> when an application changes data in multiple resource managers as a single transaction, and all of those changes are accomplished through a single commit request by a syncpoint manager. If the transaction is successful, all the changes are committed. If any piece of the transaction is not successful, then all changes are backed out. An <i>atomic instant</i> occurs when the syncpoint manager in a two-phase commit process logs a commit record for the transaction.
Consistent	Applications involved in the transaction must be written to a standard that ensures consistency. If the application is correctly designed, then the transaction should maintain a consistent view of data.
Isolated	The database managers involved in the transaction isolate the updates to their data so that only the application changing the data knows about the individual updates until the transaction is complete.
Durable	Database managers involved in the transaction ensure the data is persistent, both before and after the transaction, regardless of success or failure.

Based on the first letter of each property, this type of transaction is often referred to as an *ACID transaction*.

1.2 Resource managers and protected resources

Applications need to access resources in a manner that guarantees integrity. An application must be sure that changes to data have been either made or backed out. To enable this to happen, applications usually access resources via resource managers.

Definition of a resource manager (RM):

A *resource manager* (RM) is a subsystem or component such as CICS, IMS, or DB2 which manages resources that can be involved in transactions. Resource managers can be categorized as *work managers*, *data resource managers*, and *communication resource managers*.

Each category of resource manager is described later in this publication.

A data resource manager such as DB2 handles tasks like data access, locking and serialization, commit or rollback processing, and restart and recovery. DB2 data is referred to as a *protected* or *recoverable* resource. A protected resource is a resource that can be updated in a manner that conforms to the requirements of an ACID transaction. A protected resource need not be just a piece of data in a DB2 or IMS database, it can be any resource that can be changed during the process of a transaction.

Resource managers must provide the following functions to ensure that the resources they manage can be considered protected resources:

- ▶ *Locking* functions, to ensure that changes to protected resources not yet committed by a transaction can only be seen by that transaction
- ▶ *Logging* functions, to ensure that before and after images of changed resources are held in order to allow the changes to be backed out, or in order for restart processing to reapply changes in the event of system failure

As previously noted, there must be a syncpoint manager that logs the application change request and makes a decision as to whether the resource managers should commit or back out any changes to protected resources. This process is referred to as the *two-phase commit* process and is discussed in detail in “Introduction to two-phase commit” on page 10.

In the case of an application simply updating DB2 data, DB2 can act as both the resource manager and the syncpoint manager. This is the simplest scenario. When more than one resource manager is involved, things become more complex. There must be one syncpoint manager that talks to all involved resource managers and coordinates the commit process.

Over the years resource managers on z/OS (such as IMS, CICS and DB2) have developed to provide two-phase commit processing between them. This will involve one of the resource managers taking the role of the syncpoint manager to coordinate the two-phase commit processing. This has required code specifically written by the IMS, CICS and DB2 developers to implement the process for every possible scenario.

1.3 The role of Resource Recovery Services (RRS)

With the increasing number of resource managers now available on z/OS, there was clearly a need for a general syncpoint manager on z/OS that any resource manager could exploit. This is the role of Resource Recovery Services (RRS), a component of z/OS. RRS provides a global syncpoint manager that any resource manager on z/OS can exploit. It enables transactions to update protected resources managed by many resource managers.

In this redbook we discuss the various resource managers in high level terms, and describe how each of them takes part in a two-phase commit environment. Some of the resource managers already support two-phase commit in certain environments, so we briefly cover those capabilities. However, we concentrate on the new capabilities provided by RRS. We also provide detailed information about the theory of two-phase commit and how RRS

implements this protocol. Finally, we provide information to help you manage RRS and its interactions with the resource managers in your environment.

1.3.1 Who uses RRS

RRS is increasingly becoming a prerequisite for new resource managers, or for new capabilities in existing resource managers. Rather than having to implement their own two-phase commit protocol, these products can use the support provided by RRS. Examples of recent exploiters of RRS are:

- ▶ WLM-managed stored procedures in DB2 requires RRS. Similarly, if you want to have a single unit of recovery between a stored procedure and any resource manager *other* than DB2, you must use RRS.
- ▶ WebSphere Application Server for z/OS: in this case, RRS is *always* required; however, its use is hidden from the application developer.
- ▶ If you plan to use the Transactional EXCI interface provided by CICS Transaction Server 1.3 or later, RRS must be started in the system.

Note: Please note that the EXCI interface has two modes of operation governed by the SYNCONRETURN keyword.

▶ Transactional EXCI

The DPL request is specified without the SYNCONRETURN keyword. In this mode of operation CICS expects there to be an RRS context present when the DPL call is made. If this context is missing then the DPL call will fail indicating that RRS was unavailable.

By omitting SYNCONRETURN, the calling application is notifying CICS that any work performed during the DPL call is to be bound to the RRS context - this allows the work performed during the DPL call to be part of a wider distributed unit of work. If the RRS context is directed to commit then CICS will commit the DPL work, if it is directed to rollback then CICS will rollback the DPL work. RRS provides COMMIT and ROLLBACK verbs that allow an application to direct the completion of the RRS context.

The purpose of this mode is to allow the coordination of CICS resources with resources outside of its control.

▶ EXCI

The DPL request is specified with the SYNCONRETURN keyword. In this mode of operation CICS does not expect an RRS context, instead it creates a transaction under which the DPL call runs until the call completes. If the DPL call completes successfully (no abend) then CICS commits the DPL work (hence synconreturn). If the DPL call abends then the transaction is rolled back.

Since CICS is not expecting an RRS context when it operates in this manner, it is possible to make DPL requests even though RRS isn't started.

- ▶ APPC/MVS requires RRS when using protected conversations.
- ▶ OTMA and ODBA support in IMS/ESA® requires RRS to be enabled when Synclevel Syncpt is specified for IMS Connect.
- ▶ Transactional extensions to VSAM record level sharing (RLS) introduced in z/OS V1R4 exploit RRS to allow batch jobs include VSAM files in the scope of an ACID transaction.

Since older transaction managers like CICS already offered many of the benefits of RRS for processing their own data, not many people rushed to exploit RRS when it was first introduced. However, as more transaction managers have become RRS resource managers, and the complexity of the exchanges of transactional data increases, more and more application programmers are going to need to use RRS. This redbook provides information that will help systems programmers use RRS.

Two-phase commit and RRS

In this chapter, we introduce the concept of two-phase commit and discuss the benefits it provides. We then take a closer look at Resource Recovery Services (RRS), a z/OS component that provides two-phase commit support across multiple resource managers.

This chapter covers the following topics:

- ▶ “Two-phase commit as supported by legacy resource managers” on page 13
- ▶ “How RRS works” on page 16
- ▶ “How two-phase commit works with RRS” on page 19

2.1 Introduction to two-phase commit

The *two-phase commit* protocol is a set of actions used to make sure that an application program either makes *all* changes to the resources represented by a single unit of recovery (UR), or makes *no* changes at all. This protocol verifies that either all changes or no changes are applied even if one of the elements (such as the application, the system, or the resource manager) fails. The protocol allows for restart and recovery processing to take place after system or subsystem failure.

The two-phase commit protocol is initiated when the application is ready to commit or back out its changes. The application program that initiates the commit or backout does not need to know who the syncpoint manager is or how two-phase commit works. This is hidden from the application; a program simply calls a commit or backout service and receives a return code indicating whether this has been successfully completed.

When the application issues the commit request, the *coordinating recovery manager*, also called the *syncpoint manager*, gives each resource manager participating in the unit of recovery an opportunity to vote on whether its part of the UR is in a consistent state and can be committed. If all participants vote YES, the syncpoint manager instructs all the resource managers to commit the changes. If any vote NO, the syncpoint manager instructs them to back out the changes. This process is usually represented as two phases, as explained here.

In phase 1, the application program issues the syncpoint or backout request to the syncpoint manager. The coordinator issues a PREPARE command to send the initial syncpoint flow to all UR agent resource managers. In response to the PREPARE command, each resource manager involved in the transaction replies to the syncpoint manager, stating whether it is ready to commit or not.

After the syncpoint manager receives all the responses back from all of its agents, phase 2 is initiated. In this phase, the syncpoint manager issues the commit or rollback command based on the previous responses. If any of the agents came back with a negative response, the syncpoint initiator tells *all* syncpoint agents to roll back their changes.

If a resource manager (RM) fails before the syncpoint manager issues the PREPARE command, the syncpoint manager assumes a backout is needed and the transaction will be backed out by all RMs. If an RM fails after responding to the prepare but before receiving the commit decision (during the in-doubt stage), then all other RMs commit and the failed RM must contact RRS on restart to determine the correct outcome for the UR.

The moment when the coordinator records that it is going to tell all the resource managers to either commit or roll back is known as the *atomic instant*. Regardless of any failures after that time, the coordinator assumes that all changes are either committed or rolled back. A syncpoint manager usually logs the decision at this point. If any of the participantsabend after the atomic instant, the abending resource manager must work with the syncpoint manager when it restarts to complete any commits or rollbacks that were in process at the time of theabend.

During the first phase of the protocol, the agents do not know whether the syncpoint manager will commit or roll back the changes until the syncpoint manager has collected all responses. This time is known as the *in-doubt period*.

The UR is described as having a particular state depending on what stage it is at in the two-phase commit process, as follows:

In-reset	Before a UR makes any changes to a resource
In-flight	While it is requesting changes to resources

In-prepare	Once a commit request has been made (phase 1)
In-commit	Once the syncpoint manager has made a decision to commit (phase 2 of the two-phase commit process)
In-backout	If the syncpoint manager decides to back out

Figure 2-1 illustrates the two-phase commit protocol.

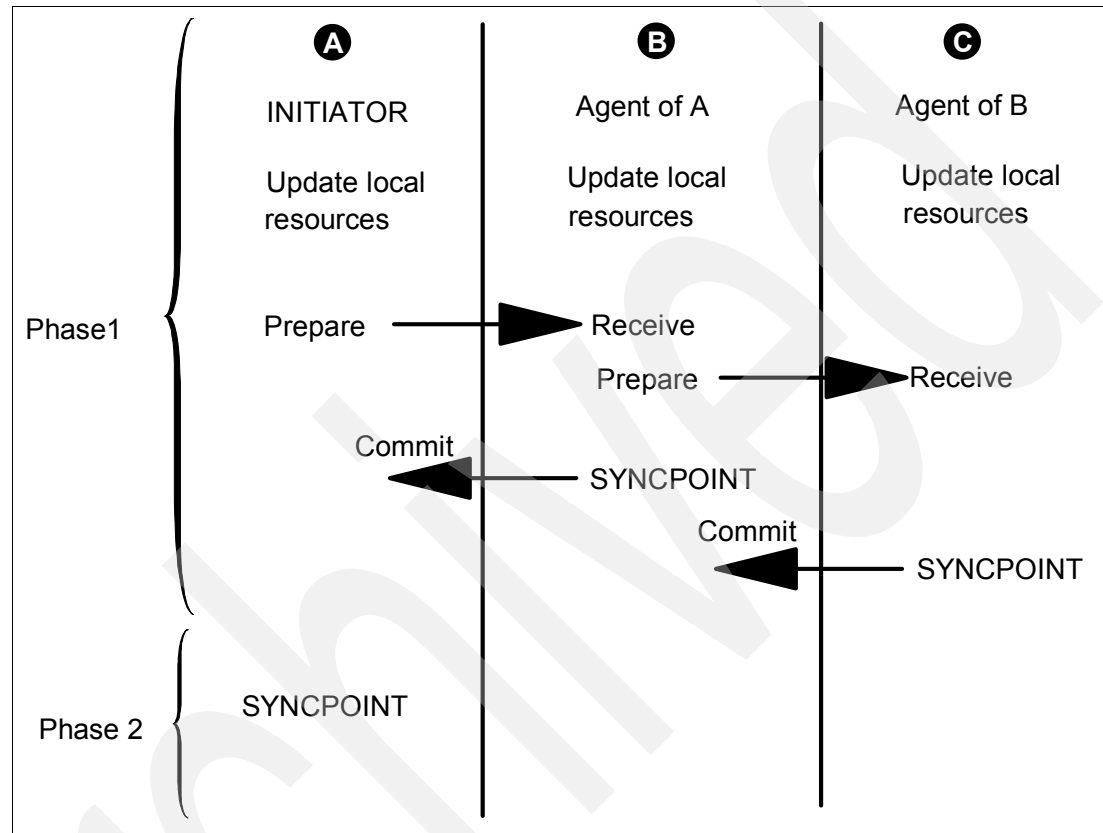


Figure 2-1 The two-phase commit protocol schema

The following example describes what happens during a two-phase commit. It is somewhat analogous to a person requesting an automated teller machine (ATM) to transfer money from a savings account to a checking account. The application program receives the input from the ATM. Each account is in a different database. To clarify our terminology, the ATM application runs under the control of a work manager (CICS, for example). Each database has its own resource manager (such as DB2 and IMS/DB).

The actions to process the ATM transaction, illustrated in Figure 2-2, are described here:

1. The ATM user requests the transfer of money from a savings account to a checking account.
2. The ATM application program receives the ATM input.
3. The ATM application requests that the savings database resource manager subtract the money from the savings database. For this function, the application uses the resource manager application programming interface (API).

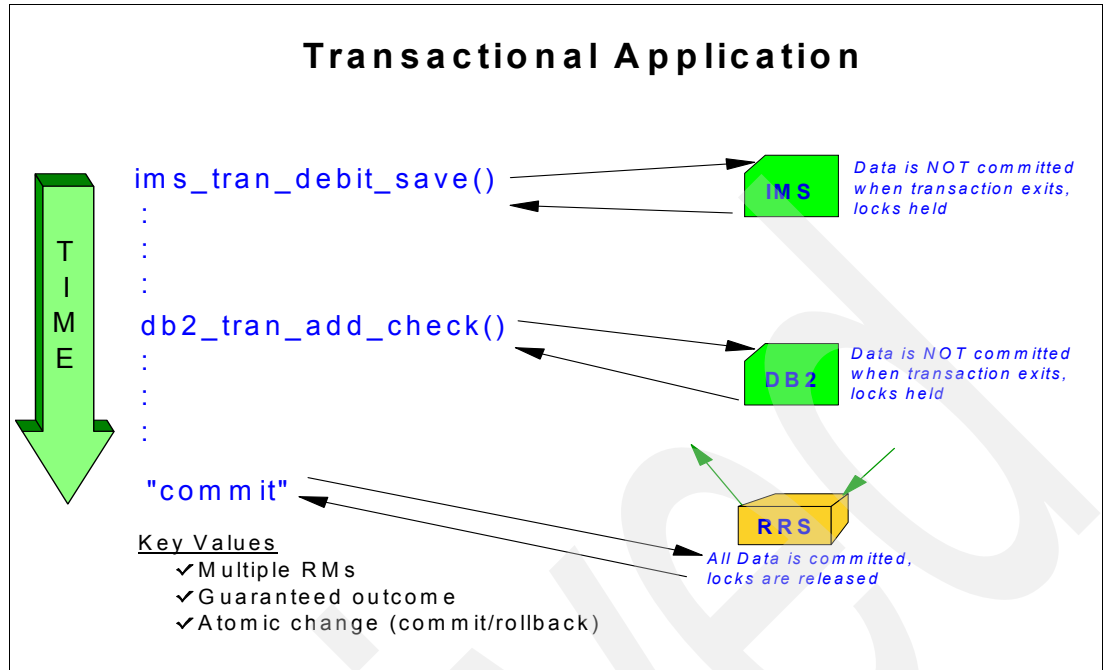


Figure 2-2 Flow of ATM transactional application - Image3

- The ATM application requests the checking database resource manager to add the money to the checking database. The application uses this resource manager's API.
- The ATM application needs to commit the database changes.

Now, assuming the two-phase commit protocol is supported by the work manager and both resource managers, the changes on both databases are represented by one UR. Because of that, as illustrated in Figure 2-3, *all* the changes are either committed or backed out.

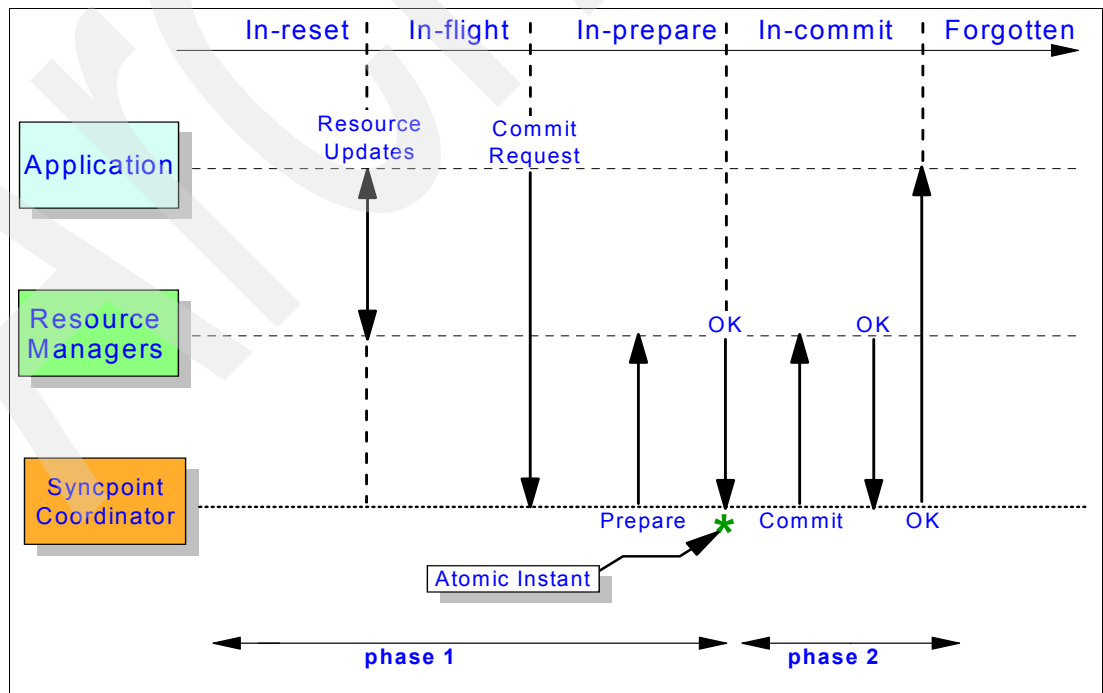


Figure 2-3 The two-phase commit protocol

6. The ATM application work manager, taking on the role of the syncpoint manager, asks the resource managers to prepare for the changes. This is phase 1.
7. The resource managers indicate whether or not they can make the changes by voting YES or NO.
8. If both answers are positive, the ATM application work manager notifies the resource managers to commit the changes; that is, the changes must be made permanent in the database.
9. The resource managers complete the commit and return OK to the ATM application work manager.
10. The application work manager gives a return code to the application program, indicating that all changes were made in the databases.
11. If any of the resource managers respond negatively in 8 (NO during the prepare step), the ATM application would have requested all the resource managers to back out the changes instead of committing.

2.2 Two-phase commit as supported by legacy resource managers

The traditional resource managers on z/OS (CICS, IMS and DB2) all support two-phase commit protocols. CICS, for example, supports full two-phase commit with IMS and DB2, and supports two-phase commit across distributed CICS systems. So why, you might ask, do we need a new syncpoint manager on S/390?

The problem is there are many restrictions imposed on application developers attempting to develop new applications that require updates in many different resource managers, perhaps across a number of systems. Many of these new applications use technologies like DB2 stored procedures and Enterprise Java™ Beans, and use client attachment facilities of CICS or IMS that do not support two-phase commit. If any of these resource managers are used by an application to update resources, it is not possible to have a global coordinator for the syncpoint.

The lack of a global syncpoint manager might have influenced some application design for the following reasons:

- ▶ The application would not have been capable of having complex and/or distributed transactions if all resource managers were not participating in the two-phase commit protocol.
- ▶ It could not have been designed as a single application (or unit of recovery) across multiple systems (except for CICS).
- ▶ The application had to program around these limitations.
- ▶ It could have limited the choice of where to put the business data in order to ensure that all the data could be committed in a single unit of recovery.
- ▶ It could have affected the recoverability of the protected resources and/or their integrity in case of a failure of one of the components, since there was no way to either commit or roll back all the updates.

Let's again consider our example of a person requesting an ATM function to move money between two accounts. The application receives the input from the ATM. Each account is in a different database. Each database has its own resource manager.

Before RRS, if the two resource managers did not participate in a two-phase commit protocol, it was possible that while the update on one database was successful, the other database could not be updated because of any failure along the path. This situation could create integrity problems in the consistency of the data.

One possible solution was to try to limit the application by using certain resource managers that could be coordinated. The other option was to introduce complex and manual recovery procedures. The introduction of RRS provides a third option.

The following sections give you a brief overview of how some traditional resource managers on z/OS work *without* RRS services.

2.2.1 CICS

CICS has always allowed you to create transactions with ACID properties (atomicity, consistency, isolation, and durability). In particular, the atomic property of a CICS transaction means that any changes the transaction makes to recoverable resources are carried out in such a way that either all happen, or none happen.

However, in releases of CICS prior to CICS Transaction Server, restrictions were imposed on distributed units of work. These restrictions were necessary because of the difficulties of coordinating updates to distributed resources. In CICS Transaction Server for OS/390, the CICS syncpoint manager allows you to distribute recoverable resources in a network, while ensuring that all updates made by a distributed unit of work are coordinated, thus preserving the transaction's atomicity.

A distributed unit of work is made up of two or more local units of work; each local unit of work represents the part of the distributed unit of work that relates to resources on one of the participating systems. The period during which one site is in doubt about whether a remote partner has committed or backed out its part of a distributed unit of work is known as the *in-doubt window*.

If a system or connection failure occurs during the in-doubt window, two things can be done: either an immediate decision is taken, or the local unit of work must wait until the doubt can be resolved correctly. In CICS releases prior to CICS Transaction Server for OS/390, CICS took an immediate decision, which could sometimes result in data in local and remote systems being inconsistent.

CICS Transaction Server for OS/390 has been changed so that it now permits local units of work that are in doubt to wait, pending the reestablishment of communication with remote sites. In summary, CICS Transaction Server for OS/390 removes the restriction that all recoverable resources should be placed in a single (file-owning) region. Resources can now be distributed in any way throughout a network; when failures occur, their integrity is preserved automatically by the CICS syncpoint manager, without the need for user intervention.

Applications written for releases of CICS before CICS TS continue to work in a CICS TS environment without any change. They automatically take advantage of the new CICS syncpoint manager support.

The CICS syncpoint manager communicates with its agents using interfaces to its local recovery connectors (RMCs). The RMCs are the communication resource managers (LU 6.2, LU 6.1, DB2, MRO, and RMI), which have the function of understanding the transport protocols and coordinating transactions between the connected systems.

There is still one exception in the CICS TS environment that existed until CICS TS 1.3. Before CICS TS 1.3, there was no two-phase commit protocol between the issuer of an EXCI call and the CICS TS region target of the call. Environments like DB2, batch and Web browsers requesting the execution of programs in the CICS region via the EXCI interface should not have updated any other non-CICS resources, since those updates were not covered by a two-phase commit protocol.

Today, after CICS Transaction Server 1.3, the unit of work within the CICS server program can update recoverable resources because it is now part of the RRS unit of recovery associated with the EXCI client program. In fact, the CICS server unit of work can be committed when the server program returns control to the client, or can continue over multiple EXCI DPL calls, until the EXCI client decides to commit or back out the unit of recovery.

2.2.2 IMS

IMS/ESA, like CICS, has always provided two-phase commit support for IMS transactions. IMS is comprised of two components: IMS/TM, which is a work manager, and IMS/DB, which is a data resource manager.

The IMS DB/TM system acts as the sync point coordinator for its resource managers and attached databases. Specifically, the DCCTL system acts as the sync point coordinator for attached databases (both IMS and non-IMS). The Coordinator Control (CCTL) acts as the sync point coordinator for units of work associated with the CCTL region. The data resource manager might be DBCTL or DB2.

When it comes to distributed transactions, IMS, like CICS, has restrictions that exist in the “traditional” environment. APPC/IMS supports distributed applications by allowing an application using APPC to connect to IMS and run a transaction which may update protected resources. APPC/IMS supports the CPI resource recovery Commit (SRRCMIT) and Backout (SRRBACK) calls for IMS-managed local resources. These protected resources include:

- ▶ IMS TM message-queue messages
- ▶ IMS DB databases
- ▶ DATABASE 2 (DB2) databases

Starting from IMS/ESA V6, the support for APPC protected conversations (SYNCLVL=SYNCPT) has been added to the transaction manager. This means that an application that updates protected IMS resources can now also update other resources and be sure that either all updates happen or none happen.

We shall see when we discuss IMS/ESA and its use of RRS that these restrictions have been addressed.

2.2.3 DB2

DB2 can act as both a data resource manager and as a work manager (when using DDF). Both CICS and IMS fully implement two-phase commit when updating DB2 data. In these cases, CICS and IMS act as the syncpoint managers and ensure that in-doubt units of work are resolved in the event of system failures. The issues involved with distributed transactions with CICS and IMS were previously discussed.

DB2 supports remote access by means of the Distributed Relational Database Architecture™ (DRDA®) protocol. This allows applications to access DB2 over APPC or TCP/IP connections, and this protocol supports two-phase commit. DRDA allows distributed units of

work across a number of DB2 servers—but no other type of resource manager can be involved.

A local or remote (via DRDA) program can use the SQL CALL statement to invoke a DB2 stored procedure. Applications that run in a stored procedures address space can access any resources available to z/OS address spaces, such as VSAM files, flat files, APPC/MVS conversations, and IMS or CICS transactions.

A stored procedure can be handed parameters and can return results to the calling application. This is very often used as a method of invoking a program that needs to access some non-DB2 data. For example, an application on a UNIX® machine could connect to DB2 on z/OS via DRDA and access DB2 data, and also invoke a stored procedure that accesses data in a VSAM file on z/OS.

The problem is that if a stored procedure needs to update a recoverable resource, and if you use DB2-managed stored procedures, DB2 will not manage syncpoint coordination across the resource managers. As described in Chapter 3, “Distributed RRS” on page 23, however, DB2 can use RRS to provide this function.

2.3 How RRS works

RRS is an *exit manager*. That means RRS itself does not perform tasks such as commit changes to a DB2 database or back out changes to an IMS database. Instead, RRS triggers resource manager-supplied exits based on certain events. An application can request RRS to commit a UR, and RRS then invokes the commit exits for all the resource managers involved in the UR.

The responsibility for coding the exits that RRS requires lies with the developer of the resource manager. It is important to note that this is not a concern for applications programmers writing client applications for use with resource managers that implement RRS support. As we shall see when discussing client application requirements when using various resource managers, RRS is largely hidden from the application programmer's view. Any application changes required are covered in the sections on each of the IBM-supplied resource managers.

Technically, by RRS we really mean Recoverable Resource Management Services (RRMS). RRMS consists of three services:

- ▶ Registration services
- ▶ Context services
- ▶ Resource Recovery Services (RRS)

In the following sections we describe registration services and context services. It is RRS that provides the means to implement two-phase commit, but a resource manager must use registration services and context services in conjunction with Resource Recovery Services.

2.3.1 Registration services

The first thing a resource manager must do in order to start using RRS is to identify itself to RRS and supply RRS with its exit routines. To do so, a resource manager uses registration services.

Depending on what resources a resource manager protects and what its role is, a resource manager will call the Set_Exit_Information service and provide for each exit it wishes to enable. A resource manager usually supplies exits for actions like PREPARE, COMMIT and

BACKOUT, and would use the registration service to notify RRS what exits are to be driven in the event of certain conditions occurring.

For more detailed information about registration services, refer to *z/OS MVS Programming: Resource Recovery*, SA22-7616.

2.3.2 Context services

Context services is an exit manager that allows resource managers to track the changes made by a given unit of work. It provides the data constructs and primitives that resource managers can use as an anchor to relate a unit of work to an application.

A *context* represents the environment that the application is running in and its work request. A work manager can create a context and have its application run under it. A work manager can actually create more than one context, but at any point of time there can only be one active context for a single task. If the work manager need to change the context, it has to explicitly switch to a different context.

Associated to the context is the unit of recovery that is currently running with all the information about the resource managers involved in this unit of work. A context may (and usually does) span several units of work. In fact, after you commit a unit of work, you can start another unit of work under the same context.

A context consists of the application program requesting the work, and the protected resources involved in the work. When an application program requests access to a resource, the resource manager might express an interest in the context associated with the application program and its work request. A resource manager that exploits RRS might need to express interest in a work context, not just a particular UR, because a context can persist over multiple URs.

A context can be either a native context or a privately-managed context, as explained here:

Native context

The automatically occurring context of the application program and protected resources associated with a work request. A native context is associated with a single application task. This context always exists and cannot be detached from the task and associated with any other task. A native context is implicitly created by the operating system.

Privately-managed context

Created by a resource manager. The resource manager (for example CICS, IMS or WebSphere Application Server) owns the privately-managed context it creates, and the resource manager can switch a privately-managed context from one task to another.

A privately-managed context is usually used by a resource manager, like IMS/TM, that accepts and manages work like transactions from outside the system. The privately-managed context acts as an anchor for the inbound application request. It can associate a transactional scope for the context, such that any resources that are actively manipulated under that context are in the same transactional scope. A native context may never be switched to another task. A privately managed context can be switched to any number of tasks.

Some of the things that are anchored in a privately managed context are user security, user characteristics (like codesets, languages time zones, currency denominations and so on), application trace indicators and so forth.

Table 2-1 on page 18 lists the differences between native context and privately-managed context.

Table 2-1 Differences between native and private-managed context

Difference	Native context	Privately-managed context
Creation of context	Implicitly	Explicitly by a resource manager
Association of context	Always with an application task	Temporarily may not be associated with any task
Association change	Cannot change association from one application task to another	Can change association from one application's task to another at any time. The change can even be to a work unit in a different home address space.
Context end	<ul style="list-style-type: none"> - When application's task ends - When a resource manager running under the application's task explicitly ends the context (though a new native context automatically begins) - When the home address space of the application's task abnormally ends 	<ul style="list-style-type: none"> - When a resource manager explicitly ends a privately-managed context that it owned. If the context is associated with an application's task, the resource manager must be running under that task. - If the owning resource manager ends or unregisters, a context disassociated from an application's task ends immediately. - If the owning resource manager ends or unregisters, a context associated with an application's task continues until the task ends. - If the application's task associated with a privately-managed context ends, the system invokes the PVT_CONTEXT_OWNER exit routine, if provided. The routine indicates if the privately-managed context is to be ended or disassociated from the task.

A resource manager that does work on behalf of an application must express an interest in the context associated with that application in order to be notified of commit/backout. The resource manager needs to express interest in contexts in order to get notified about events related to the work request, such as end context processing.

2.3.3 RRS invocation

Even though each resource manager that works with RRS is different, all must consider a common set of actions in order to use RRS services:

1. The resource manager must register to RRS.
2. After registering, the resource manager has to provide to RRS the exit routines that will be called to manage an event such as commit or backout.
3. The resource manager has to decide if it needs to create a privately managed work contexts for a unit of work.
4. The resource manager has to decide how to use RRS services to implement the two-phase commit protocol. Figure 2-4 on page 19 shows the interaction of RRS with the execution of a transaction.

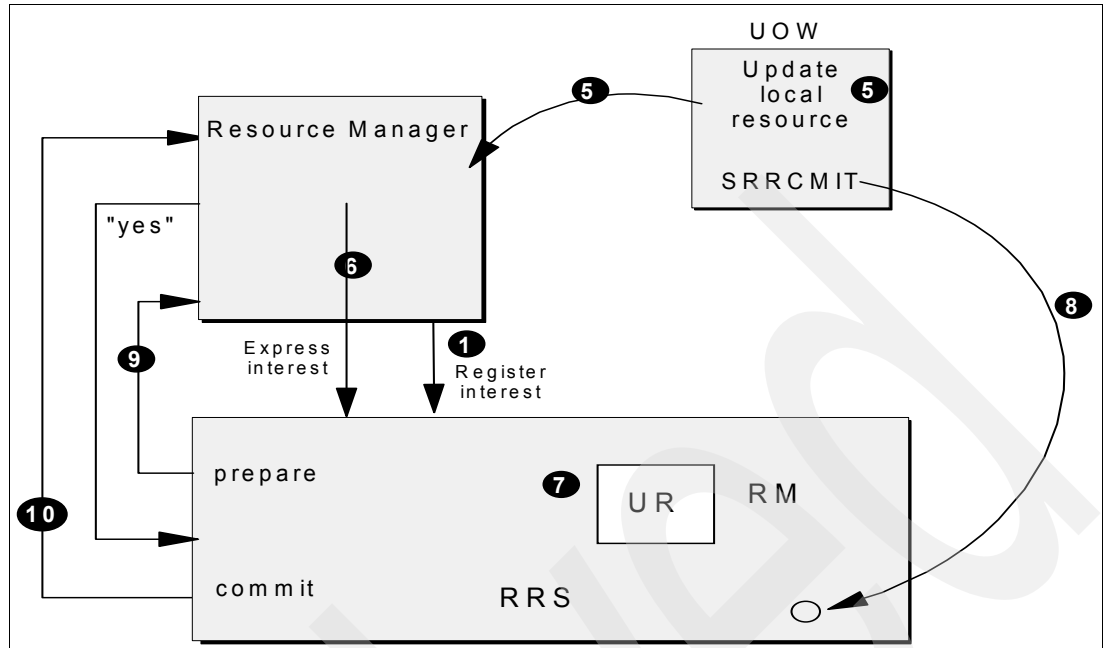


Figure 2-4 RRS interactions

5. A transaction is started, and during its execution it makes some changes to protected resources of the resource manager.
 6. Because of that, the resource manager calls RRS to express interest in the work context.
 7. RRS creates a unit of recovery (UR) to start representing the resource managers that are going to be involved at commit/backout time because they all have changes belonging to the same UR.
- Note:** The process identified by actions 1, 2, 3 and 4 can be repeated as many times as different resource managers are involved during the life of the transaction.
8. When the application reaches the syncpoint, it initiates the commit process by explicitly calling RRS to initiate the two-phase commit process using SRRCMIT or SRRBACK interface, or implicitly through the stub.
 9. RRS starts the two-phase commit protocol against all the resource managers that are related to the specific UR by sending the **prepare** command.
 10. If they all answer positively, RRS logs the “atomic” instant of commit and sends back the commit to all participants. If any respond negatively, RRS communicates to roll back all changes to all participants.

2.4 How two-phase commit works with RRS

In a recovery environment, each resource manager still relies on its own commit/backout mechanism to manage local resources. When an application makes changes to protected resources across different resource managers, RRS gets involved as a syncpoint manager across the interested resource managers that are acting as agents. In this case, RRS provides the “atomic” syncpoint coordination in the two-phase commit protocol.

The final objective of RRS is to provide transactional capability to the application, no matter which resources managers are involved and how they are involved—where transactional

capability identifies the possibility to either commit all the changes performed by the application across multiple resource managers, or none at all.

In this architecture, RRS and the current subsystem syncpoint managers work together to provide full recovery at the application level. Depending on the application design, in some of these solutions RRS acts as the global coordinator and the subsystem syncpoint managers act as agents; in other models, it might be vice versa.

The important thing is that, if an application needs an atomic syncpoint when it updates resources across multiple resource managers, all interested resource managers should support a two-phase commit protocol, either natively or through RRS. When the application is designed in such a distributed environment, the objective is to achieve a global transaction capability represented by a single unit of recovery, where all the application elements represented by the logical unit of work across different resource managers can be coordinated by a unique syncpoint manager.

This concept is illustrated in Figure 2-5 on page 20. There you can see a sample of a distributed application and how it can behave differently, depending on whether or not it uses RRS. In the first configuration, the connection from the APPC conversation, the DB2 Stored procedure is performed using CAF, so the whole application appears as two separate units of recovery (one is the APPC conversation, and the other is the DB2 STP and IMS) with two different syncpoint managers. That means that changes made by the two units of recovery can be either committed or backed out independently. This is because CAF does not support a two-phase commit protocol.

In the second configuration, the APPC conversation has been changed to use RRSAF in the connection toward DB2 stored procedures. In this case, the application is represented by a single unit of recovery, and all changes made across the different logical units of work are either committed or backed-out all together. This is because all the resource managers are participating in a two-phase commit protocol, either natively as IMS or through RRS as for APPC and RRSAF, and RRS is the syncpoint manager.

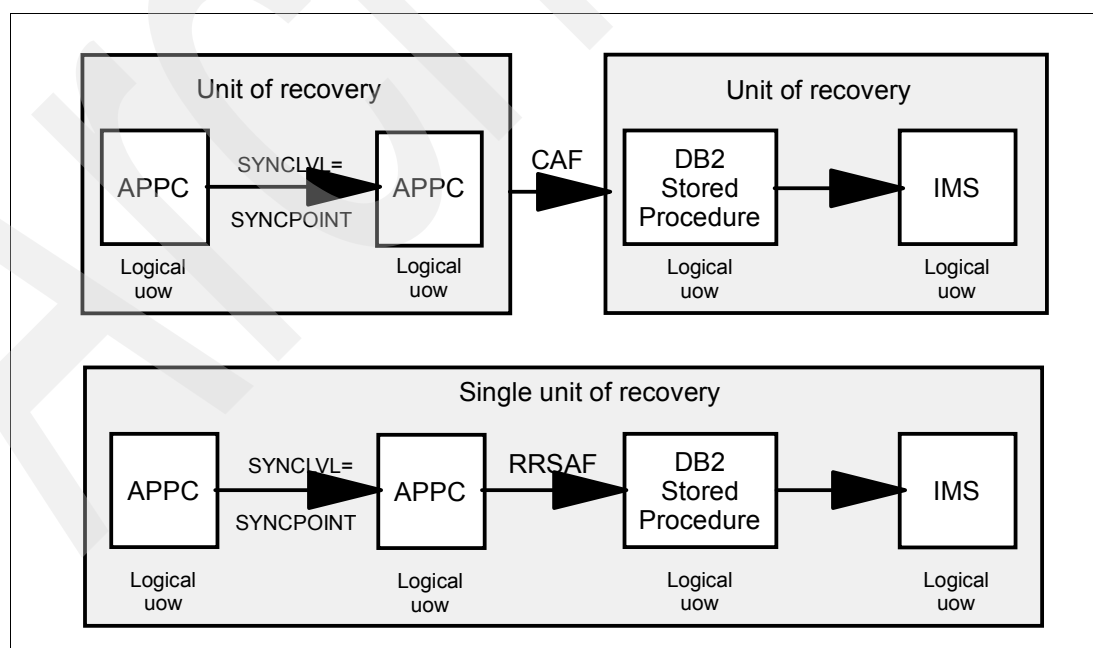


Figure 2-5 One-phase commit vs two-phase commit protocol

The two-phase commit diagram might slightly change when RRS comes into play. As illustrated in Figure 2-6, RRS can function as a syncpoint manager for resource managers that do not currently participate in the two-phase commit protocol. In this new picture, once an application requests that its resource updates be committed, RRS works with the resource managers to determine if the request can be honored. If the request can be honored, RRS works with the resource managers to ensure that all updates are committed.

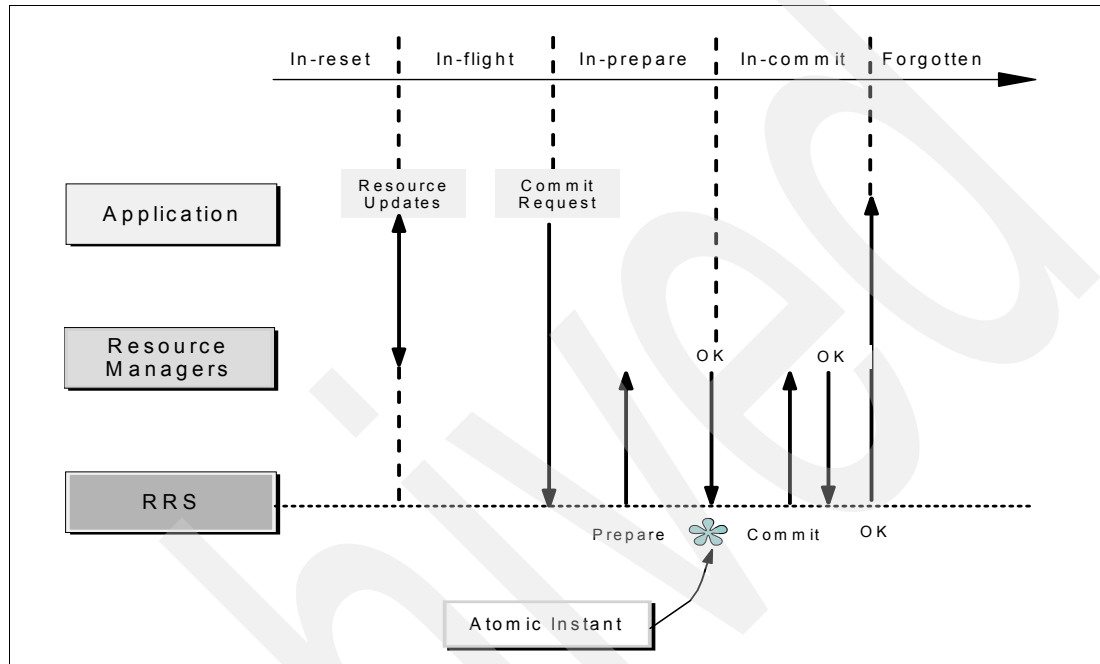


Figure 2-6 Two-phase commit protocol with RRS

In phase 1 of the two-phase commit process, RRS ask each resource manager if it can guarantee that it can make its update. If they all agree, RRS records that the changes can be committed. When the action of logging the commit request is done, the atomic instant has been reached.

After the atomic instant is reached for a given unit of recovery, there is no way to fall back. The resource managers are required to commit the resources updated by the application. This means that even if a failure occurs past the atomic instant, the resource managers and RRS must work together to ensure that the resources are eventually committed, regardless of the reason for the failure (note that this effort may occur a considerable amount of time after the commit attempt).

2.5 Summary

This chapter gave an outline of what the two-phase commit process is, and of the role RRS plays as a syncpoint manager on z/OS.

In the remaining chapters, we look at resource managers that exploit RRS and discuss each in some detail. We deal with work managers, data resource managers, and a communications resource manager. Remember that subsystems like CICS, IMS and DB2 may assume the role of a work manager or data resource manager, depending on the context in which we examine them. Therefore, you may find elements of these subsystems discussed in a number of chapters, depending on the function they perform.

Archived

Distributed RRS

In this chapter we discuss the distributed environment, in which a transaction might span multiple work managers and resource managers across multiple systems. We also describe how Resource Recovery Services (RRS) comes into play to handle this unit of work as a single unit of recovery (UR).

This chapter covers the following topics:

- ▶ “Distributed two-phase commit” on page 24
- ▶ “RRS distributed syncpoint support” on page 25
- ▶ “Multisystem cascaded transactions” on page 26

3.1 Distributed two-phase commit

In a distributed environment, you might want to have a transaction that spans multiple work managers and resource managers across multiple systems be handled as one unit of recovery (UR).

Most of the single resource manager concepts remain unchanged even in a distributed resource recovery environment, where a work request can be distributed across more than one system. Each resource manager is normally capable of either committing or backing out changes that it makes to its own recoverable resources.

However, in a distributed resource recovery environment, not only must there be a syncpoint manager on each system to coordinate the local resource managers, but also one of the syncpoint coordinators must also act as the overall syncpoint manager for the UR. That syncpoint manager must communicate with the other syncpoint managers using the full two-phase commit protocol.

In Figure 3-1, imagine a transaction running on Work Manager A that updates resources in Resource Managers A1 and A2. The transaction also runs remote transactions on System B and System C, and wants all updates done by the remote resource managers to be handled as one UR.

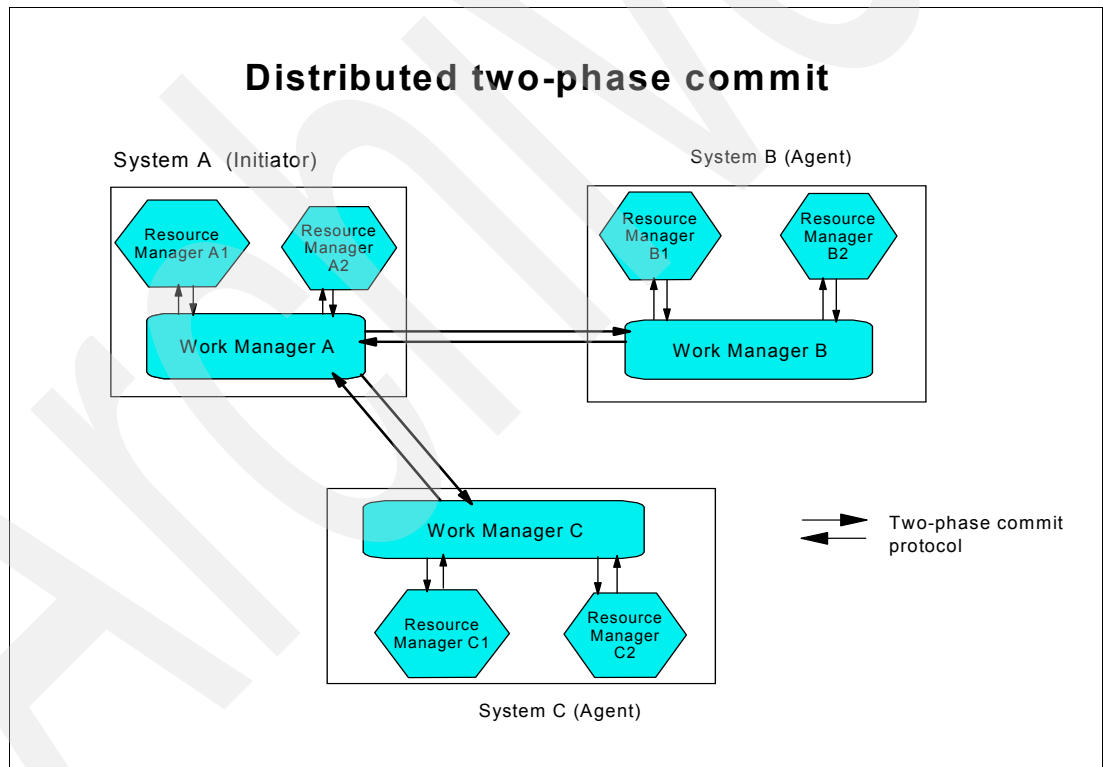


Figure 3-1 Distributed two-phase commit

When the transaction running on Work Manager A decides to commit the work, the following needs to happen:

1. Work Manager A assumes the role of the global syncpoint manager for the complete UR, as well as the role of the syncpoint manager for the local resource managers. It sends a "prepare to commit" message to the local resource managers and also to the work managers on System B and System C.

2. The work managers on System B and System C act as the local syncpoint managers for the resource managers on their respective systems. Each work manager sends the “prepare to commit” to the local resource managers and checks the responses received.
3. The work managers on System B and System C must now reply to the “prepare to commit” received from System A in step 1. If their local resource managers have confirmed they are ready to commit, then System B and System C reply “ready to commit” to System A.
4. The work manager on System A then makes the overall decision to commit or back out based on the responses from the local resource managers on System A and the work managers on System B and System C.
5. Assuming everyone has replied “ready to commit”, then the work manager on System A logs the decision and sends the commit request to its local resource managers and the work managers on System B and System C.
6. On receipt of the commit request from System A, the work managers on System B and System C send a commit request to their local resource managers. When they get a “commit confirmed” from the local resource managers, they return “commit confirmed” to the work manager on System A.
7. The work manager on System A returns an “OK” to the application when it has received the commit confirmed from System B and System C and its own local resource managers.

In this scenario, we call the work manager on System A the *initiator* and the work manager on System B and System C *agents*.

In a distributed resource recovery environment, no matter how many distributed elements are involved, an application should be designed in a way that only one of the transaction programs initiates syncpoint activity (commit or back out) for the distributed unit of recovery.

3.1.1 RRS distributed syncpoint support

To support this distributed environment, RRS allows a resource manager to operate as syncpoint coordinator. In RRS terms there are two roles a resource manager can take: a distributed syncpoint resource manager (DSRM), and a server distributed syncpoint resource manager (SDSRM).

In both cases the key point is that once all the resource managers involved on a system vote to commit, RRS does not drive the commit exits for the involved resource managers until the DSRM/SDSRM resource manager tells it to do so. Note that this is different from the behavior of RRS when it is the syncpoint coordinator—when RRS is acting as the syncpoint coordinator, as soon as all the resource managers vote yes to commit, then RRS drives the commit exits for those resource managers.

The DSRM role is suited to a peer-to-peer relationship, where any system involved in the multisystem UR can be the initiating system. The *initiating system* is the one where the commit request is first issued.

The SDSRM role is suited to a client/server relationship where one system (the client) is always the initiating system and all the other systems (the servers) are agent systems. The client always initiates the commit/backout. A resource manager that takes on the SDSRM role uses RRS calls that are specific to an SDSRM.

You can refer to *z/OS MVS Programming: Resource Recovery*, SA22-7616, for greater detail on the differences between a DSRM and an SDSRM. However, it is not necessary to understand the fine details unless you are going to write your own communications resource

manager, where a communications resource manager is intended to be a manager that coordinates a two-phase commit across multiple nodes in a distributed transaction.

When you view RRS panels or logs, you will notice a Role column; this is set to Participant, DSRM, or SDSRM. Example 3-1 displays an RRS Unit of Recovery Details panel showing a UR involving three resource managers.

Example 3-1 RRS Unit of Recovery Details panel showing SDSRM role

```
RRS Unit of Recovery Details          Row 1 to 3 of 3
Command ==>                          Scroll ==> PAGE

Commands r-Remove Interest v-View URI Details

UR identifier : BA6945ED7E8F000000001FB010F0000
Create time : 2003/12/02 19:08:40.195258      Comments :
UR state : InFlight      UR type : Prot
System : SC48      Logging Group : WTSCPLX1
SURID : N/A
Work Manager Name : BBO.CLHA1.CLUA11.WSA11.IBM
  Display Work IDs          Display IDs formatted
  Luwid . . : Present
  Eid . . : Not Present
  Xid . . : Present
Expressions of Interest:
S  RM Name                  Type  Role
   IMS.IM4B___V081.STL.SANJOSE.IBM Prot Participant
   DFHRXDM.SCSCERW1.IBM    Prot Participant
   BBO.CLHA1.CLUA11.WSA11.IBM Prot  SDSRM
```

This example shows an IMS system with a role of Participant, and a CICS system with a role of Participant. It also shows a WebSphere for z/OS server with a role of SDSRM.

So why does WebSphere take the SDSRM role? Well, in this case we are running an EJB™ application in WebSphere that accesses IMS and CICS. The application also accesses resource managers on another system—and WebSphere needs to coordinate commit processing with those remote systems. WebSphere needs to be able to make the final decision about whether to commit or not, based on input from the external systems. These external systems may or may not be z/OS systems.

3.1.2 Multisystem cascaded transactions

Prior to z/OS V1.2, there was no support in RRS for a UR to be coordinated by RRS across multiple z/OS systems in a sysplex. Distributed two-phase commit processing across z/OS systems was implemented using a communications resource manager that used the RRS DSRM/SDSRM support.

Then in z/OS V1.2, RRS introduced support for multisystem cascaded transactions. A *cascaded transaction* is one in which several units of recovery can exist and can be managed by RRS, so as to act as a single transaction. Because each UR can have a separate work context, cascaded transactions may span multiple systems in a sysplex.

A *cascaded UR* consists of one parent UR and one or more child URs. Cascaded URs can be embedded in other cascaded URs, but there is always one top-level UR. A *UR family* is the collection of all the URs linked by a parent/child relationship. Changes made by all the URs in the UR family are either all committed, or all backed out.

A child UR may reside on a different z/OS system from that of the parent UR. In RRS terminology, the system where the top-level UR of a UR family resides is called the *coordinator* and the system where a child UR resides is called a *subordinate*.

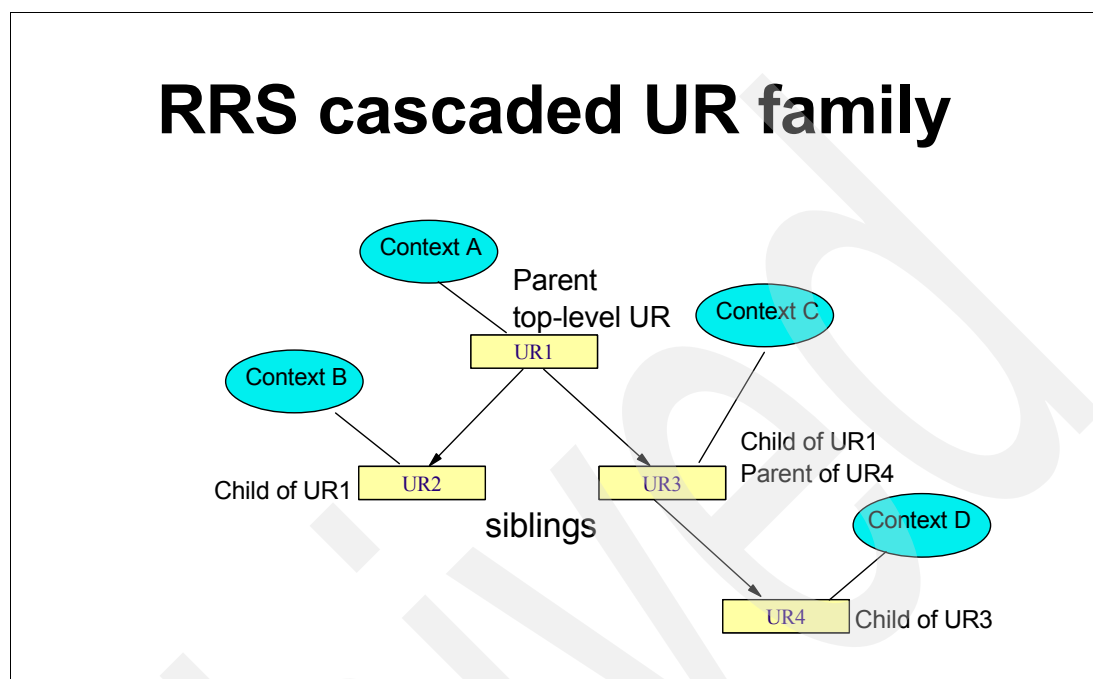


Figure 3-2 RRS cascaded UR family

Typically, a work manager needs to create a cascaded UR when a single work request involves multiple work managers that may be distributed across a sysplex.

Cascaded UR support in RRS requires that all systems involved be in the same RRS logging group. Refer to 4.2.1, “RRS logging group name” on page 33, for a description of an RRS logging group.

An example of a work manager that uses cascaded UR support is IMS V8, which uses this function to provide full shared queue support for synchronous APPC and OTMA workloads across a sysplex. This allows the work to be distributed and executed on any IMS system in the shared queue group in a sysplex.

For a detailed discussion of cascaded URs, refer to *z/OS MVS Programming: Resource Recovery*, SA22-7616.

Archived



Part 2

Implementing and managing RRS

In this part we describe the tasks required to set up RRS, how to operate in the RRS environment, how to use the ISPF panels to debug potential problems, and how RRS handles recovery and restart situations.

Archived

Implementing RRS

In this chapter, we provide an overview of the tasks required to implement RRS. The chapter contains basic “how-to” information for system setup and operation in an RRS environment. It also describes our experiences with sample workloads, showing how RRS can be used in different environments.

This chapter covers the following topics:

- ▶ “Define the logging environment” on page 32
- ▶ “Define the RRS infrastructure” on page 42

Refer to “Managing RRS” in *z/OS MVS Programming: Resource Recovery*, SA22-7616, for more information about implementing RRS.

4.1 RRS Implementation overview and planning

The RRS environment includes the RRS address space, System Logger log streams, and a set of ISPF panels. System Logger must be active on all systems where RRS is to be run. Refer to *System Programmer's Guide to: z/OS System Logger*, SG24-6898, for details on implementing System Logger.

Before you can start RRS, you must perform the tasks described in Table 4-1.

Table 4-1 RRS setup tasks

Task	Required
Define the RRS log streams	Required
Establish the priority for the RRS address space	Required
Define RRS as a subsystem	Required
Create procedures to start RRS	Required
Set up automation to restart RRS	Optional
Define RRS security definitions	Optional
Enable RRS ISPF panels	Optional (enables ISPF application to look at RRS log streams)
Set up RRS component trace	Optional

Before you begin these tasks, you need to make certain planning decisions about RRS, such as:

- ▶ What log streams your installation will use
- ▶ What RRS logging group name to use
- ▶ What type of log stream (DASD-only or Coupling Facility)
- ▶ The size of the log streams

4.2 Define the logging environment

There are five RRS log streams. All are *required* except the ARCHIVE log stream. Required means that RRS does not start without being able to connect to these log streams. RRS performs all the logging in the log streams; the resource managers can provide persistent interest data that RRS logs, but RRS does the actual logging. Table 4-2 summarizes the log streams and their contents.

Table 4-2 RRS log streams and their content

Log stream type	Log stream name	Content
RRS archive log	ATR.lgname.ARCHIVE	Information about completed URs. This log is optional. See Note for further details.
RRS main UR state log	ATR.lgname.MAIN.UR	The state of active URs. RRS periodically moves this information into the RRS delayed UR state log when UR completion is delayed.

Log stream type	Log stream name	Content
RRS resource manager data log	ATR.lgname.RM.DATA	Information about the resource managers using RRS services.
RRS delayed UR state log	ATR.lgname.DELAYED.UR	The state of active URs, when UR completion is delayed.
RRS restart log	ATR.lgname.RESTART	Information about incomplete URs needed during restart. This information enables a functioning RRS instance to take over incomplete work left over from an RRS instance that failed.
<p>Note: If the writes activity to the RRS ARCHIVE log stream is very high, this might impact the performance throughput of ALL RRS transactions if this log stream is defined and actively in use by RRS. This log stream is fully optional and only needed by the installation for any type of post transaction history type of investigation.</p> <p>To avoid this performance impact the installation can DELETE the archive log stream. To do so, the log stream must be disconnected from RRS on ALL systems and then, use the IXCMIAPU utility, DELETE LOGSTREAM NAME(ATR.<groupname>.ARCHIVE) to delete the log stream definition and, using the IDCAMS utility, DELETE IXGLOGR.ATR.<groupname>.ARCHIVE.* SCRATCH to delete the data set associated with the log stream.</p> <p>If you choose not to use the ARCHIVE log stream, a warning message is issued at RRS startup time about not being able to connect to the log stream (however, RRS will continue its initialization process).</p>		

4.2.1 RRS logging group name

The RRS images on different systems in a sysplex run independently, but share log streams in order to keep track of transactions. An *RRS logging group* is a group of systems that share an RRS workload. To define a logging group, use the GNAME parameter on the procedure used to start RRS. If you omit the GNAME parameter, the default logging group name is the sysplex name.

RRS on each system (there can only be one RRS active on a system) in a sysplex can be in, at most, one logging group. Within the same logging group, if a system or RRS fails, RRS on a different system can use the shared logs to take over the failed system's outstanding work, thereby enabling quick recovery from system and RRS failures.

The GNAME is completely transparent to the RMs. The RMs do not send a call to a specific RRS (unlike DB2, for example, where callers would identify which DB2 they want to talk to). On the other hand, all the information in RRS is associated with a particular GNAME, so if you change GNAMEs, all the RRS information from the old GNAME is no longer accessible.

Note: Resource managers do not know about the GNAME. An RM registers with RRS. The installation decides the GNAME name. RMs can provide a logname to RRS and RRS will provide a logname to an RM, but the RRS logname is basically a time stamp; no GNAME is involved.

There are advantages to using multiple RRS logging groups:

- You can use different log groups to subdivide the transaction work in a sysplex. (For example, you can use separate logging groups for test systems and production systems.)

- You can restart RRS with a different log group name to cause a cold restart and keep the data in the old logs for debugging and data recovery purposes. (The RRS panels allow you to browse any set of logs; you only need to specify the GNAME on the panel. However, this option is only meaningful for recovery purposes.)

4.2.2 Log stream characteristics

RRS supports both Coupling Facility log streams and DASD-only log streams. A DASD-only log stream has a single-system scope and cannot be used in a multisystem sysplex environment except in particular circumstances. For example, you might have an instance of RRS on a test image that uses its own logging group that is not shared with any other system in the sysplex. In this particular configuration, RRS can use DASD-only log streams. Usually, either for restart issues or because of the workload configuration, RRS is configured to use Coupling Facility log streams.

All instances of RRS in the same logging group must have access to the Coupling Facility structures and log stream data sets used by the RRS log streams for that logging group. This allows other RRS instances in a sysplex to access data in the event of failure of an RRS instance or system. This is required to permit resource managers to be restarted on different systems in a sysplex.

4.2.3 RRS log stream structure sizing

Table 4-3 provides initial considerations on the amount of storage required for the RRS log streams. These recommendations should result in reasonably efficient usage of Coupling Facility storage, while minimizing the likelihood that you will have to redefine the structures due to the variations in your workload. However, the exact amount of storage you need for each log stream depends on the installation's RRS workload; SMF88 data can be used to understand if the structure sizes require any adjustments. Refer to the section entitled "Estimating Log Stream Sizes" in the IBM Redbook *Systems Programmers Guide to System Logger* for more information about sizing log streams.

Prior to starting RRS for the first time, you can get an estimate of the required structure sizes by using the CF Sizer tool, available on the Web at:

<http://www.ibm.com/servers/eserver/zseries/pso>

We used the values shown in Table 4-3 as input to the CF Sizer tool.

Table 4-3 Sample I/O activity for the RRS log streams

Log stream	Writes per sec	Storage requirements
RM.DATA	2	Low, if few resource managers. Medium, if many resource managers.
RESTART	10	Medium
MAIN.UR	50	High
DELAYED.UR	10	High
ARCHIVE	50	Low

It is still a good practice, after you have run some workload, to reevaluate the log stream allocation sizes through SMF type 88 records. We suggest mapping each Coupling Facility log stream to a unique Coupling Facility structure. The Coupling Facility structures must be defined in the CFRM policy. Log streams are mapped to those structures through the LOGR policy.

If your installation has a constraint on the number of Coupling Facility structures in your CFRM policy, you can group multiple RRS log streams in a single Coupling Facility structure.

In that case, the following grouping is suggested:

- ▶ Place the RM.DATA and RESTART log streams in one structure.
- ▶ Place the MAIN.UR, DELAYED.UR, and ARCHIVE (if used) into another structure.

RRS log streams are active log streams with the exception of the ARCHIVE. In this case, “active” means that RRS manages the content of its log streams and keeps it up to date with the current workload running on the system. As a result, these log streams should not require a great deal of storage in the interim storage medium, and should not generate many offload operations.

In contrast to the other log streams, the ARCHIVE is a funnel-type log stream, containing a record for each completed transaction. A *funnel-type* log stream is one in which RRS only writes to the log stream and never reuses or deletes the ARCHIVE log records. For this reason, you should expect this log stream to use offload data sets. The volume of data in the log stream can be managed through a combination of AUTODELETE(YES) and RETPD values set to the number of days you want to keep this data in your installation.

4.2.4 Define the RRS log streams

RRS supports both Coupling Facility log streams and DASD-only log streams. The following list describes the steps required to set up RRS log streams. If you are using DASD-only log streams, you can skip step 2 and step 3.

The following tasks must be completed:

1. Verify the DFSMS definitions required for staging and offload datasets.
2. Define the Coupling Facility structures to the CFRM policy and activate the new policy after that has been updated.
3. Define the structures to the System Logger policy.
4. Define the log streams to the System Logger policy.

Verify DFSMS definitions required for RRS

To ensure successful operation, the data sets used by System Logger must be set up with the correct attributes. To ensure this, the SMS constructs used by System Logger must be correct.

- ▶ Determine the naming conventions for the RRS log streams and log stream data sets. The log stream names are `ATR.gname.logstreamname`, while the default data set names for offload and staging are `IXGLOGR.ATR.gname.logstreamname` where *gname*, the group name, defaults to the SYSPLEX name. You can specify the high level qualifier for the data sets using the HLQ or EHLQ parameter on the log stream definition in the System Logger policy.
- ▶ Ensure that the correct SMS classes are assigned to the System Logger data sets, either by specifying the SMS class names in the Logger policy, or by coding appropriate SMS ACS routines. In particular, ensure that SHAREOPTIONS(3,3) is specified in the Data Class to avoid data gap or data loss conditions.

Define the Coupling Facility structures in the CFRM policy

This task is only required if you are defining Coupling Facility-based log streams.

If you are defining Coupling Facility-based log streams, then each log stream needs to be mapped to a Coupling Facility structure. Coupling Facility structures are defined in the CFRM policy. Log streams are mapped to these structures in the LOGR policy.

In the sample shown in Example 4-1, we have created a Coupling Facility structure for each log stream.

Example 4-1 CFRM policy sample

```
//MAINSTR JOB CLASS=A,MSGCLASS=A
//POLICY EXEC PGM=IXCMIAPU
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
  DATA TYPE(CFRM)
  STRUCTURE NAME(RRS_MAINUR_1)
    SIZE(4096)
    INITSIZE(1024)
    PREFLIST(FACIL01, FACIL02)

  STRUCTURE NAME(RRS_DELAYED_1)
    SIZE(24576)
    INITSIZE(8192)
    PREFLIST(FACIL01, FACIL02)

  STRUCTURE NAME(RRS_ARCHIVE_1)
    SIZE(24576)
    INITSIZE(6144)
    PREFLIST(FACIL01, FACIL02)

  STRUCTURE NAME(RRS_RMDATA_1)
    SIZE(4096)
    INITSIZE(1024)
    PREFLIST(FACIL01, FACIL02)

  STRUCTURE NAME(RRS_RESTART_1)
    SIZE(28672)
    INITSIZE(9216)
    PREFLIST(FACIL01, FACIL02)
/*
```

Define the structures in the System Logger policy

This task is only required if you are defining Coupling Facility-based log streams.

After the structures are defined in the CFRM policy, the next step is to define the structure(s) in the System Logger policy. This definition is used by System Logger to match log streams to Coupling Facility structures. Example 4-2 shows input to the IXCMIAPU utility to define the structures to the System Logger policy.

Example 4-2 Structure definition in the System Logger policy

```
/DEFSTREXCPGM=IXCMIAPU
//SYSPRINT DDSYSOUT=A
//SYSINDD *
  DATA TYPE(LOGR)

  DEFINE STRUCTURE NAME(RRS_MAINUR_1)
    LOGSNUM(1)
    AVGBUFSIZE(158)
```



```

MAXBUFSIZE(65276)

DEFINE STRUCTURE NAME(RRS_DELAYED_1)
  LOGSNUM(1)
  AVGBUFSIZE(158)
  MAXBUFSIZE(65276)

DEFINE STRUCTURE NAME(RRS_ARCHIVE_1)
  LOGSNUM(1)
  AVGBUFSIZE(262)
  MAXBUFSIZE(65276)

DEFINE STRUCTURE NAME(RRS_RMDATA_1)
  LOGSNUM(1)
  AVGBUFSIZE(252)
  MAXBUFSIZE(1024)

DEFINE STRUCTURE NAME(RRS_RESTART_1)
  LOGSNUM(1)
  AVGBUFSIZE(158)
  MAXBUFSIZE(65276)
/*

```

MAXBUFSIZE/AVGBUFSIZE

MAXBUFSIZE, in conjunction with AVGBUFSIZE, is used to determine the CF structure ENTRY/ELEMENT ratio. When data is written to the CF, it is written in increments equal to ELEMENT size. A MAXBUFSIZE greater than 65276 gives an element size of 512; a MAXBUFSIZE equal to or less than 65276 results in an element size of 256.

We recommend running for a time with the AVGBUFSIZE values shown in Table 4-4, then adjusting the AVGBUFSIZE value to match the value shown in the IXCMIAPU report.

Table 4-4 Suggested starting AVGBUFSIZE values

Log stream	Recommended starting AVGBUFSIZE
ARCHIVE	262
DELAYED.UR	158
MAIN.UR	158
RESTART	158
RM.DATA	252

Sample JCL to produce the IXCMIAPU report is shown in Example 4-3.

Example 4-3 JCL to get IXCMIAPU Logger Policy report

```

//LOGRLIST JOB (0,0),'LIST LOGR POL',CLASS=A,REGION=4M,
//          MSGCLASS=X,NOTIFY=&SYSUID
//STEP1    EXEC PGM=IXCMIAPU
//SYSPRINT DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//SYSIN    DD *
          DATA TYPE(LOGR) REPORT(YES)

```

It does not matter if the defined AVGBUFSIZE does not exactly match the average buffer size as reported by IXCMIAPU, because System Logger dynamically adjusts the Entry/Element ratio. System Logger will adjust the ratio to avoid potential problems, especially if you do not share the same structure among multiple log streams, each of which could have different characteristics.

Define the log streams to the System Logger policy

This step is required for both DASD-only log streams and Coupling Facility log streams. Samples for both definitions are illustrated here, since some parameters differ.

Coupling Facility log stream definition

Example 4-4 shows input to the IXCMIAPU utility to define log streams to LOGR for log streams using Coupling Facility structures.

Example 4-4 Coupling Facility log streams definition in the System Logger policy

```
/DEFSTREXCPGM=IXCMIAPU
//SYSPRINT DDSYSOUT=A
//SYSINDD *
  DATA TYPE(LOGR)

  DEFINE LOGSTREAM NAME(ATR.PLEX1.MAIN.UR)
    LOWOFFLOAD(60)
    HIGHOFFLOAD(80)
    STG_DUPLEX(YES)
    DUPLEXMODE(COND)
    HLQ(test)
    LS_SIZE(1024)
    LS_DATACLAS(vsam1s)
    STG_SIZE(1024)
    STRUCTNAME(RRS_MAINUR_1)

  DEFINE LOGSTREAM NAME(ATR.PLEX1.DELAYED.UR)
    LOWOFFLOAD(60)
    HIGHOFFLOAD(80)
    STG_DUPLEX(YES)
    DUPLEXMODE(COND)
    HLQ(test)
    LS_SIZE(960)
    LS_DATACLAS(vsam1s)
    STG_SIZE(960)
    STRUCTNAME(RRS_DELAYED_1)

  DEFINE LOGSTREAM NAME(ATR.PLEX1.ARCHIVE)
    LOWOFFLOAD(0)
    HIGHOFFLOAD(80)
    STG_DUPLEX(NO)
    HLQ(test)
    LS_SIZE(960)
    LS_DATACLAS(vsam1s)
    STRUCTNAME(RRS_ARCHIVE_1)
    RETPD(2)
    AUTODELETE(YES)

  DEFINE LOGSTREAM NAME(ATR.PLEX1.RM.DATA)
    LOWOFFLOAD(60)
    HIGHOFFLOAD(80)
    STG_DUPLEX(YES)
    DUPLEXMODE(UNCOND)
```

```

HLQ(test)
LS_SIZE(192)
LS_DATACLAS(vsam1s)
STG_SIZE(192)
STRUCTNAME(RRS_DATA_1)

DEFINE LOGSTREAM NAME(ATR.PLEX1.RESTART)
  LOWOFFLOAD(60)
  HIGHOFFLOAD(80)
  STG_DUPLEX(YES)
  DUPLEXMODE(COND)
  HLQ(test)
  LS_SIZE(960)
  LS_DATACLAS(vsam1s)
  STG_SIZE(960)
  STRUCTNAME(RRS_RESTART_1)

```

AUTODELETE and RETPD

Note: Use only AUTODELETE(NO) and RETPD(0) for all RRS log streams except ARCHIVE. Why? Because they can have a disastrous effect if specified otherwise.

With settings AUTODELETE(YES) and RETPD>0, even though RRS will attempt to delete unnecessary log entries, all data will be offloaded to the offload data sets and held for the number of days specified for RETPD. AUTODELETE(YES) allows the System Logger (rather than RRS) to decide when to delete the data.

When a new offload data set is allocated and AUTODELETE(YES) is specified, the System Logger will delete the data on the old offload data set that has passed the retention period. Since data in the MAIN.UR log stream is managed by RRS, it is better to let RRS manage the life of the records on this log stream, in order to avoid the risk that RRS will need records that have been deleted by System Logger because of the AUTODELETE option.

For the ARCHIVE log stream, RRS never uses information written on the Archive log; the information is intended for installation use if a catastrophic problem occurs. You must use retention period and autodelete support to delete old entries; specify these values large enough to manage this log stream to a reasonable size and to provide enough coverage in time to recover in any potential situation.

HIGHOFFLOAD

The HIGHOFFLOAD parameter is used to determine when the space dedicated to the log stream in the Coupling Facility is filling up and an offload needs to be initiated to regain available space. HIGHOFFLOAD should be set at 80% for all RRS log streams, at least initially; then use the SMF88 report to evaluate whether this value needs to be tuned.

HLQ

This parameter specifies the up to 8-byte high level qualifier for both the log stream data set name and the staging data set name. HLQ must be 8 alphanumeric or national (\$,#,or @) characters, padded on the right with blanks if necessary. The first character must be an alphabetic or national character.

LOWOFFLOAD

The LOWOFFLOAD parameter defines the amount of data which may be retained in the log stream interim storage following an offload process. In the RRS environment, the

LOWOFFLOAD value should be high enough to retain the data required for backout of the UR, but low enough to keep the number of offloads to a minimum.

LOWOFFLOAD should be set between 20% and 60% for all RRS log streams as described in the examples, and at 0% for ARCHIVE.

LS_SIZE

LS_SIZE defines the allocation size for the *offload* data sets. It should be specified large enough to contain several offloads, possibly a day's worth. All RRS log streams (except ARCHIVE) should only offload a minimal amount of data.

Attention: If LS_SIZE is not specified in the log stream definition, and an extent size is not specified in the data class pointed to by LS_DATACLAS, then this value is taken from the ALLOCxx Parmlib member or set via an Automatic Class Selection (ACS) routine. The default value in ALLOCxx is 2 tracks. For more information, refer to *z/OS MVS Initialization and Tuning Reference*, SA22-7592.

It is very important to remember that log stream staging and offload (log) data sets are single extent VSAM linear data sets, and the shareoptions *must* be specified as '3,3'. If the shareoptions are anything other than '3,3' then there is a risk the System Logger will be unable to read offloaded data and provide RRS with return code 8, reason code 804 (IlgRsnCodeNoBlock).

STG_SIZE

For a Coupling Facility log stream, STG_SIZE defines the size of the staging data set to be allocated if STG_DUPLEX(YES) and DUPLEXMODE are specified. If STG_DUPLEX(YES) and DUPLEXMODE(UNCOND) are specified, then the data in the Coupling Facility log stream is always duplexed to the staging data set.

If STG_DUPLEX(YES) and DUPLEXMODE(COND) are specified, then the data in the Coupling Facility log stream is duplexed to the staging data set only if the CF becomes volatile or failure-dependent.

The size of the staging data set (STG_SIZE) must be large enough to hold as much data as the log stream storage in the Coupling Facility. Data is written to the staging data set in 4096 byte increments, regardless of the buffer size.

Attention: When staging data sets are used with a Coupling Facility log stream, STG_SIZE must be specified large enough to hold *at least* as much data as the CF log stream. If you do not specify any value for STG_SIZE, System Logger allocates a staging data set whose default size is the size of the corresponding structure in the Coupling Facility.

The structure storage is divided equally among the active log streams. If the number of log streams connected to the structure will vary, the STG_SIZE value should be chosen based on the least number of log streams that will normally be connected.

Offload processing is triggered based on the smaller capacity of either the structure log stream storage or the staging data set.

STG_DUPLEX

STG_DUPLEX(YES) with DUPLEXMODE(COND) means that if the CF becomes volatile, or resides in the same failure domain as the System Logger system, the log stream data is

duplexed to the staging data set; otherwise, it is duplexed to buffers in the System Logger dataspace.

A CF is in the same failure domain when the Coupling Facility LPAR and the LPAR running this z/OS reside in the same physical hardware box, central processing complex (CPC). Duplexing to the staging data set means the cost of an I/O will be incurred for each write.

DASD-only log streams definitions

Example 4-5 shows definitions for the RRS log streams when defined as DASD-only log streams.

Example 4-5 DASD-only log stream definitions in the System Logger policy

```
/DEFSTREXCPGM=IXCMIAPU
//SYSPRINT DD SYSOUT=A
//SYSINDD *
  DATA TYPE(LOGR)

  DEFINE LOGSTREAM NAME(ATR.PLEX1.MAIN.UR)
    LOWOFFLOAD(60)
    HIGHOFFLOAD(80)
    DASDONLY(YES)
    HLQ(TEST)
    LS_SIZE(1024)
    LS_DATACLAS(VSAMLS)
    STG_SIZE(1024)

  DEFINE LOGSTREAM NAME(ATR.PLEX1.DELAYED.UR)
    LOWOFFLOAD(60)
    HIGHOFFLOAD(80)
    DASDONLY(YES)
    HLQ(TEST)
    LS_SIZE(960)
    LS_DATACLAS(VSAMLS)
    STG_SIZE(960)

  DEFINE LOGSTREAM NAME(ATR.PLEX1.ARCHIVE)
    LOWOFFLOAD(0)
    HIGHOFFLOAD(80)
    DASDONLY(YES)
    HLQ(TEST)
    LS_SIZE(960)
    LS_DATACLAS(VSAMLS)
    AUTODELETE(YES)
    RETPD(2)
    STG_SIZE(2000)

  DEFINE LOGSTREAM NAME(ATR.PLEX1.RM.DATA)
    LOWOFFLOAD(60)
    HIGHOFFLOAD(80)
    DASDONLY(YES)
    HLQ(TEST)
    LS_SIZE(192)
    LS_DATACLAS(VSAMLS)
    STG_SIZE(192)

  DEFINE LOGSTREAM NAME(ATR.PLEX1.RESTART)
    LOWOFFLOAD(60)
    HIGHOFFLOAD(80)
```

DASDONLY(YES)
HLQ(TEST)
LS_SIZE(960)
LS_DATACLAS(VSAMLS)
STG_SIZE(960)

DASDONLY

This parameter specifies whether the log stream being defined is a Coupling Facility or a DASD-only log stream. If you specify DASDONLY(NO), which is the default, the log stream is defined as a Coupling Facility log stream.

If you specify DASDONLY(YES), then the log stream is defined as a DASD-only log stream and does not use the Coupling Facility for log data. With DASDONLY(NO), you can also specify STG_DUPLEX, DUPLEXMODE, and LOGGERDUPLEX parameters to select a method of duplexing for a Coupling Facility log stream.

HLQ

This parameter specifies the up to 8-byte high-level qualifier for both the log stream data set name and the staging data set name. HLQ must be 8 alphanumeric or national (\$, #, or @) characters, padded on the right with blanks, if necessary. The first character must be an alphabetic or national character.

MAXBUFSIZE

MAXBUFSIZE may be specified for a DASDONLY log stream, and it defines the largest block that can be written to the log stream. The default value is 65532 and should be used in a RRS environment.

STG_DUPLEX

STG_DUPLEX(YES) with DUPLEXMODE(COND) are *not* applicable to DASDONLY log streams.

Note: For further details on RRS logging environment, refer to Chapter 6 in *System Programmer's Guide to: z/OS System Logger*, SG24-6898.

4.3 Define the RRS infrastructure

In these sections we describe the tasks required to complete the RRS setup on your z/OS image.

4.3.1 WLM definitions

Use the Workload Manager (WLM) policy to control the RRS priority. The RRS priority needs to be equal to or higher than the dispatching priority of its resource managers. You can use the SYSSTC service class for the RRS address space to achieve a higher dispatching priority.

4.3.2 RRS subsystem definitions

Update the IEFSSNxx PARMLIB member to include the following statement:

```
SUBSYS SUBNAME(RRS)
```

Place this statement after the statement that defines the primary subsystem.

4.3.3 Define RRS procedure

ATTRRS is the name of the cataloged procedure that IBM supplies in SYS1.SAMPLIB. Copy SYS1.SAMPLIB(ATTRRS) to SYS1.PROCLIB(RRS). The member name RRS specified here can be replaced with any other member name, as long as it matches the subsystem name specified in the IEFSSNxx PARMLIB member.

4.3.4 RRS automation

If RRS fails, it can use automatic restart management to restart itself in a different address space on the same system. RRS, however, does not restart itself following a SETRRS CANCEL command. To stop RRS and cause it to restart automatically, use the FORCE command with ARM and ARMRESTART.

4.3.5 Define RRS panels to ISPF

RRS provides ISPF panels to allow an installation to work with RRS. The panels provide a way for you to troubleshoot resource recovery problems. Before you can use the panels, however, you must set up access authorization, allocate the libraries containing the panels, and add the RRS application to the ISPF primary option menu.

We recommend that you do set up the RRS panels, because you may encounter failure scenarios in which you would need to use the panel information to clean up outstanding transactions. There is no other mechanism for determining the state of the various resource managers. If you have a problem running RRS, you will need to use the RRS panels to help identify and fix the trouble in your sysplex.

To install the panels, follow these steps:

1. Update your logon procedure. Be sure the following libraries, where the RRS panels are stored, are in your concatenations.

(**Note:** If IPCS or WLM is installed in your logon procedure, then you get RRS along with them.)

- a. In your SYSPROC concatenation:

```
//          DD DSN=SYS1.SBLSCLIO,DISP=SHR
```

- b. In your ISPLIB concatenation:

```
//          DD DSN=SYS1.SBLSMSG0,DISP=SHR
```

- c. In your ISPLIB concatenation:

```
//          DD DSN=SYS1.SBLSPLN0,DISP=SHR
```

- d. In your ISPTLIB concatenation:

```
//          DD DSN=SYS1.SBLSTBL0,DISP=SHR
```

- e. In your ISPSLIB concatenation:

```
//          DD DSN=SYS1.SBLSKELO,DISP=SHR
```

2. Add the following lines to your primary options menu, member ISR@PRIM within your ISPLIB concatenation, in your logon procedure:

- a. Add to your menu options definitions:

```
%RRS +-%Resource Recovery Svcs Panels
```

- b. Add to your application list within the ISR@PRIM menu definition:

```
RRS, 'PANEL(ATRFPCMN) NEWAPPL(RRSP) '
```

4.3.6 Define RRS SAF authorization

In your installation, you can configure RRS to allow a user to manage all the RRS images in the sysplex from a single image. Access to RRS system management functions is controlled by the following RACF® resource.

To control RRS access across a sysplex, RRS uses the MVSADMIN.RRS.COMMANDS.*gname.sysname* resource in the FACILITY class, where *gname* is the logging group name and *sysname* is the system name.

If you are running RRS on a single system, RRS can use either the MVSADMIN.RRS.COMMANDS.*gname.sysname* resource or the MVSADMIN.RRS.COMMANDS resource in the FACILITY class to control access to RRS system management functions on the current system.

4.3.7 Define RRS component trace

We recommend running all the systems with the following CTRACE options in effect all the times:

```
TRACEOPTS
ON
BUFSIZE(64M)
OPTIONS('EVENTS(URSERVS,LOGGING,CONTEXT,EXITS,STATECHG,RRSAPI,RESTART)')
```

Use the **D TRACE,COMP=SYSRRS** command to see the current setting and use the **TRACE CT,ON,COMP=SYSRRS,PARM=CTIRRSxx** command to alter the settings once the CTIRRSxx parmlib member has been updated. RRS does not have to be recycled to enable RRS CTRACE.



RRS operations

In this chapter, we discuss RRS operations such as stopping and starting RRS. We also describe management of the RRS log streams.

This chapter covers the following topics:

- ▶ “Stopping RRS” on page 48
- ▶ “Stopping RRS” on page 48
- ▶ “Using RRS panels” on page 48

5.1 Starting RRS

You start RRS by starting the RRS address space. Normally you start RRS at IPL, and it should stay active on a system as long as resource managers that require RRS are running. Start RRS from the COMMNDxx member of PARMLIB (not from IEACMDxx, because other services that RRS depends on have not been started yet).

If you start RRS after JES is up, then you must start it with SUB=MSTR. On our system, we start RRS with the command **S RRS**. Example 5-1 shows an example of starting RRS.

You can only start one instance of RRS on a system at a time.

Example 5-1 RRS startup syslog sample without ARCHIVE log stream

```
IEF695I START RRS      WITH JOBNAME RRS      IS ASSIGNED TO USER STC
      , GROUP SYS1
ATR221I RRS IS JOINING RRS GROUP #@$#PLEX ON SYSTEM #@$2
IXL014I IXLCONN REQUEST FOR STRUCTURE RRS_RMDATA_1
WAS SUCCESSFUL.  JOBNAME: IXGLOGR ASID: 0014
CONNECTOR NAME: IXGLOGR_#@$2 CFNAME: FACILO3
IXL014I IXLCONN REQUEST FOR STRUCTURE RRS_MAINUR_1
WAS SUCCESSFUL.  JOBNAME: IXGLOGR ASID: 0014
CONNECTOR NAME: IXGLOGR_#@$2 CFNAME: FACILO3
IXL014I IXLCONN REQUEST FOR STRUCTURE RRS_DELAYEDUR_1
WAS SUCCESSFUL.  JOBNAME: IXGLOGR ASID: 0014
CONNECTOR NAME: IXGLOGR_#@$2 CFNAME: FACILO3
IXL014I IXLCONN REQUEST FOR STRUCTURE RRS_RESTART_1
WAS SUCCESSFUL.  JOBNAME: IXGLOGR ASID: 0014
CONNECTOR NAME: IXGLOGR_#@$2 CFNAME: FACILO3
IXG231I IXGCONN REQUEST=CONNECT TO LOG STREAM ATR.#@$#PLEX.ARCHIVE DID
NOT SUCCEED FOR JOB RRS.  RETURN CODE: 00000008 REASON CODE: 0000080B
DIAG1: 00000008 DIAG2: 0000F801 DIAG3: 05030004 DIAG4: 05020010
ATR132I RRS LOGSTREAM CONNECT HAS FAILED FOR
OPTIONAL LOGSTREAM ATR.#@$#PLEX.ARCHIVE.
RC=00000008, RSN=00000000
ASA2011I RRS INITIALIZATION COMPLETE. COMPONENT ID=SCRRS
```

On startup RRS connects to its log streams, and any failure to connect to any log stream except the ARCHIVE log stream results in an RRS abend. Example 5-1 shows what happens when the ARCHIVE log stream is not available: RRS issues ATR132I message and completes its initialization.

Example 5-2 shows the result of the command, which displays all RRS log streams.

Example 5-2 RRS log streams

```
D  LOGGER,LOGSTREAM,LSN=ATR.*
IXG601I  20.10.29  LOGGER DISPLAY
INVENTORY INFORMATION BY LOGSTREAM
LOGSTREAM          STRUCTURE          #CONN  STATUS
-----
ATR.#@$#PLEX.DELAYED.UR  RRS_DELAYEDUR_1  000003  IN USE
  SYSNAME: #@$1
    DUPLEXING: STAGING DATA SET
  SYSNAME: #@$3
    DUPLEXING: STAGING DATA SET
  SYSNAME: #@$2
    DUPLEXING: STAGING DATA SET
ATR.#@$#PLEX.MAIN.UR    RRS_MAINUR_1    000003  IN USE
```

```

SYSNAME: #@$1
  DUPLEXING: STAGING DATA SET
SYSNAME: #@$3
  DUPLEXING: STAGING DATA SET
SYSNAME: #@$2
  DUPLEXING: STAGING DATA SET
ATR.#@$#PLEX.RESTART      RRS_RESTART_1      000003 IN USE
SYSNAME: #@$1
  DUPLEXING: STAGING DATA SET
SYSNAME: #@$3
  DUPLEXING: STAGING DATA SET
SYSNAME: #@$2
  DUPLEXING: STAGING DATA SET
ATR.#@$#PLEX.RM.DATA      RRS_RMDATA_1      000003 IN USE
SYSNAME: #@$1
  DUPLEXING: STAGING DATA SET
SYSNAME: #@$3
  DUPLEXING: STAGING DATA SET
SYSNAME: #@$2
  DUPLEXING: STAGING DATA SET

```

5.1.1 RRS warm start

There is no option that can be passed to RRS to tell it to perform a warm or cold start. Instead, RRS performs a warm start once it finds data in the RM.DATA log stream.

Note that when RRS successfully warm starts (if data is available in the RM.DATA log stream), all RRS log streams (with the exception of the ARCHIVE log stream) should be intact in order for RRS to access data about incomplete transactions.

The first instance of RRS to start (within a logging group) performs log stream recovery.

After a warm start occurs, RRS is available to resource managers. This is the normal mode of operation in a production system.

5.1.2 RRS cold start

On startup, if RRS finds an *empty* RM.DATA log stream, then it cold starts. RRS flushes any log data found in the MAIN.UR and DELAYED.UR log streams to the ARCHIVE log, if it exists.

An RRS cold start applies to the entire RRS logging group. You cannot cold start RRS on just one system in a sysplex that is a part of a logging group. This is because the log streams are shared across all systems in the sysplex in that logging group.

After an RRS cold start, there is no data available to RRS to complete any work that was in progress before RRS was cold started. RRS can be cold started by stopping all RRS instances in the logging group, and deleting and redefining the RM.DATA log stream using the IXCMIAPU utility. RRS should only be deliberately cold started in very controlled circumstances; make sure of the following:

- ▶ All resource managers that require RRS are stopped on all systems that are a part of the RRS logging group to be cold started. Use the RRS ISPF panels to check on resource manager status.
- ▶ Using the RRS ISPF panels, check that no incomplete URs exist for any resource manager.

5.2 Stopping RRS

Use the **SETRRS CANCEL** operator command to stop RRS. It is good practice to issue this as a part of your system shutdown, because it results in a cleaner system recovery. It should also be done before doing a sysplex shutdown.

If RRS fails to stop as a result of the **SETRRS CANCEL** command, then you can issue a **FORCE RRS,ARM** operator command.

Note: Never stop RRS on a system while there are active resource managers using RRS services. The effects of this depends on the resource manager; however, they range from disruption of work, to complete failure of the resource manager. We discuss the effects of RRS being made unavailable on particular resource managers in Part 3, “RRS exploiters” on page 71, where we deal with each resource manager.

5.3 Using RRS panels

In this section, we provide detailed examples using RRS panels to view data from a particular workload. Figure 5-1 shows the RRS primary options panel. The first option selected is option 2, Display/Update RRS-related Resource Manager information. We can see the RRS panel content when running our sample workload, and view the meaning of the content.

```
Option ==>                                RRS
Select an option and press ENTER:

1 Browse an RRS log stream
2 Display/Update RRS related Resource Manager information
3 Display/Update RRS Unit of Recovery information
4 Display/Update RRS related Work Manager information
5 Display/Update RRS UR selection criteria profiles
6 Display RRS-related system information
```

Figure 5-1 RRS primary options panel

If you select option 2, you get the RRS Resource Manager Information panel, which shows the state of each registered resource manager. As shown in Figure 5-2 on page 49, each APPC LU is its own resource manager:

- ▶ DB2 has two resource managers, one for the RRS/AF connection (DSN.RRSATF.IBM.RDF0), and one for the WLM managed stored procedures (DSN.RRSPAS.IBM.RDF0).
- ▶ IMS has only one resource manager (IMS.IMR1____V061.STL.SANJOSE.IBM). DCE/AS has one resource manager per server address space.

Figure 5-2 also shows that there are two DCE/AS servers, MVSSYS1IMSSYS1SERVER1 and MVSSYS1IMSSYS1SERVER2. The proper state for an active resource manager is RUN. If the resource manager is not active, its state is RESET. If the state is UNSET, it means an error occurred and problem determination documentation needs to be gathered.

The resource manager appears in the list the first time it registers with RRS, and it never goes away until RRS is cold started. Therefore, if a resource manager is moved to another system,

that resource manager appears in the list for both systems: one in RESET state, and the other (if active) in RUN state.

RRS Resource Manager List				
Command ==>				
Commands: v-View Details u-View URs r-Remove Interest				
S	RM Name	State	System	Logging Group
	MVSSYS1IMSSYS1SERVER2	Run	#@\$2	#@\$#PLEX
	MVSSYS1IMSSYS1SERVER1	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0IMS1.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0IMS4.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0IMS3.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0IMS2.IBM	Run	#@\$2	#@\$#PLEX
	IMS.IMR1_V061.STL.SANJOSE.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0ASCHB.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0LUA07.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0LUA06.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0ASCHA.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0LUP07.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0LUA05.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0LUA04.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0LUP06.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0LUP05.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0LUP04.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0LUA03.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0LUGR.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0LUP03.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0LUGR1.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0LUGR0.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0PROT1.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0PROT2.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0PROT3.IBM	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0PROT4.IBM	Run	#@\$2	#@\$#PLEX
	DSN.RRSATF.IBM.RDF0	Run	#@\$2	#@\$#PLEX
	DSN.RRSPAS.IBM.RDF0	Run	#@\$2	#@\$#PLEX
	ATB.USIBMT6.MF0ASCHG.IBM	Run	#@\$2	#@\$#PLEX

Figure 5-2 RRS Resource Manger List showing state, system, logging group

If you select option 3 from the RRS primary menu panel, the RRS Unit of Recovery Selection panel is displayed; see Figure 5-3 on page 50. The RRS Unit of Recovery Selection panel allows users to select the units of recovery (URs) that they wish to see. After you press Enter, if there are no URs, then message ATR060I is displayed indicating that condition.

```

RRS Unit of Recovery Selection
  Command ==>
  ATR060I No information matches the selection criteria
  Commands: save-Save Profile  get-Get Profile  ENTER-Query

  Profile Name  . .

  UR and Work Identifier Criteria=====
  URID pattern . .

  SURID Pattern

  LUWID pattern (netid.luname,instnum,segnum)

  TID  . . . . .      (from 1 to 4294967295 in decimal)
  Low TID . .      High TID . .

  GTID Pattern
  00-0F
  10-1F
  20-27

  Format ID      (from 1 to 4294967295 in decimal)

  GTRID Pattern
  00-0F
  10-1F
  20-2F

```

Figure 5-3 RRS unit of recovery selection

If there are units of recovery to display, then the Unit of Recovery List panel is displayed; see Figure 5-4 on page 51.

RRS Unit of Recovery List				Row 1 to 16 of 41
Command ==>				Scroll ==> PAGE
Commands: v-View Details c-Commit b-Backout r-Remove Interest				
S	UR Identifier	System State	Logging Group Type	Comments
	B0DE5BB27E96F8A00000029301030000	#@\$2	#@\$#PLEX	
		InFlight	Unpr	
	B0DE5BB27E96FB380000029401030000	#@\$2	#@\$#PLEX	
		InFlight	Unpr	
	B0DE5BB27E96F6080000029201030000	#@\$2	#@\$#PLEX	
		InFlight	Unpr	
	B0DE5BB27E96E6780000028C01030000	#@\$2	#@\$#PLEX	
		InFlight	Unpr	
	B0DE5BB27E96E9100000028D01030000	#@\$2	#@\$#PLEX	
		InFlight	Unpr	
	B0DE5BB27E970D600000029B01030000	#@\$2	#@\$#PLEX	
		InFlight	Unpr	
	B0DE5BB27E9712900000029D01030000	#@\$2	#@\$#PLEX	
		InFlight	Unpr	
	B0DE5BB27E9717C00000029F01030000	#@\$2	#@\$#PLEX	
		InFlight	Unpr	
	B0DE5BB17E96BF900000027D01030000	#@\$2	#@\$#PLEX	
		InFlight	Unpr	
	B0DE5BB17E96D6E80000028601030000	#@\$2	#@\$#PLEX	
		InFlight	Unpr	
	B0DE5BB27E970300000029701030000	#@\$2	#@\$#PLEX	
		InFlight	Unpr	
	B0DE5BB27E96EBA80000028E01030000	#@\$2	#@\$#PLEX	
		InFlight	Unpr	
	B0DE5BB17E96D4500000028501030000	#@\$2	#@\$#PLEX	
		InFlight	Unpr	
	B0DE5BB27E96F3700000029101030000	#@\$2	#@\$#PLEX	
		InFlight	Unpr	
	B0DE5BB17E96BA600000027B01030000	#@\$2	#@\$#PLEX	
		InFlight	Unpr	

Figure 5-4 Unit of Recovery List

For most processing, URs are fleeting and the user has difficulty displaying them. However, the RRS Unit of Recovery List, showing 16 of 41 possible URs, is displayed after processing one and only one WLM-managed DB2 stored procedure. These URs are related to the NUMTCB parameter for the WLM-managed stored procedure address space.

When the stored procedure is invoked, WLM detects that a new address space is needed. The address space is created with NUMTCB plus one TCB. DB2 expresses an unprotected interest in each TCB. A stored procedure is assigned to a TCB. When a commit is executed, DB2 determines if it is the only interested resource manager. If it is, then no further action is taken, from an RRS perspective.

However, if there *is* another interested resource manager, then DB2 converts its interest into a protected interest, and assumes that the role of the SDSRM and RRS is called using the appropriate services for the presumed abort protocol. After the unit of recovery is complete, DB2 expresses an unprotected interest in the TCB once again.

If you place a v on the command line for one of the units of recovery listed in Figure 5-4, you can obtain information about all the resource managers that expressed interest in a unit of recovery. Figure 5-5 on page 52 shows the details of the unit of recovery.

```

RRS Unit of Recovery Details
Command ==>
Row 1 to 1 of 1
Scroll ==> PAGE

Commands r-Remove Interest v-View URI Details

UR identifier : B0DE5BB27E96F8A00000029301030000
Create time : 2003/08/07 18:58:01.984113      Comments :
UR state : InFlight      UR type : Unpr ASID : 0000
System : #@$2      Logging Group : #@$#PLEX
SURID : N/A
Work Manager Name :
    Display Work IDs      / Display IDs formatted
    Luwid . : Not Present
    Eid . . : Not Present
    Xid . . : Not Present
Expressions of Interest:
S   RM Name      Type Role
    DSN.RRSPAS.IBM.RDF0      Unpr Participant

```

Figure 5-5 Unit of Recovery Details panel

The Unit of Recovery Details panel displays the exits for which a resource manager can be called, as well as the current status of each exit. In this case there was no sync point requested, so no exits have been called yet, and the status shows as Uncalled; see Figure 5-6.

```

RRS URI Details
Command ==>

UR identifier : B0DE5BB27E96F8A00000029301030000
URI token . . : 7E872B4000000465000000000000000000
RM name . . . : DSN.RRSPAS.IBM.RDF0
Type . . . . : Unpr      Status . : ACTIVE
Role . . . . : Participant State . : InFlight
SURID : N/A

Exit/State      Status      Duration
BACKOUT . . . : Uncalled
COMPLETION . . : Uncalled
COMMIT . . . . : Uncalled
DSE/IN DOUBT . : Uncalled
End_UR . . . . : Uncalled
EXIT_FAILED . . : Uncalled
ONLY_AGENT . . : Uncalled
PREPARE . . . . : Uncalled
STATE_CHECK . . : Uncalled

```

Figure 5-6 RRS URI Details

When an exit is called, its status changes to CALLED. When an exit returns, the status reflects the return code that the exit passed back to RRS. These return codes are documented in *z/OS MVS Programming: Resource Recovery*, SA22-7616.

If you want to browse an RRS log stream, return to the RRS primary options menu and select option **1, Browse an RRS log stream menu**. This will bring you to the RRS logstream Browse Selection panel shown in Figure 5-7 on page 53.

From this panel you may choose which log stream to browse. Notice that the group name defaults to the name of our sysplex. However, since the group name for our test is APPC, we must always overstrike this field whenever we access this panel. This ensures that we select the correct set of log streams.

```

                                RRS logstream Browse Selection
Command ==>

Provide selection criteria and press Enter:

Select a logstream to view:                Level of report detail:
2  1. RRS Archive log                      2  1. Summary
    2. RRS Unit of Recovery State logs      2. Detailed
    3. RRS Restart log
    4. RRS Resource Manager Data log

RRS Group Name . . . UTCPLXHD Default is the sysplex name
Output dataset  . . ATR.REPORT

Optional filtering:
  Entries from . . . . . local date in yyyy/mm/dd format
                        local time in hh:mm:ss format
  through  . . . . . local date in yyyy/mm/dd format
                        local time in hh:mm:ss format
  UR identifier . . . . . (Options 1,2,3)
  RM name   . . . . . (Option 4)
  SURID     . . . . . (Options 1,2,3)

```

Figure 5-7 RRS logstream Browse Selection

For any of the log streams you choose to browse, RRS places the data in a file called userid.ATR.REPORT. This file may be preallocated and the data set name may be changed, or if the data set does not exist, RRS will dynamically allocate it.

Note: If you have an active ARCHIVE log stream that has extensive activity, use the filtering options to reduce the size of the report, as you can easily exceed the size of the data set allocation and get a B37 abend.

If you are searching for a particular UR in the ARCHIVE log stream, we recommend that you restrict the time frames of both Entries from and through, because RRS uses the UR identifier and the from time to intelligently restrict where to start searching in the log stream. However, RRS makes no assumptions about *when* all archive records are found. Thus, if a record is requested in the middle of the log stream, RRS searches to the end of the log stream.

The RRS/MVS Log Stream Browse Detail Report panel, shown in Figure 5-8, is displayed when the RRS Unit of Recovery State logs are selected with the Detailed option. Notice that this display shows both the MAIN.UR and DELAYED.UR log stream content. Since our workload never generates records that go to the MAIN.UR log stream, the display accurately shows that it is empty.

```

BROWSE      JOHNETZ.ATR.REPORT                               Line 00000000 Col 001 080
Command ==>                                         Scroll ==> CSR
***** Top of Data *****
RRS/MVS LOG STREAM BROWSE DETAIL REPORT

READING ATR.APPC.MAIN.UR          logstream
ATR053I ATR.APPC.MAIN.UR          IS EMPTY,
RC=00000008 RSN=00000846

READING ATR.APPC.DELAYED.UR       logstream

N9B      2003/08/07 14:43:12.440187 BLOCKID=0000000040C82811
URID=B0DE58617E9CC0000000171401020000 LOGSTREAM=ATR.APPC.DELAYED.UR
STATE=InCommit EXITFLAGS=00800000 FLAGS=A0000000
LUWID=USIBMT6.MF3LUA07 586199B04C09 0001 TID=          GTID=

FORMATID=          (decimal)          (hexadecimal)
GTRID=

EQUAL=

RMNAME=ATB.USIBMT6.MF3LUA07.IBM          ROLE=Participant
CMITCODE=00000FFF BACKCODE=00000FFF PROTOCOL=PresumeNothing

FORMATID=          (decimal)          (hexadecimal)
GTRID=

EQUAL=

N9A      2003/08/07 14:43:12.676904 BLOCKID=0000000040C82DFD
URID=B0DE58617E9CE53000000103901010000 LOGSTREAM=ATR.APPC.DELAYED.UR
STATE=InDoubt EXITFLAGS=00000000 FLAGS=A0000000
LUWID=USIBMT6.MF2LUA07 586191565603 0001 TID=          GTID=

FORMATID=          (decimal)          (hexadecimal)
GTRID=

EQUAL=

RMNAME=ATB.USIBMT6.MF2IMS2.IBM          ROLE=DSRM
CMITCODE=00000000 BACKCODE=00000FFF PROTOCOL=PresumeNothing
RMNAME=IMS.IMR3_V061.STL.SANJOSE.IBM ROLE=Participant
CMITCODE=00000FFF BACKCODE=00000FFF PROTOCOL=PresumeAbort

```

Figure 5-8 RRS/MVS Log Stream Browse Detail Report

Figure 5-9 on page 55 displays the ARCHIVE log stream browse output panel.

```

BROWSE      JOHNBTZ.ATR.REPORT                                Write failed with abend
Command ==>                                         Scroll ==> CSR
***** Top of Data *****
RRS/MVS logstream BROWSE DETAIL REPORT

READING ATR.APFC.ARCHIVE          logstream

N9A      2003/08/07 00:11:37.787861 BLOCKID=0000000000000001
URID=B0DD95907E9A2C780000005601010000 JOBNAME=SP303 USERID=*
SYNCPOINT=Commit RETURN CODE=00000000
START=2003/08/07 04:11:37.772092 COMPLETE=2003/08/07 04:11:37.787469
EXITFLAGS=00800000
LUWID=USIBMT6.MF1LUA07 958E44394D09 0001 TID=          GTID=

FORMATID=          (decimal)          (hexadecimal)
GTRID=

BQUAL=
RMNAME=ATB.USIBMT6.MF2PROT3.IBM          ROLE=DSRM
  FLAGS=100E0000 PROTOCOL=PresumeNothing
  StateCheck EXIT RC=00000000
  Prepare EXIT RC=00000000
  DistSp EXIT RC=00000000
  Commit EXIT RC=00000000
  Backout EXIT RC=Uncalled
  EndUr EXIT RC=00000000
  ExitFailed EXIT RC=Uncalled
  Completion EXIT RC=00000000
  OnlyAgent EXIT RC=Uncalled
RMNAME=DSN.RRSATF.IBM.RDF2          ROLE=Participant
  FLAGS=10000000 PROTOCOL=PresumeNothing
  StateCheck EXIT RC=Uncalled
  Prepare EXIT RC=00000010
  DistSp EXIT RC=Uncalled
  Commit EXIT RC=Uncalled
  Backout EXIT RC=Uncalled
  EndUr EXIT RC=Uncalled
  ExitFailed EXIT RC=Uncalled
  Completion EXIT RC=Uncalled
  OnlyAgent EXIT RC=Uncalled

```

Figure 5-9 ARCHIVE log stream browse output

In the upper-right corner of the screen it shows that an abend code was received. This happened because no filtering was used, and the data set ran out of space. Also note that the selected URs, which were placed in this document, show the completed states of records and the return codes from all the exits called.

We use the ARCHIVE log stream to determine the RRS view of the outcome of records when we execute recovery scenarios. For example, DB2 has a unit of recovery that is outstanding, but RRS does not know about it, so you cannot display the UR to get a time stamp. As previously stated, filtering using dates is extremely important in order to get a timely response when archive logs are large.

Note: We used a non-supported interface to quickly determine the time that a unit of recovery was created. The first eight digits of the URID are a part of a time stamp. When this is fed into a time-of-day conversion routine, the user can quickly determine the approximate time of the creation of the unit of recovery. Again, this is not a supported interface and could change in the future.

Figure 5-10 shows the last panel that we will discuss. We chose to browse the Resource Manager Data log stream. This log stream gives the details for all of the resource managers that the RRS on this system knows about, since the last cold start.

```

Menu Utilities Compilers Help

  BROWSE      IBMUSER.ATR.REPORT                      Line 00000000 Col 001 080
  Command ==>                                         Scroll ==> CSR
***** Top of Data *****
RRS/MVS logstream BROWSE DETAIL REPORT

READING ATR.APPC.RM.DATA          logstream

N9B      2003/08/07 16:17:43.231213 BLOCKID=000000000009FC91
RESOURCE MANAGER=IMS.IMR4      V061.STL.SANJOSE.IBM LOGGING SYSTEM=N9B
RESOURCE MANAGER MAY RESTART ON ANY SYSTEM
RESOURCE MANAGER WAS LAST ACTIVE WITH RRS ON SYSTEM N9B
LOG NAME IS IMS.IMR4      .OLDS.IBM
M9A      2003/08/07 16:17:22.448002 BLOCKID=000000000009BB99
RESOURCE MANAGER=DSN.RRSPAS.IBM.RDF1          LOGGING SYSTEM=M9A
RESOURCE MANAGER MUST RESTART ON SYSTEM M9B
RESOURCE MANAGER WAS LAST ACTIVE WITH RRS ON SYSTEM M9B
LOG NAME IS DSN3.DB2.RRSP.IBM.RDF1

```

Figure 5-10 RRS/MVS logstream BROWSE DETAIL REPORT

The first entry shows that IMS was brought down cleanly, which means that there were no outstanding protected units of recovery. In this case RRS allows the IMS subsystem to be restarted anywhere.

The second entry needs some interpretation: If DB2 were down, then this state implies that there were outstanding protected units of recovery that need resolving; therefore, when DB2 is brought back up, it must be brought back up on the same system. However, in our case, we know that DB2 was up and running well, so RRS assumes that a resource manager must be restarted on the same system—and only changes that determination when the resource manager terminates. If there are no protected units of recovery, then RRS changes the state to allow restart anywhere.

Attention: In z/OS 1.6, RRS removes the restriction that a restart manager must restart on the same system if there are outstanding URs for that resource manager.

RRS performance and availability

A major factor affecting RRS performance and availability is directly related to the RRS logging environment. For this reason, proper tuning and placement of the RRS log streams become important elements in the RRS configuration. In this chapter, we discuss RRS performance and tuning in relation to Stem Logger definitions. We also describe how to monitor performance of the log streams.

This chapter covers the following topics:

- ▶ “Availability considerations for RRS log streams” on page 58
- ▶ “Performance considerations of RRS log streams” on page 59

6.1 Availability considerations for RRS log streams

For Coupling Facility log streams, System Logger always keeps two sets of the log data: one in the Coupling Facility, and one in each image's dataspace. If the Coupling Facility fails, System Logger is capable of rebuilding the merged log data with the data coming from each image dataspace.

If your installation is sensitive to a double failure (for example, in which you could lose both the Coupling Facility and the z/OS image at the same time), then you can configure a further System Logger step to prevent any risk of losing data: use DASD staging data sets. In such a configuration, System Logger maintains the local copy of log data on DASD instead of the dataspace—and if an error occurs on the Coupling Facility and the z/OS image, the DASD backup can serve as a copy of valid data when System Logger restarts.

Using the STG_DUPLEX and DUPLEXMODE parameters in the LOGR policy, you can request how you want System Logger to duplex Coupling Facility-resident log data for a log stream. Duplexing is done on a connection basis, depending on whether the system contains a single point of failure and is therefore vulnerable to loss of data.

A failure-dependent environment is an environment where one failure can result in simultaneously losing both the Coupling Facility structure for a log stream and the local storage buffer copy of the data on the system connected to the Coupling Facility.

You can specify STG_DUPLEX(YES) DUPLEXMODE(COND) when you cannot afford any data loss. This is a flexible way to protect against a single point of failure. With DUPLEXMODE(COND), System Logger will only duplex log data to staging data sets when needed. Connections that do not have a single point of failure can exploit the performance benefits of backing up log data to the local storage buffer. If DUPLEXMODE(UNCOND) is specified, System Logger unconditionally duplexes Coupling Facility data log data to DASD staging data sets, regardless of whether there is a single point of failure in the configuration.

Since duplexing the logs might impact performance, and RRS can run effectively without duplexing, your installation must decide on the risk it can afford to take based on the following considerations:

- ▶ The RRS transaction state logs, MAIN.UR and DELAYED.UR, are large and frequently updated. Since RRS might be able to tolerate and recover from loss of data and gaps in its Unit of Recovery state logs, you may want to consider not duplexing these two logs in order to minimize the performance impact.

For these log streams, where duplexing is less desirable, you can either specify DUPLEXMODE(COND) and let the System Logger determine when to duplex the Coupling Facility log data, or you can specify STG_DUPLEX(NO) to disable duplexing.

- ▶ RRS cannot tolerate or recover from a gap in the RM.DATA log. Any loss of data, unresolved gap, or permanent error in this log forces an RRS cold start. For availability reasons, we recommend that you unconditionally duplex RM.DATA log. This log is small and infrequently updated, so its impact on performance is minimal.

By default, DASD-only log streams are hardened on staging data sets.

Note: For further details about this topic, refer to *System Programmer's Guide to z/OS System Logger*, SG24-6898.

6.2 Performance considerations of RRS log streams

Because RRS writes data to a log stream, the interim storage defined for log data begins to fill, eventually reaching or exceeding its high threshold. You can specify the high and low thresholds for each RRS log stream to define when System Logger begins and ends the process of offloading (or moving) log data to DASD log data sets. For information on how System Logger offloading works, refer to Chapter 9 in *z/OS MVS Setting Up a Sysplex*, SA22-7625.

Offloading works differently for CF and DASD-only log streams:

- ▶ For a CF log stream, offload processing moves data from CF to DASD offload data sets when the CF usage reaches the high threshold.
- ▶ For a DASD-only log stream, offload processing is triggered by the staging data set usage.

However, for either type of log stream, when a log stream reaches or exceeds its high threshold, System Logger begins offloading enough of the oldest log stream data to get to the low offload threshold point specified in the log stream definition. It is possible that a log stream might exceed the high threshold before offloading starts, because applications might keep writing data before System Logger can begin offload processing.

The log records in the MAIN.UR and the DELAYED.UR log streams are transient transactional data. They only last for the life of a transaction and are marked for deletion when the transaction completes. RRS also requires a relatively high amount of its transaction log data to reside in the Coupling Facility or local storage buffers for quick access. If the availability of the Coupling Facility space is not of concern to you, you can limit the offload activity by specifying a high threshold for offloading data to DASD. Therefore, for the MAIN.UR and DELAYED.UR logs, you can use a high threshold of 80% and a low threshold of 60%.

However, the log records in the RM.DATA log are small and permanent; never offload during normal processing. You can use a high threshold of 80% and a low threshold of 60% for both the RM.DATA log and the RESTART log. If the ARCHIVE log is defined, you can use a low threshold of 0%.

The following factors also affect the frequency and overhead of the offload processing.

Coupling Facility size

Finding the optimum size for the Coupling Facility structure is important. This helps in handling the offload processing rate and in keeping more log data available on the Coupling Facility. If the Coupling Facility space allocated for a log stream is 100% full, all write requests against this log stream will be rejected and retried until offloading can complete. You can use SMF type 88 records to monitor the offload activity, and use the RMF™ Coupling Facility Report to evaluate the current Coupling Facility configuration, making make adjustments as necessary.

Staging data set size

For a Coupling Facility log stream, the high offload threshold applies to both the Coupling Facility space and the staging data set size. System Logger begins offload processing if either one exceeds the high threshold.

If the staging data sets are too small for a log stream, the staging data set keeps filling up and System Logger offloads Coupling Facility data to DASD frequently. If the staging data sets are filled completely, System Logger is not able to log data until it can free up staging data set space. In general, the sizing guideline for staging data sets is to make them large enough to hold all the log data in the Coupling Facility.

For a DASD-only log stream, offloading of log data to DASD log data sets is always triggered by staging data sets usage. If the staging data sets are too small, System Logger offloads log data from local storage buffers to DASD log data sets frequently.

6.2.1 RRS performance monitoring

Performance data for the RRS log streams can be produced by a number of tools. Reports can also be analyzed, to see if adjustments need to be made to the System Logger definitions.

- ▶ SMF 88 records are produced by the System Logger
- ▶ SMF 70-78 records are produced by RMF or equivalent product

SMF type 88 records can be formatted by the IXGPRT1 tool. Check for occurrences of DASD SHIFT, STRUCTURE FULL, ENTRY FULL and OFFLOAD for the RRS log streams. The ARCHIVE log stream is the only RRS log stream that should experience OFFLOAD processing because it is continually written to by RRS; no records are ever deleted.

RMF can provide many reports on structure usage and performance, as well as many other aspects of system performance.

Note: For further details on how to monitor RRS log streams and how to interpret the tools report, refer to *System Programmer's Guide to z/OS System Logger*, SG24-6898.

RRS restart and recovery

In this chapter, we discuss RRS restart and resource manager restart and recovery. Although RRS restart and resource manager restart are two separate items, they must be considered together because in many instances RRS restart issues have a major effect on resource manager restart issues.

Examples of scenarios in which both RRS and RM restart need to be considered are:

- ▶ RRS may fail on an image and need to be restarted. Resource managers on that image may also need to be restarted.
- ▶ Resource managers may fail on an image and need to be restarted on that same image or on other images in the sysplex. RRS restart issues affect *where* those resource managers may restart.
- ▶ The whole image may fail and RRS recovery issues in the sysplex may affect where failed resource managers may restart.

This chapter covers the following topics:

- ▶ “RRS restart” on page 62
- ▶ “Resource manager restart” on page 63
- ▶ “Resource Manager restart restrictions” on page 63

7.1 RRS restart

As discussed in “RRS operations” on page 45, RRS can be stopped and restarted on a system, and perform a warm start or cold start, depending on the data it finds in its RM.DATA log stream.

7.1.1 RRS log takeover

In the event of a system failure or an extended RRS outage on a system, other instances of RRS on other images in the sysplex that are in the same RRS logging group can perform log takeover. See “RRS logging group name” on page 33 for a description of an RRS logging group.

If RRS on one systems fails, or if the whole image fails, the remaining RRS instances in the RRS logging group detect that an RRS instance has failed. This can be detected in two ways:

- ▶ Each RRS instance polls the logging group for failed RRS instances approximately every 30 seconds.
- ▶ When a resource manager enters restart processing with an RRS instance, RRS checks for any failed RRS instances in the logging group.

When an RRS instance detects a failed RRS instance, it moves outstanding URs associated with the failed instance into the RRS Restart logs. The URs that are moved are:

- ▶ All non-cascaded or local cascaded URs that were logged on the failed RRS instance.
- ▶ All multisystem cascaded URs where the coordinator UR was logged on the failed RRS instance. The coordinator UR in a multisystem cascaded UR is the root UR of the UR tree, that is, the UR with no parent.

The RRS logs are shared across all RRS instances in the logging group, thus the restart data is available to all RRS instances in the logging group. When a resource manager restarts on one of the other images, RRS on that image is able to retrieve the outstanding URs for that resource manager from the Restart log stream.

When an RRS instance performs Log Takeover, it issues the following message:

```
ATR222I LOG TAKEOVER FOR SYSTEM sysname HAS COMPLETED SUCCESSFULLY.
```

In our testing we found the RRS Log Takeover performed within seconds of an RRS failure, but this was in a fairly active sysplex with 16 images.

7.2 Resource manager restart

Resource manager restart and recovery is a complicated subject. Each resource manager, whether it is WebSphere Application Server, CICS, IMS, DB2 or so on, has its own particular restart issues and interacts with RRS differently. In Part 3, “RRS exploiters” on page 71, we discuss the major resource managers and examine their specific restart and recovery issues in relation to RRS.

Note: In this redbook, we limit our discussion on resource manager restart to issues specific to RRS. Most resource managers do extensive restart and recovery processing that may have nothing to do with RRS. For example, DB2 for z/OS supports a number of attach facilities, some of which use RRS, and some that do not. DB2 restart and recovery processing occurs with or without RRS. If RRS is unavailable at DB2 restart, DB2 may have to postpone recovery for certain URs that were being coordinated by RRS.

Here, we describe what resource managers in general do with RRS in a restart situation.

7.2.1 Resource manager startup sequence

When a resource manager restarts, there are a number of actions it performs in order to interact with RRS to get information about any URs that RRS had logged.

A generalized resource manager startup sequence is:

1. The resource manager registers itself using Registration Services.
2. The resource manager sets its exits for the exit managers. The two exit managers involved are Context Services and the Resource Recovery Service (RRS).
3. The resource manager uses the ATRIRLN service to retrieve the RRS logging group name and also to check if RRS has a record of the resource manager log name.
4. The resource manager uses ATRIBRS to begin restart processing with RRS.

7.2.2 Resource Manager restart restrictions

RRS imposes certain restart restrictions on resource managers if they fail while there are outstanding URs for that resource manager. These restrictions are z/OS release-dependent, so we point out which restrictions apply to which release of z/OS. In the following sections, we discuss issues for a single resource manager scenario and for multiple resource manager scenarios.

Single resource manager scenario

Prior to z/OS 1.6, if a resource manager fails, RRS determines whether there are any outstanding URs for that resource manager. If there are outstanding URs, then RRS marks that resource manager as having to restart on the same system. You must restart the resource manager on the same image in order for it to perform RRS restart processing.

If RRS also fails on that image, then RRS log takeover processing occurs and a resource manager may restart on any system in the RRS logging group. The practical implications of this are—unless you have a system failure, you are expected to restart a failed resource manager on the *same* image.

With z/OS 1.6 and above, RRS permits a resource manager to restart on any system that is a part of the RRS logging group, even if there are still URs outstanding for that resource

manager. However, this support does *not* mean that a resource manager can ignore group restart restrictions, as explained in the next section.

Attention: The pre-z/OS 1.6 restart restriction does not necessarily mean you cannot restart a resource manager on a different image. It means that if you *do* restart the resource manager on a different image, then RRS returns an error code to the resource manager when the resource manager attempts restart processing with RRS (by calling the ATRIBRS service). The implications of this depend on the particular resource manager.

Multiple resource manager scenario

By multiple resource manager, we mean a scenario where multiple resource managers have an interest in a UR and one or more resource managers fail. The same restrictions apply to this scenario as apply to the single resource manager scenario—but there are further restrictions because RRS treats multiple resource managers involved in a UR as a group.

In z/OS V1R2, RRS enabled “restart anywhere”, which allows its resource managers more flexibility as to where they are allowed to restart. RRS provided resource manager grouping relief by allowing resource managers to restart on any system in the same RRS logging group that supports multisystem cascaded transactions, provided RRS did not remain active on that system or the resource manager was not involved in any incomplete transactions with RRS.

Prior to z/OS 1.6, the baseline restart restriction still exists such that, if RRS remained active across resource manager failure and the resource manager had incomplete interests with RRS, then the resource manager needs to restart on the same system.

The constraints on the resource manager restart processing impose the following restrictions on the resource manager restart scenarios.

- In a peer-level recovery environment

After a system failure, if you attempt to restart the resource managers on an alternate system, perform transactional recovery, and then move the resource managers back to the system where they were last active for new work - if the resource manager fails to clean up its incomplete transactions or somehow gets involved in new work, then you would not be able to move the resource manager back to the system where it was last active.

- In a different peer-level recovery scenario

When a resource manager fails, its partner resource manager on an alternate system will take over the failed resource manager's workload, which includes resolving the failed transactions and starting new work. However, with the current resource manager restart restrictions, this type of takeover is not possible.

Table 7-1 lists the resource manager restart environments and RRS restrictions on where the resource managers can restart.

Table 7-1 Resource manager - restart restrictions

Environment for RM Restart	Restrictions
The resource manager has no protected interest in any incomplete unit of recovery (UR).	No restart restrictions. The resource manager can start on any system in the sysplex.
The resource manager has an incomplete protected interest in one or more URs. RRS has remained active on the system where the resource manager was last active.	The resource manager must restart on the same system. RRS fails any attempt by this resource manager to restart on a different system.

Environment for RM Restart	Restrictions
<p>The resource manager has an incomplete protected interest in one or more URs. RRS did not remain active on the system where the resource manager was last active, but RRS might have already restarted on the system where it failed.</p> <p>From the restart group, no resource managers are currently active with RRS.</p>	<p>The resource manager restarts on a system that supports Restart Anywhere: No restart restriction. The resource manager can restart on this system</p>
<p>The resource manager has an incomplete protected interest in one or more URs. RRS did not remain active on the system where the resource manager was last active, but RRS might have already restarted on the system where it failed.</p> <p>From the restart group, one or more resource managers are currently active with RRS.</p>	<p>The currently active resource manager(s) restarted on a system that supports Restart Anywhere: The resource manager can restart on any system that supports Restart Anywhere, but would be restricted from restarting on a system without this support.</p>

In z/OS V1R2, RRS provided some restart grouping relief. However, if RRS remains active across resource manager failure, the restart grouping still governs where the resource manager can restart. Table 7-2 lists z/OS releases and their restart restrictions.

Table 7-2 Restart restrictions based on software level

Release	Restart grouping	Restart restrictions
OS/390 R9-R10 z/OS V1R1	If the first RM in the RM group restarts on any of these systems, the restart grouping applies. This forces other RMs in the group to restart on this system.	RM has to restart on the same system if RRS remained active across RM failure and the RM had incomplete transactions.
Z/OS V1R2 - V1R5	If the first RM in the restart group restarts on any of these systems, there is no restart grouping if the rest of the RM group restarts on one of these systems.	RM has to restart on the same system if RRS remained active across RM failure and the RM had incomplete transactions.
z/OS V1R6 and above	If the first RM in the restart group restarts on any of these systems, there is no restart grouping if the rest of the RM group restarts on one of these systems. (same as z/OS 1.2).	RM is permitted to restart on any system, even if RRS had remained active across RRS failure and RM had incomplete transaction.

Even though RRS relaxed the resource manager restart grouping in z/OS V1R2, there may still be functional requirements for resource managers that have restart affinities to restart on the same system. For example, a WebSphere for z/OS server that connects to a CICS system using local attach requires that the CICS system be on the same z/OS image as itself. You can use ARM grouping or automation software to enforce related resource managers to restart as a group.

7.2.3 Example of resource manager restart within the same RRS logging group

In this section, we describe resource manager restarts within a sysplex, assuming all systems are in the same RRS logging group. This is the normal mode of operation within a sysplex.

In the following scenario, SYS1 and SYS2 are in the same sysplex, with RRS on both systems in the same logging group. Then SYS1 fails. RMA and RMB had common work being coordinated by RRS on SYS1.

If both SYS1 and SYS2 are running z/OS V1R2 (or above), both RMA and RMB can restart on SYS2, or RMA can restart on SYS2 and RMB can restart on SYS1 when it becomes active again. If RMA accepts new work on the recovery system, RRS will restrict it to have to restart on the new system.

If SYS2 is running z/OS V1R2 (or above) and SYS1 is running a down-level z/OS release, if RMA restarts first on SYS2, then RMB will have to restart on SYS2 also due to the incompatibility of the z/OS releases. If SYS2 is running a release lower than z/OS V1R2, then RMA restarting on SYS2 will force RMB to restart on SYS2 because they are in the same restart group.

Prior to z/OS 1.6, if a resource manager restarting on the peer system later attempts to move back to its original system SYS1, the resource manager cannot have any outstanding transactions; otherwise, RRS will prevent it from moving to a different system.

7.2.4 Example of resource manager restart outside the same RRS logging group

In this section we describe a resource manager restart on a system that is not a part of the same RRS logging group (for example, a DR hot backup system that is not a part of the production sysplex).

In this scenario, RRS on SYS1 and SYS2 are in different RRS logging groups. SYS1 and SYS2 may or may not be in the same sysplex. RMA and RMB are registered RRS resource managers.

RMA failed and is recovered from hot standby on SYS2. Registering as an RRS resource manager will not prevent the resource manager from restarting on a different system in the event of resource manager failure. RRS only enforces a resource manager having to restart on the same system if the resource manager has any outstanding transactions coordinated by RRS.

Prior to z/OS 1.6, if RMA was involved in any RRS-coordinated transactions at the time of its failure, typically RMA needs to restart on the same system. However, since RMA is restarting on a system that is not in the same logging group as its original system, RRS has no knowledge of RMA's old identity. RRS allows RMA to restart on SYSA.

Note: In this scenario, RRS is not be able to complete any outstanding transactions involving RMA that were in progress on SYS1 prior to RMA's failure.

7.2.5 Sample DB2/MQ restart scenario with RRS

This is a relatively simple scenario demonstrating a number of points about resource manager restart and showing how to check resource manager states using the RRS ISPF panels.

We ran a batch job on system SC53 that updated both DB2 and MQ in one UR. We cancelled the MQ subsystem while the UR was in-flight. We then attempted to restart the MQ subsystem on a different image (SC52).

Both resource managers are running on SC53. Example 7-1 shows a screenshot from the RRS ISPF panels.

Example 7-1 Resource Manager list from the RRS panel

RRS Resource Manager List

Row 1 to 11 of 11

Command ==>

Scroll ==> PAGE

Commands: v-View Details u-View URs r-Remove Interest

S	RM Name	State	System	Logging Group
	CSQ.RRSATF.IBM.MQCB	Run	SC53	WTSCPLX1
	CSQ.RRSATF.IBM.MQJ2	Run	SC53	WTSCPLX1
	CSQ.RRSATF.IBM.MQV1	Run	SC53	WTSCPLX1
	DSN.RRSATF.IBM.DB7A	Run	SC53	WTSCPLX1
	DSN.RRSATF.IBM.DB7J	Run	SC53	WTSCPLX1
	DSN.RRSATF.IBM.DB7L	Run	SC53	WTSCPLX1
	DSN.RRSATF.IBM.D7V1	Run	SC53	WTSCPLX1
	DSN.RRSPAS.IBM.DB7A	Run	SC53	WTSCPLX1
	DSN.RRSPAS.IBM.DB7J	Run	SC53	WTSCPLX1
	DSN.RRSPAS.IBM.DB7L	Run	SC53	WTSCPLX1
	DSN.RRSPAS.IBM.D7V1	Run	SC53	WTSCPLX1

Example 7-1 is a list of resource managers on system SC53. The MQ subsystem we are interested in is MQV1; the resource manager name for that subsystem is CSQ.RRSATF.IBM.MQV1. The DB2 subsystem we are interested in is D7V1, and the resource manager for the RRS attach facility for D7V1 is DSN.RRSATF.IBM.D7V1.

Both resource managers have a state of Run. This means that the Resource Manager is fully active with RRS.

We now run a batch job that updates MQV1 and D7V1 within the same UR. The program was written to pause for a number of minutes before issuing a commit so that we have time to display panels and so on.

We can see the UR in progress by going into the RRS panels, taking option 2, which is the Display/Update RRS-related Resource Manager information panel, and viewing all URs for that resource manager by selecting u against that resource manger. We looked at the UR associated with MQV1, as shown in Example 7-2.

Example 7-2 Unit of Recovery for MQV1 Resource Manager

RRS Unit of Recovery List

Row 1 to 5 of 5

Command ==>

Scroll ==> PAGE

Commands: v-View Details c-Commit b-Backout r-Remove Interest f-View UR Family

S	UR Identifier	System	Logging Group	
		State	Type	Comments
	BA6CEFFA7E627000000000001020000	SC53	WTSCPLX1	
		InFlight	Unpr	
	BA6CEFFC7E6273740000000001020000	SC53	WTSCPLX1	
		InFlight	Unpr	
	BA6CEFFE7E6276E80000000001020000	SC53	WTSCPLX1	
		InFlight	Unpr	
	BA6CF0047E627A5C0000000001020000	SC53	WTSCPLX1	

	InFlight	Unpr
<u>BA6D07707E627DD00000000001020000</u>	SC53	WTSCPLX1
	InFlight	Prot

In Example 7-2, we can see that there is only one instance of an protected conversation, which is the one we are interested in.

We selected that UR by using a v to see the UR details screen; refer to Example 7-3.

Example 7-3 Unit of Recovery Details information panel

```

RRS Unit of Recovery Details
Command ==>
Row 1 to 2 of 2
Scroll ==> PAGE

Commands r-Remove Interest v-View URI Details

UR identifier : BA6D07707E627DD00000000001020000
Create time : 2003/12/05 18:50:23.990574      Comments :
UR state : InFlight      UR type : Prot
System : SC53      Logging Group : WTSCPLX1
SURID : N/A
Work Manager Name : SC53.MURPHYAR.0026
  Display Work IDs      / Display IDs formatted
  Luwid . . : Not Present
  Eid . . : Not Present
  Xid . . : Not Present
Expressions of Interest:
S  RM Name      Type Role
   CSQ.RRSATF.IBM.MQV1      Prot Participant
   DSN.RRSATF.IBM.D7V1      Prot Participant

```

The UR details panel shows that the UR has two resource managers participating. Each has a role of Participant, which means that RRS acts as the syncpoint coordinator. The UR state is shown as InFlight, meaning this UR has not reached a commit or backout point yet. We now cancel the MQV1 subsystem. RRS is notified that the RM has failed by the following message:

```

ATR169I RRS HAS UNSET EXITS FOR RESOURCE MANAGER CSQ.RRSATF.IBM.MQV1 REASON:
UNREGISTERED

```

Next, we go to the RRS panel and select option Browse an RRS log stream, and then select RRS Resource Manager Data log to browse the RRS resource manager data log. We should see data as shown in Example 7-4:

Example 7-4 Resource Manager Data log

```

SC66      2003/12/05 14:12:39.408437 BLOCKID=0000000001B041D1
RESOURCE MANAGER=CSQ.RRSATF.IBM.MQV1      LOGGING SYSTEM=SC66
RESOURCE MANAGER MUST RESTART ON SYSTEM SC53
RESOURCE MANAGER WAS LAST ACTIVE WITH RRS ON SYSTEM SC53
LOG NAME IS CSQ3.MQ.RRS.IBM.MQV1

```

RRS has marked MQV1 as: must restart on system SC53.

By checking the RRS Resource Manager List (under the option Display/Update RRS related Resource Manager information on the main RRS panel), we can view the status of the MQV1 resource manager on SC53; see Example 7-5.

Example 7-5 Resource Manager List

```
RRS Resource Manager List          Row 1 to 11 of 1
Command ==>                               Scroll ==> PAGE
```

Commands: v-View Details u-View URs r-Remove Interest

S	RM Name	State	System	Logging Group
	CSQ.RRSATF.IBM.MQCB	Run	SC53	WTSCPLX1
	CSQ.RRSATF.IBM.MQJ2	Run	SC53	WTSCPLX1
	CSQ.RRSATF.IBM.MQV1	Reset	SC53	WTSCPLX1
	DSN.RRSATF.IBM.DB7A	Run	SC53	WTSCPLX1
	DSN.RRSATF.IBM.DB7J	Run	SC53	WTSCPLX1
	DSN.RRSATF.IBM.DB7L	Run	SC53	WTSCPLX1
	DSN.RRSATF.IBM.D7V1	Run	SC53	WTSCPLX1
	DSN.RRSPAS.IBM.DB7A	Run	SC53	WTSCPLX1
	DSN.RRSPAS.IBM.DB7J	Run	SC53	WTSCPLX1
	DSN.RRSPAS.IBM.DB7L	Run	SC53	WTSCPLX1
	DSN.RRSPAS.IBM.D7V1	Run	SC53	WTSCPLX1

Resource manager CSQ.RRSATF.IBM.MGV1 shows as Reset, which means that the resource manager is no longer known to RRS on system SC53.

We now restart MQV1 on SC52 to see what happens when we attempt to start a resource manager on a different system in the sysplex, when RRS has not marked it restart anywhere; see Example 7-6.

Example 7-6 MQV1 restart log

```
CSQR001I -MQV1 RESTART INITIATED
CSQR003I -MQV1 RESTART - PRIOR CHECKPOINT RBA=00002E34381C
CSQR004I -MQV1 RESTART - UR COUNTS - 744
IN COMMIT=0, INDOUBT=0, INFLIGHT=1, IN BACKOUT=0
CSQR007I -MQV1 UR STATUS 745
T CON-ID          THREAD-XREF      S   URID          TIME
- - - - -
R RRSBATCH                F00002E3429E3 2003-12-05 14:13:02
....
...
CSQR030I -MQV1 Forward recovery log range 756
from RBA=00002E34381C to RBA=00002E3443AA
CSQR005I -MQV1 RESTART - FORWARD RECOVERY COMPLETE - 757
IN COMMIT=0, INDOUBT=0
CSQR032I -MQV1 Backward recovery log range 758
from RBA=00002E3443AA to RBA=00002E3429E3
CSQR006I -MQV1 RESTART - BACKWARD RECOVERY COMPLETE - 759
INFLIGHT=0, IN BACKOUT=0
CSQR002I -MQV1 RESTART COMPLETED
```

We see that MQV1 begins normal restart processing, finds one inflight UR from its logs, and backs it out. Note that MQV1 does not need to communicate to RRS to resolve this inconsistency since an inflight UR will always be resolved by being backed out.

MQV1 then attempts to perform restart processing with RRS on SC52, as shown in Example 7-7.

Example 7-7 MQV1 restart processing

```
CSQM065E -MQV1 CSQMIGQA MQCONN failed, reason=2350
CSQ3017I -MQV1 CSQ3RRSR RRS function ATRIBRS failed, RC=00000F02
```

MQ issues the CSQ3017I message to tell us that ATRIBRS has failed with a return code of F02. By referring to *MVS Programming: Resource Recovery*, we find that a return code F02 from an ATRIBRS call means an ATR_RESTART_WRONG_SYSTEM error.

If now we view the RRS resource manager list panel, we see the content shown in Example 7-8.

Example 7-8 Resource Manager List after MQV1 restart

```
RRS Resource Manager List                                Row 1 to 13 of 1
Command ==>                                           Scroll ==> PAGE

Commands: v-View Details u-View URs r-Remove Interest

S  RM Name                                           State      System  Logging Group
   BBO.CLC11.CLUC11.WSC11.IBM                      Reset     SC52    WTSCPLX1
   BBO.CLHA1.BBON004.BBON004.IBM                    Run       SC52    WTSCPLX1
   BBO.CLHA1.CLHA1.WSC11D.IBM                      Set       SC52    WTSCPLX1
   BBO.CLHA1.CLUC11.WSC11.IBM                      Run       SC52    WTSCPLX1
   CSQ.RRSATF.IBM.MQV1                             Set      SC52    WTSCPLX1
   CSQ.RRSATF.IBM.MQ4D                              Run       SC52    WTSCPLX1
   DFHRXDM.SCSCERW3.IBM                            Run       SC52    WTSCPLX1
   DSN.RRSATF.IBM.DB4D                              Run       SC52    WTSCPLX1
   DSN.RRSATF.IBM.D7V1                             Run       SC52    WTSCPLX1
   DSN.RRSPAS.IBM.DB4D                              Run       SC52    WTSCPLX1
   DSN.RRSPAS.IBM.D7V1                             Run       SC52    WTSCPLX1
   HWS.IM4DCONNVO21.SVL.SANJOSE.IBM                Run       SC52    WTSCPLX1
   IMS.IM4D____V081.STL.SANJOSE.IBM                Run       SC52    WTSCPLX1
```

MQV1 shows a state of Set, which indicates the resource manager has started and set its exits with RRS, but has not started RRS restart processing. This is because RRS rejected the ATRIBRS call from MQV1. MQV1 is now *not* in a position to use RRS facilities. This will prevent it from joining a queue sharing group because it uses RRSF to attach to DB2, for example.

In this scenario we could stop MQV1 on SC52 and restart it on SC53—or we could cancel RRS on SC53. Note that both actions are disruptive.

Remember that each resource manager exhibits different behavior in this type of scenario. WebSphere Application Server, for example, cannot start without being able to connect to RRS, so in this scenario it would just abend on startup on SC53. WebSphere Application Server only restarts on SC52 if RRS remains active on SC52.

Note: z/OS 1.6 eases the restrictions that can result in the previous scenario.



Part 3

RRS exploiters

In this part, we take a closer look at resource managers that use RRS services.

Archived

WebSphere Application Server for z/OS

In this chapter, we describe how WebSphere Application Server for z/OS V5.1 uses RRS.

The chapter covers the following topics:

- ▶ Introduction to WebSphere Application Server V5
- ▶ J2EE™ terminology
- ▶ WebSphere Application Server features that exploit RRS
- ▶ WebSphere Application Server connectors for DB2, CICS, IMS and WebSphere MQ
- ▶ Restart and recovery issues with RRS
- ▶ Test scenarios showing RRS archive logs

8.1 Introduction

WebSphere Application Server for z/OS is a part of the WebSphere Application Server family from IBM. Websphere Application Server provides a Java 2 Enterprise Edition (J2EE) and Web Services-compliant application server. WebSphere Application Server for z/OS V5 is compliant with the J2EE 1.3 specification.

WebSphere Application Server provides the ability to run an EJB application that may connect to many data sources (such as IMS, CICS or DB2) in a fully transactional environment.

WebSphere Application Server for z/OS exploits many z/OS functions to leverage the qualities of service provided in a z/OS environment. For example, WLM is used for the scheduling of application servers and Sysplex Distributor can be used to provide high availability configurations in a sysplex environment.

RRS is used by WebSphere Application Server as an integral part of its transactional support. Before describing how WebSphere Application Server uses RRS, we discuss transactional support in general in a J2EE 1.3 environment and explain J2EE terminology.

8.2 J2EE terminology

In this section we briefly discuss J2EE terminology as it relates to transaction support in a J2EE environment. The transactional model used by J2EE conforms to open standards such as the X/Open DTP model.

A J2EE application server implementation provides a transaction manager that coordinates updates to various resource managers. The transaction manager must be able to manage work contexts and must be able to move work contexts from one transaction manager to another. The specification that defines how transaction managers implement these functions is called Java Transaction Services (JTS). JTS maps to the CORBA Object Transaction Service (OTS) specification. JTS uses most of the same interfaces as OTS, and communications between transaction managers uses the IIOP protocol, as with OTS.

The transaction manager must communicate with the application server, the application programs, and the resource managers. The standard API used to provide these functions is called the Java Transaction API (JTA). The JTA defines a set of high level interfaces that a transaction manager, application program, and resource manager use to enable transactional support. In this chapter we focus on the connection between the transaction manager and a resource manager.

The X/Open XA protocol defines how a transaction manager communicates with resource managers with two-phase commit support. In the J2EE platform, the XA protocol is implemented by the JTA XAResource interface.

A J2EE platform provides support for two type of transactions: resource manager local transactions (RMLT) and JTA (or global) transactions. A *global transaction* supports updates to multiple distributed resource managers within one unit of recovery, whereas a *local transaction* may only update one resource manager within a unit of recovery.

A resource manager must be XA-compliant in order to participate in a global transaction unless the provider of the J2EE platform and resource manager build in extra support for two-phase commit processing not based on XA.

The JTA specification defines three resource manager types, as described here:

- ▶ JDBC™-compliant database managers

Java Database Connectivity (JDBC) is a standardize Java interface to a database manager. The database can be a relational database like DB2, or non-relational like IMS/DB. The JDBC driver may or may not be XA-compliant. Many database manager providers supply both an XA-compliant and a non-XA compliant JDBC driver.

- ▶ JMS Providers

Java Message Service is a standardize Java interface to an enterprise messaging system like MQSeries. The JMS driver may be XA-compliant or non-XA-compliant.

- ▶ J2EE Connector Architecture (JCA)-compliant enterprise information systems

The J2EE JCA defines a standard architecture for connecting J2EE platforms to heterogeneous enterprise information systems (EIS). Examples of JCA-compliant EIS are the CICS transaction gateway and the IMS connect products. The interface between the application server and EIS is called a *resource adapter*. Resource adapters may be XA-compliant or non-XA-compliant.

A global transaction may access any or all of these resource manager types within a single transaction. The J2EE 1.3 specification says that a J2EE platform must allow access to the following resource manager types within the scope of a single transaction:

- ▶ A single JDBC database
- ▶ A single JMS provider
- ▶ Multiple JCA-compliant EISs, provided they provide XA-compliant resource adapters

8.3 RRS exploitation

Unlike the more traditional resource managers such as CICS or IMS, WebSphere Application Server for z/OS uses RRS services as a part of its base implementation to provide two-phase commit support. WebSphere Application Server for z/OS does not start without RRS being available, and it abends in the event of an RRS failure.

In addition to support of XA-capable resource managers, WebSphere Application Server for z/OS supports resource managers that provide an RRS-enabled resource adapter. WebSphere Application Server for z/OS refers to this support as *RRSTransactional*. Currently, any resource managers that are connected to using RRS must reside on the same z/OS image as the WebSphere Application Server.

Important: WebSphere Application Server *always* requires RRS, even if there are no RRSTransactional resource managers being used. WebSphere Application Server uses RRS services to provide part of its transactional support for XA-capable resource managers,.

We can look at three types of two-phase commit support provided by the transaction manager services in WebSphere Application Server for z/OS:

1. OTS support (distributed two-phase commit supported) as defined by OMG

OTS support uses RRS to hold expressions of interest and to take a SDSRM role.

2. XA J2EE (distributed two-phase commit supported)

If there are multiple RMs involved, then RRS is used to hold expressions of interest. WebSphere Application Server takes an SDSRM role. Note again that RRS is used even if no RRS enable adapters are involved.

3. RRS-enabled adapters

If all RMs involved in a transaction support RRS, then WebSphere Application Server allows RRS to take on the role of syncpoint coordinator. If there is a mixture of XA and RRS RMs, then RRS will take an SDSRM role.

Some resource adapters on z/OS may support both XA and RRS modes of operation, depending on how they are configured. The RRS mode of operation is usually restricted to instances where both WebSphere Application Server and the resource manager reside on the same z/OS image. Examples of these resource managers are discussed below.

A global transaction in WebSphere Application Server may use multiple resource managers that use either RRS or XA resource adapters, and WebSphere Application Server must coordinate the updates across all RMs. To do this, WebSphere Application Server generally takes an RRS SDSRM role so that it becomes the syncpoint coordinator and RRS drives the commit or backout processing only for those resource managers that support RRS. Refer to Chapter 3, “Distributed RRS” on page 23 for a discussion of distributed two-phase commit and how it is supported in RRS.

Even if there are no RRSTransactional resource managers involved in a global transaction, WebSphere Application Server still uses RRS to hold expressions of interest and it takes an RRS SDSRM role. WebSphere Application Server does not have its own logging system, so it uses RRS to provide this function. This is important when looking at restart and recovery issues because WebSphere Application Server requires access to data in RRS logs even if you never use RRS-enabled resource managers.

8.4 Connectors for JDBC, JMS and JCA

This section reviews the available connectors for WebSphere Application Server that allow an application to access resource managers such as DB2, IMS, CICS and MQseries. Here, we concentrate on the two-phase commit support provided with these adapters. For a full description of these adapters, see *WebSphere for z/OS V5 Connectivity Handbook*, SG24-7064.

8.4.1 IMS connectors

There are two ways to connect from WebSphere Application Server to IMS. One way is to use the IMS connect product, and the other way is to use the IMS JDBC support provided as a part of IMS/DB. In the following sections, we describe both methods in more detail.

IMS Connect V2.1

IMS connect provides a bridge to IMS from TCP/IP-attached or local clients using the IMS Java connector. IMS connect uses the OTMA interface to attach to IMS. WebSphere Application Server for z/OS supports using IMS connect in both TCP/IP attach or local attach mode. IMS connect always uses RRS because IMS OTMA attach requires that the client (IMS connect) be an RRS resource manager. The behavior of IMS connect depends on whether TCP/IP attach or local attach is used; here we describe both scenarios.

TCP/IP attach to IMS connect

The resource adapter for WebSphere Application Server for z/OS supplied by IMS connect is XA-compliant when operating in TCP/IP attach mode. This means that two-phase commit is supported from WebSphere Application Server transactions to a TCP/IP-connected IMS connect server. In this environment, IMS connect always takes on an RRS SDSRM role. This

allows IMS connect to act as a communications resource manager (CRM) and to interface to other external transaction managers using XA protocol.

In this environment, as shown in Figure 8-1, the IMS connect server and the OTMA-attached IMS system must reside on the same z/O image—and RRS must be active on that image. The WebSphere Application Server can be on a remote z/OS image, although it is possible to configure the resource adapter to attach in remote mode even if the WebSphere Application Server resides on the same z/OS images as IMS connect and the IMS system. However, this is not a recommended configuration. Local attach is more efficient and provides for better restart and recovery because RRS is used.

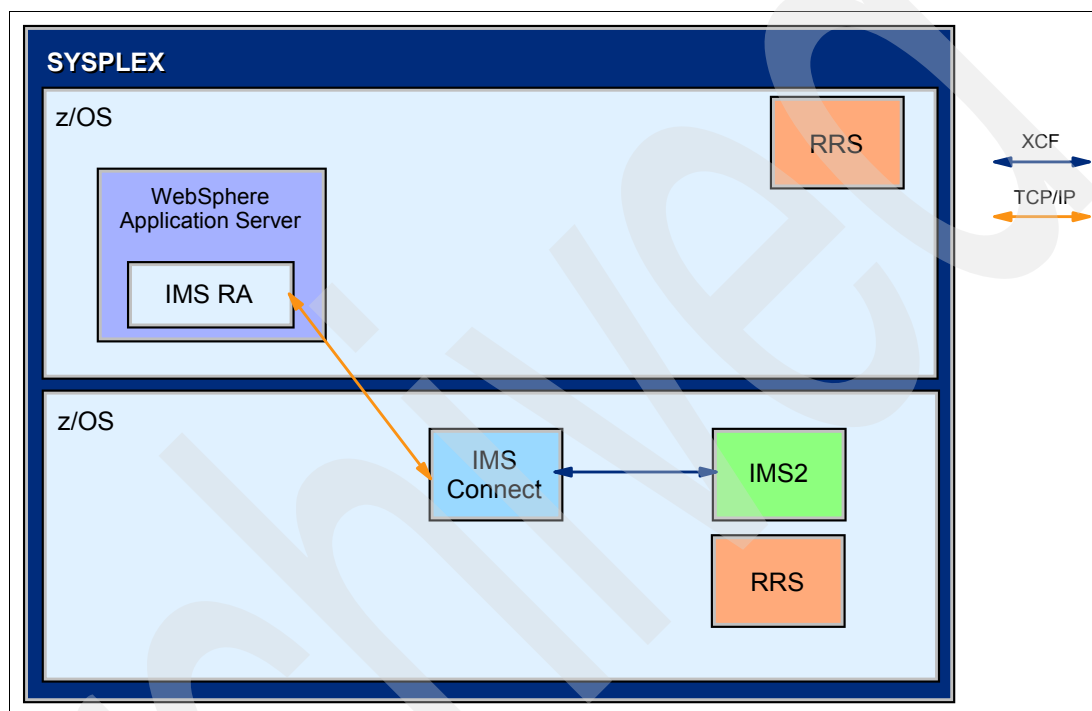


Figure 8-1 IMS connect in TCP/IP attach mode

WebSphere Application Server local attach to IMS connect

The resource adapter for WebSphere Application Server for z/OS supplied by IMS connect is RRSTransactional when operating in local attach mode. This means that two-phase commit is supported from WebSphere Application Server transactions to a locally connected IMS connect server.

In local mode, the resource adapter communicates to IMS connect using the MVS Program Call (PC) facility. In this environment IMS connect does not take on an RRS SDSRM role because it does not need to communicate with an external transaction manager using XA protocol. In this scenario, WebSphere Application Server may take an RRS SDSRM role for this transaction, depending on what other type of resource managers are involved. If all resource managers involved are RRSTransactional, then RRS is the syncpoint manager.

In the local attach environment, WebSphere Application Server, the IMS connect server and the OTMA-attached IMS system must reside on the same z/OS image and RRS must be active on that image; see Figure 8-2.

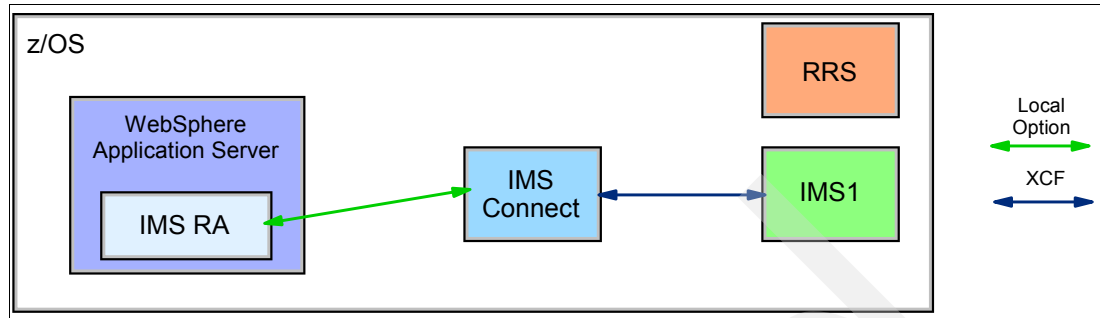


Figure 8-2 IMS connect in local attach mode

IMS JDBC connector

IMS V8 provides a JCA-compliant resource adapter that provides a JDBC interface to an IMS system. The connection to IMS is via the IMS Open Database Access (ODBA) interface.

From a WebSphere Application Server view, the resource adapter is RRSTransactional and must be run as a part of a global transaction. The IMS JDBC resource adapter does not support Localtran. In this scenario, WebSphere Application Server may take an RRS SDSRM role for this transaction, depending on what other type of resource managers are involved. If all resource managers involved are RRSTransactional, then RRS is the syncpoint manager. If there are XA resource managers involved, then WebSphere Application Server assumes the SDSRM role.

For the IMS JDBC connect, WebSphere Application Server, and the ODBA-attached IMS system must reside on the same z/OS image and RRS must be active on that image.

8.4.2 CICS connectors

WebSphere Application Server can connect to local or remote CICS systems using the CICS Transaction Gateway (CTG) product. The current version of CTG at the time of writing is Version 5.1.

CTG V5.1 is a set of client and server components that allow a Java application to invoke services in an attached CICS server. The CTG provides JCA-compliant resource adapters that use either CICS ECI or EPI. In a WebSphere Application Server for z/OS environment, only the ECI adapter is supported.

The CICS CTG can operate in two modes: local attach or remote attach. For remote attach, a CTG gateway address space is required on z/OS that communicates to a CICS system residing on the same z/OS image using the CICS EXCI interface.

WebSphere Application Server local attach to CTG

The ECI resource adapter for WebSphere Application Server for z/OS supplied by CTG is RRSTransactional when operating in local attach mode. This means that two-phase commit is supported from WebSphere Application Server transactions connect to a CICS system that resides on the same z/OS image.

In local mode, there is no requirement for the CTG Gateway address space. The resource adapter communicates to a CICS system directly using EXCI. In this scenario, WebSphere Application Server may take an RRS SDSRM role for this transaction, depending on what other types of resource managers are involved. If all resource managers involved are RRSTransactional, then RRS is the syncpoint manager.

In the local attach environment, WebSphere Application Server and the CICS systems must reside on the same z/OS image and RRS must be active on that image.

WebSphere Application Server TCP/IP attach to CTG

The ECI resource adapter for WebSphere Application Server for z/OS supplied by CTG is *not* XA-compliant when operating in remote attach mode. This means that two-phase commit is *not* supported from WebSphere Application Server transactions connect to a CICS system via the CTG gateway address space.

In remote mode, WebSphere Application Server on z/OS connects to a CTG Gateway address space via TCP/IP and the CTG gateway address space connects to a local CICS system using the EXCI interface.

Even though the remote CICS attachment does not support two-phase commit, it may be possible to include a connection to a remote CICS system as part of a WebSphere Application Server Global transaction. The JCA provides for an option called *last resource commit optimization* (also known as *last agent optimization*) which allows the use of a single one-phase commit resource manager in a global transaction as long as any other resource managers in the transaction support two-phase commit.

WebSphere Application Server V5 provides support for this using a function known as *Last Participant Support* (LPS). WebSphere Application Server will prepare all the RMs that support two-phase commit and if they all vote to commit, then WebSphere Application Server can issue a commit to the one-phase commit RM, followed by a commit to the two-phase commit RMs.

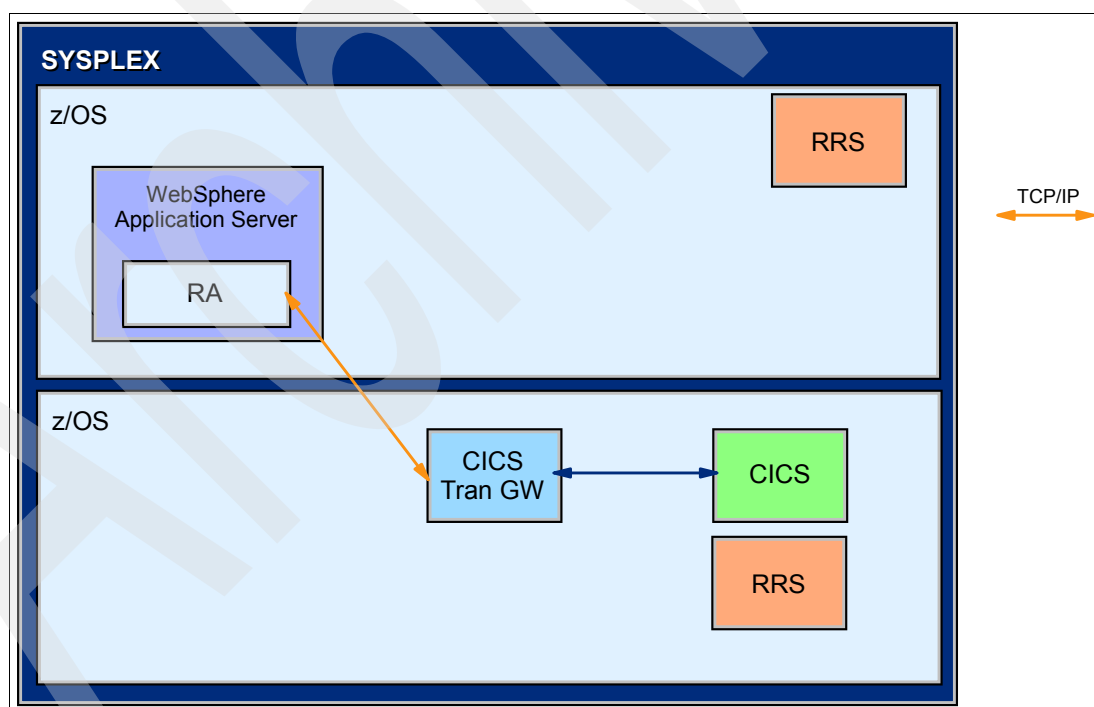


Figure 8-3 WebSphere Application Server connection to CICS using the CTG

8.4.3 DB2 connector

WebSphere Application Server can connect to DB2 using one of the DB2 for z/OS supplied Java Database Connectivity (JDBC) drivers. JDBC is a Java API used for accessing databases.

The JDBC standard defines four different type of drivers, as described here:

- ▶ **Type 1 driver**

This driver provides a bridge to an ODBC driver. It is not commonly used any more.

- ▶ **Type 2 driver**

This driver provides platform-specific and database-specific code to access the database. Type 2 drivers usually offer the best performance, but a specific driver is required for each platform and database combination.

- ▶ **Type 3 driver**

This driver provides a bridge to a middleware product that connects to the database. The driver is usually written in Java, and it is portable across platforms

- ▶ **Type 4 driver**

This driver connects directly to the database using a standard protocol supported by the database server. For example, Type 4 drivers for DB2 use the DRDA protocol to connect to a DB2 database server directly. Type 4 drivers are written entirely in Java and are thus much easier to port across platforms.

DB2 for z/OS provides two versions of JDBC driver, both of which are supported by WebSphere Application Server V5. In the following descriptions, we refer to a JDBC driver (which DB2 for z/OS supplies) and a WebSphere Application Server JDBC provider (which is an entry under the WebSphere Application Server admin panels used when defining a datasource).

DB2 for z/OS local JDBC driver

This driver is supplied with DB2 for z/OS V5 and above and is often called the “legacy JDBC driver”. The WebSphere Application Server V5 JDBC provider that supports this driver is called the DB2 for zOS Local JDBC provider (RRS).

It is a JDBC type 2 driver and is RRSTransactional. The type 2 driver uses the DB2 RRSAF interface to connect to DB2—and both WebSphere Application Server and the DB2 subsystem must reside on the same z/OS image.

IBM DB2 universal JDBC driver for z/OS

The DB2 universal JDBC driver for z/OS is supplied with DB2 for z/OS V8. It is also available as a PTF for DB2 for z/OS V7.

WebSphere Application Server V5 level W502002 supports this new driver by adding two new JDBC providers that can be configured, as described here:

DB2 universal JDBC Provider

Selecting this provider uses the driver without XA support; when configuring the datasource, you can specify if you want to use the driver as a type 2 or a type 4 driver.

If configured as a type 2 driver, this driver supports RRSTransactional and behaves like the legacy JDBC driver. RRSAF connects to the DB2 subsystem, which must be on the same z/OS image as WebSphere Application Server. A transaction in this environment must be defined as Global. WebSphere Application Server expresses an interest in the UR to RRS and takes on the RRS SDSRM role if there are other RMs involved in the transaction that are XA-compliant.

If configured as a JDBC type 4 driver, then there is no two-phase commit support provided. The DB2 subsystem can be local or remote because the DRDA protocol is used to connect to it. This environment supports only Localtran.

In an environment where both WebSphere Application Server and the DB2 subsystem reside on the same z/OS image, it is preferable to configure the driver as JDBC type 2 for performance reasons. RRSAF attach will always perform better than DRDA.

DB2 universal JDBC Provider (XA)

Selecting this provider uses the driver with XA support, and the driver can only be configured as a JDBC type 4 driver.

The DB2 subsystem can be local or remote because the DRDA protocol is used to connect to it. RRSAF cannot be used in this environment, but WebSphere Application Server still uses RRS to express an interest in the UR and takes the RRS SDSRM role if there are other RMs involved in the transaction that are RRSTransactional.

8.4.4 WebSphere MQ connector

The J2EE specification 1.3 requires that a J2EE platform must provide a Java message Service (JMS) implementation. JMS is the Java API used for accessing messaging services.

WebSphere Application Server V5 satisfies the J2EE 1.3 specification by including a JMS provider in the product. This is referred to as the Embedded WebSphere JMS Provider, the WebSphere JMS Provider, or simply embedded messaging. It can be installed optionally as a JMS provider. In addition, WebSphere provides the capability to install an external JMS provider, such as IBM WebSphere MQ or a generic JMS provider.

The following sections describe the Websphere MQseries JMS provider.

WebSphere MQ JMS provider

MQseries V5.3 provides an RRSTransactional driver. When you select this provider, you must configure the WebSphere Application Server Queue Connection Factory to use MQ Bindings support because MQ client connection is not supported.

In this environment, WebSphere Application Server and the Websphere MQseries Queue manager must reside on the same z/OS image. The driver uses the MQSeries RRS adapter to connect to the Queue manager. WebSphere Application Server expresses an interest in the UR to RRS and takes on the RRS SDSRM role if there are other RMs involved in the transaction that are XA-compliant.

8.4.5 Connector summary table

Table 8-1 provides a summary of the WebSphere Application Server connectors and indicates which ones are RRSTransactional or XA-compliant.

Table 8-1 Connector summary

Connector	RRSTransactional	XA-compliant?	Two-phase supported to remote system
DB2 for z/OS JDBC (RRS)	Yes	No	No
DB2 Universal JDBC non-XA type 2	Yes	No	No
DB2 Universal JDBC non-XA type 4	No	No	No

Connector	RRSTransactional	XA-compliant?	Two-phase supported to remote system
DB2 Universal JDBC XA type 4	N/A	Yes	Yes
CICS transaction Gateway	Yes	No	No
IMS Connect	Yes	Yes	Yes
IMS JDBC	Yes	N/A	No
WebSphere MQ JMS	Yes	N/A	No
WebSphere Application Server integrated JMS	No	Yes	Yes

In Table 8-1, the “Two-phase commit supported to remote systems” column indicates whether two-phase commit is supported to resource managers that reside on a different z/OS to WebSphere Application Server. Be aware that a No in this column does not necessarily mean an update to this remote resource manager cannot be a part of a WebSphere Application Server global transaction; as explained in 8.4.2, “CICS connectors” on page 78, WebSphere Application Server supports one instance of a one-phase commit-capable resource manager within a Global transaction as long as all the other resource managers involved support two-phase commit.

8.4.6 RRS versus XA resource adapters

In general, if you are connecting WebSphere Application Server to an resource manager on the same z/OS image and you have a choice of RRSTransactional or XA, you should use RRSTransactional for the following reasons:

- ▶ Performance is better with RRSTransactional because these connectors normally use native z/OS calls and resource-specific interfaces instead of the TCP/IP connectivity used with XA resource managers.
- ▶ Resource manager restart is better with RRS. RRS is superior because once a decision is hardened within RRS, resource managers can start independently of each other and they can resolve in-doubt URs by communicating to RRS. In contrast, XA RMs need to communicate with each other in order to communicate a final decision, so both resource managers need to be active at the same time for restart processing to work.

8.5 Restart and recovery issues with RRS

As mentioned, WebSphere Application Server for z/OS supports both X- capable resource managers and RRSTransactional resource managers. Information relating to the RRSTransactional resource managers and XA-capable resource managers is held in RRS. In addition, XA partner information is held in an HFS log.

8.5.1 RRS failure

If RRS fails, then WebSphere for z/OS abends. WebSphere Application Server requires RRS to provide its transactional support and cannot function without it. RRS must be restarted on the image before WebSphere Application Server can be restarted on that image.

Example 8-1 shows the messages produced if RRS becomes unavailable while WebSphere Application Server is running.

Example 8-1 RRS failure sample messages

```
BB000133I WEBSPPHRE FOR Z/OS STOP COMMAND ISSUED FOR SERVER WSDMN.  
BBOT0024A RRS HAS BECOME UNAVAILABLE. WEBSPPHRE MUST BE RESTARTED.  
BB000035W TERMINATING THE CURRENT PROCESS, REASON=C9C218F7.  
BB000009E WEBSPPHRE FOR Z/OS DAEMON WSDMN ENDED ABNORMALLY,  
REASON=C9C212C4.
```

RRS must be restarted before the WebSphere Application Server can be restarted. If RRS cannot be started, then WebSphere Application Server must be restarted on another z/OS image where RRS is operational and is a part of the same RRS logging group as the failed instance.

8.5.2 Failure and restart

In RRS terms, WebSphere Application Server is a communications resource manager, not a data resource manager. As such, it does not need to do the same style of restart processing as a database manager like DB2 or IMS. It does not need to do forward or backward recovery from logs or release retained locks. However, it *does* need to connect to any other transaction managers that were involved in any distributed transactions in order to inform them of the correct outcome of a transaction.

Prior to WebSphere Application Server V5.1, WebSphere Application Server had no logging system of its own. It used RRS to hold information on all transactions and on startup, it would retrieve this information from RRS. WebSphere Application Server V5.1 introduced an XA partner log, which holds information about XA resources accessed. The XA partner log can be an HFS file or a z/OS logger logstream. WebSphere Application Server V5.1 now uses both RRS and its XA partner log when resolving outstanding URs on restart.

As discussed in Chapter 7, “RRS restart and recovery” on page 61, when a resource manager restarts it goes through a restart process with RRS where it retrieves any outstanding URs that exist for it and then goes through recovery processing for each one.

On WebSphere Application Server startup, you see the messages reported in Example 8-2 indicating RRS restart processing.

Example 8-2 RRS restart messages

```
BBOT0012I SERVER PBOS001 IS WARM STARTING WITH RRS  
BBOT0008I TRANSACTION SERVICE RESTART INITIATED ON SERVER PBOS001  
BBOT0009I TRANSACTION SERVICE RESTART UR STATUS COUNTS FOR SERVER  
PBOS001: IN-BACKOUT=0, IN-DOUBT=0, IN-COMMIT=0
```

In this example there were no outstanding units of recovery.

In a restart situation, WebSphere Application Server tries to resolve any outstanding URs—but it may not be possible for it to do so if it needs to communicate with another transaction manager. There may be communication links down or the other transaction manager may not be available.

In this case, RRS has outstanding URs in which WebSphere Application Server has expressed an interest. As discussed in 8.5.3, “Peer restart and recovery” on page 84, this can

result in RRS refusing to allow WebSphere Application Server restart on any other system in the sysplex because RRS does not mark the WebSphere Application Server instance as “restart anywhere” until there are no outstanding URs for that instance.

Important: In z/OS 1.6, RRS permits a resource manager to restart on another system even if there are outstanding URs.

On WebSphere Application Server startup, you get the message shown in Example 8-3 if WebSphere Application Server cannot resolve in-doubt URs.

Example 8-3 Sample message for in-doubt UR

```
BBOT0015D OTS UNABLE TO RESOLVE ALL INCOMPLETE TRANSACTIONS FOR SERVER PBOS001. REPLY  
CONTINUE OR TERMINATE
```

If you reply CONTINUE, then RRS marks the WebSphere Application Server server as having to restart on this system. It may be necessary to manually resolve in-doubt URs to get around this issue.

With the introduction of the XA partner log in WebSphere Application Server v5.1, there is a new issue in that the XA partner log and the RRS logs must be in sync in order to coordinate restart processing for XA-compliant RMs. If either log has been recovered to a different point in time to the other, then there may be a mismatch. In this case you will get the message shown in Example 8-4.

Example 8-4 Log data mismatch sample message

```
BBOT0025D OTS HAS ENCOUNTERED A LOG DATA MISMATCH. REPLY CONTINUE IF THIS IS EXPECTED OR  
TERMINATE IF THIS IS UNEXPECTED
```

Reply TERMINATE, to ensure data integrity. Then investigate why the XA partner log and the RRS logs are out of sync.

8.5.3 Peer restart and recovery

If a failure occurs, Automatic Restart Manager (ARM) can restart WebSphere Application Server for z/OS and related server instances on the same system or on an alternate system in the sysplex. The latter condition is achieved through peer restart and recovery, which restarts the control region on another system and goes through the transaction restart and recovery process so that it can assign outcomes to transactions that were in progress at the time of failure. Resource managers (such as DB2) that were being accessed at the time of failure may hold locks that are scoped to a transaction UR (unit of recovery). Once an outcome has been assigned to a UR, the resource managers can, generally, drop those locks.

System requirement for restart and recovery

Make sure every system (your original system, as well as any systems intended for recovery) have the following installed:

- ▶ z/OS Version 1.2 or higher
- ▶ BCP APAR OA01584
- ▶ RRS APARs OA02556 and OA2556
- ▶ WebSphere Application Server for z/OS Version 5 or higher

The following products all support RRS. Individually, they also support peer restart and recovery, if the prerequisites are all properly installed:

- ▶ DB2 Version 7 or higher
- ▶ IMS Version 8 or higher
- ▶ CICS Version 1.3 or higher
- ▶ MQSeries Version 5.2 or higher

Note: If you do not plan to use peer restart, you do not need to abide by these functional prerequisites. Your system continues to use the restart in place function that already exists.

Important: When setting up the ARM policy for a sysplex, make sure that both systems have the same level of application server installed. For example, you cannot use a V5.0 application server to perform peer restart and recovery for a V5.1 application server.

Ensure that the location service daemon and node agent are already running on all systems that might be used for recovery. Otherwise, the recovering system might attempt to recover on a system that is not running the location service daemon and node agent. If this happens, the server will fail to start, and recovery will fail.

Check the service level of the application servers you are using for peer restart and recovery. Even though it is possible to perform peer restart and recovery across different service levels, minor differences in configuration data can prevent an application server from starting if the service level of the peer system is lower than the service level of the failed system.

When the post installer detects a service level difference, it sends a message to the operator asking if recovery should continue. You can set up your automation to provide a positive response to this message and allow recovery to continue under these conditions. However, if startup is attempted and fails, the transactions needing to be recovered are now associated with the peer system, and RRS must be stopped on that system before these transactions can be moved back to the failed system.

Therefore, we recommend that you modify your ARM policy to turn off peer restart and recovery while you are adding WebSphere Application Server service on your sysplex, to ensure that the systems performing peer restart and recovery are at the same service level as the failed system.

InFlight work and presumed abort mode

If you have a distributed transaction that spans several servers, transactional locks may be held by resource managers involved in that work. When a failure occurs before that distributed transaction has started to commit, WebSphere Application Server for z/OS and the resource managers go into *presumed abort mode*. In this mode, the resource managers abort (rollback) the transaction.

- ▶ The effect of a server failure or communications failure will vary, depending on which server is executing the work at the time of failure.
- ▶ An OTS timeout may be required to roll back the subordinate branches of the distributed transaction tree.

This can occur when you have a server B Web client that is driving a session bean in the same server. That session bean has executed work against entity beans in servers C and D. All of the servers are involved in the same distributed, global transaction.

Suddenly, server B fails while the session bean is InFlight (meaning it had not yet started to commit). Servers C and D are waiting for more work or the start of the two-phase commit

protocol, but while in this state, the transactional locks may still be held by the resource managers. So, the server roles are as follows:

- ▶ Server A: Servlet/JSP executed
- ▶ Server B: Session bean accessed
- ▶ Server C: Entity bean accessed
- ▶ Server D: Entity bean accessed

Once the timeout occurs, since we were InFlight at the time of the failure, we will roll back the transaction branch.

When local resource managers are involved, RRS will ensure that they are called to perform presumed abort processing. When doing recovery, RRS will work with the resource managers to ensure that the recovery is done properly. When a failure occurs while work is InFlight, RRS will direct the resource managers involved in the local UR to roll back.

The WebSphere Application Server for z/OS runtime always assumes that there is recovery to do. Every time a server comes up, it does something different, depending on what mode it is in:

- ▶ If the server is running in restart/recovery mode, it checks to see whether there is any recovery required. If so, it attempts to complete the recovery and either succeeds or terminates.
- ▶ If the server is running normally, the restart/recovery transaction does not have to complete before it takes on new work. Once it knows what the restart work is, it can begin to take in new work.

Handling new work during recovery

The procedures for the recovery of InFlight and InDoubt work have been described in some detail, but how is new work handled on a recovered server? Once the InDoubt and InFlight work has been completed, the WebSphere Application Server for z/OS server shuts down. A new application server configured for that system may now be started up to accept new work.

Special considerations must be taken to begin new work on a WebSphere Application Server for z/OS using IMS Connect after recovering to an alternate system. After the recovery completes, IMS Connect starts, but it is not usable without some manual intervention.

On the current IMS Connect WTOR, execute the commands **nn,viewhws** followed by **nn,opens XXX** where XXX is the IMS subsystem name displayed in the result of the **nn,viewhws** query. The IMS datastore needs to reflect ACTIVE status, as seen in Example 8-5.

Example 8-5 Datastore sample WTOR

```
*17 HWSC0000I *IMS CONNECT READY* IMSCONN
R 17,VIEWHWS
IEE600I REPLY TO 17 IS;VIEWHWS
HWSC0001I HWS ID=IMSCONN Racf=N
HWSC0001I Maxsoc=100 Timeout=12000
HWSC0001I Datastore=IMS Status=ACTIVE
HWSC0001I Group=IMSGROUP Member=IMSCONN
HWSC0001I Target Member=IMSA
HWSC0001I Port=9999 Status=ACTIVE
HWSC0001I No active Clients
HWSC0001I Port=LOCAL Status=ACTIVE
HWSC0001I No active Clients
```

At this point, IMS Connect will be ready for new work to be completed on the server.

When peer restart and recovery do not work

The major reason for recovery failure is if you experience a network outage while in the process of recovering. If the system cannot reach the superior or subordinate because the network is dead, communications cannot reestablish and the transaction cannot completely resolve.

When WebSphere for z/OS cannot automatically resolve all of the URs returned from RRS at restart, RRS will not allow WebSphere to move back to the home (original) system. If WebSphere tries to go back while URs are still incomplete, you will receive an error code (C9C2186A) and a message describing an F02 return code from ATRIBRS.

To get around this, manual resolution is required to mark the server for “restart anywhere.” RRS will do that after all of the URs in which WebSphere is involved are “forgotten.” If RRS fails to mark the server as “restart anywhere,” the server, upon failure, is required to start on the recovery system. This is undesirable because it does not allow you to move the server back to its true home system.

The ultimate goal is to resolve all transactions that WebSphere (the server instance-owned interests that could not complete recovery) is involved in, and then if necessary, remove all WebSphere interests that remain in those URs. After that is complete, browsing the RM data log will show if the resource manager is marked “restart anywhere.” Your goal is to see the following messages displayed:

```
RESOURCE MANAGER=BSS00.SY1.BBOASR4A.IBM  
RESOURCE MANAGER MAY RESTART ON ANY SYSTEM
```

You will not want to receive the following messages:

```
RESOURCE MANAGER=BSS00.SY2.BBOASR4A.IBM  
RESOURCE MANAGER MUST RESTART ON SYSTEM SY2
```

Note that if a server region is currently running, then RRS will mark the Resource Manager as having to restart on the current system. This is because RRS cannot make the decision to mark the resource Manager as “restart anywhere” until after the resource manager has ended and RRS can determine if there are any outstanding UR interests for that resource manager.

8.6 Example scenarios

In this section, we describe scenarios we ran to test RRS. For our test setup we used EJBs that were developed for the IBM Redbook *WebSphere for z/OS V5 Connector Handbook*, SG24-7064.

In these scenarios, the EJB updates both CICS and IMS resources within one transaction and we deployed the EJB in two different modes, as follows:

- ▶ The first mode used the RRS connector for both the IMS and the CICS connector.
- ▶ The second mode used the XA connector for IMS and the RRS connector for CICS.

Both modes permit Global tran support for the transaction. In both instances the IMS connector and IMS system were running on the same system as WebSphere Application Server, but in the second mode we could have run the IMS connector and IMS system on a remote system.

8.6.1 Application updating CICS and IMS using RRS connectors

For this test we configured the CICS and IMS resource adapters to use RRSTransactional support. The WebSphere Application Server application updated both CICS and IMS within the scope of one transaction.

Example 8-6 is an extract from the RRS Archive log showing entries related to a successful transaction.

Example 8-6 RRS Archive extract

```
SC48      2003/12/02 10:49:06.050724 BLOCKID=00000000412FACD1
URID=BA6919507E8F0000000001F1010F0000 JOBNAME=WSA11S  USERID=ASSR1
PARENT URID=00000000000000000000000000000000
SURID=N/A
WORK MANAGER NAME=BBO.CLHA1.CLUA11.WSA11.IBM
SYNCPOINT=Commit RETURN CODE=00000000
START=2003/12/02 15:49:06.038266 COMPLETE=2003/12/02 15:49:06.050490
EXITFLAGS=00840000
LUWID=USIBMSC.SCSCERW1 691951BA88FB 0001 TID=          GTID=

FORMATID=003284271494 (decimal) C3C20186 (hexadecimal)
GTRID=
BA691950551DB8230000003600000009BA1DDA49113244A40000018000000003090C060B
BQUAL=
BA691950551DB8230000003600000009BA1DDA49113244A40000018000000003090C060B00000001
RMNAME=IMS.IM4B_____V081.STL.SANJOSE.IBM ROLE=Participant
  FLAGS=10021000 PROTOCOL=PresumeAbort
  StateCheck EXIT RC=Uncalled
  Prepare     EXIT RC=00000000
  DistSp      EXIT RC=Uncalled
  Commit      EXIT RC=00000000
  Backout     EXIT RC=Uncalled
  EndUr       EXIT RC=Uncalled
  ExitFailed  EXIT RC=00000000
  Completion  EXIT RC=Uncalled
  OnlyAgent   EXIT RC=Uncalled
RMNAME=DFHRXDM.SCSCERW1.IBM ROLE=Participant
  FLAGS=10001000 PROTOCOL=PresumeAbort
  StateCheck EXIT RC=Uncalled
  Prepare     EXIT RC=00000000
  DistSp      EXIT RC=Uncalled
  Commit     EXIT RC=00000010
  Backout     EXIT RC=Uncalled
  EndUr       EXIT RC=Uncalled
  ExitFailed EXIT RC=00000010
  Completion  EXIT RC=Uncalled
  OnlyAgent   EXIT RC=Uncalled
```

Example 8-7 shows that WebSphere Application Server, which is the work manager from an RRS point of view, has issued a commit. RRS is the syncpoint manager and there are two resource managers: IMS.IM4B_____V081.STL.SANJOSE.IBM (our IMS system) and DFHRXDM.SCSCERW1.IBM (our CICS system). WebSphere Application Server, in this case, does not take an SDSRM role because in our transaction there are only RRSTransactional RMs involved.

For each resource manager we see that the Prepare, Commit and ExitFailed exits are driven by RRS. The CICS commit exit returns x'10' which signifies ATRX_FORGET. This means CICS has completed commit processing and requests RRS to delete its interest in this UR.

8.6.2 Application updating CICS and IMS with RRS and XA connectors

For the next test we configured the IMS resource adapter to use XA support by defining the IMS connection as remote. The CICS resource adapter remained as RRSTransactional. The WebSphere Application Server application updated both CICS and IMS within the scope of one transaction.

Example 8-7 is an extract from the RRS Archive log showing entries related to a successful transaction.

Example 8-7 RRS Archive log extract

```

SC48      2003/12/03 07:55:30.657321 BLOCKID=0000000041308871
          URID=BA6A34607E8F03740000009C010F0000 JOBNAME=IM4BCONN USERID=STC
          PARENT URID=00000000000000000000000000000000
          SURID=N/A
          WORK MANAGER NAME=HWS.IM4BCONNV021.SVL.SANJOSE.IBM
          SYNCPOINT=AtraCmt RETURN CODE=00000000
          START=2003/12/03 12:55:30.610100 COMPLETE=2003/12/03 12:55:30.657105
          EXITFLAGS=00800000
          LUWID=                                TID=                                GTID=

          FORMATID=003284271494 (decimal) C3C20186 (hexadecimal)
          GTRID=
BA6A346025B116230000003700000006BA1DDA49113244A40000018000000003090C060B
          BQUAL=
BA6A346025B116230000003700000006BA1DDA49113244A40000018000000003090C060B00000001
000002D800000008000000040001
          RMNAME=HWS.IM4BCONNV021.SVL.SANJOSE.IBM ROLE=SDSRM
          FLAGS=10000000 PROTOCOL=PresumeAbort
          StateCheck EXIT RC=Uncalled
          Prepare     EXIT RC=00000000
          DistSp      EXIT RC=Uncalled
          Commit       EXIT RC=00000000
          Backout      EXIT RC=Uncalled
          EndUr        EXIT RC=Uncalled
          ExitFailed   EXIT RC=Uncalled
          Completion  EXIT RC=Uncalled
          OnlyAgent   EXIT RC=Uncalled
          RMNAME=IMS.IM4B____V081.STL.SANJOSE.IBM ROLE=Participant
          FLAGS=10020000 PROTOCOL=PresumeAbort
          StateCheck EXIT RC=Uncalled
          Prepare     EXIT RC=00000000
          DistSp      EXIT RC=Uncalled
          Commit       EXIT RC=00000000
          Backout      EXIT RC=Uncalled
          EndUr        EXIT RC=Uncalled
          ExitFailed   EXIT RC=00000000
          Completion  EXIT RC=Uncalled
          OnlyAgent   EXIT RC=Uncalled

SC48      2003/12/03 07:55:30.671862 BLOCKID=0000000041308AC3
          URID=BA6A34607E8F000000000200010F0000 JOBNAME=WSA11 USERID=ASCR1
          PARENT URID=00000000000000000000000000000000
          SURID=N/A

```

```
WORK MANAGER NAME=BBO.CLHA1.CLUA11.WSA11.IBM
SYNCPOINT=AtraPrp RETURN CODE=00000000
START=2003/12/03 12:55:30.636966 COMPLETE=2003/12/03 12:55:30.671676
EXITFLAGS=00840000
LUID=USIBMSC.SCSCERW1 6A34624CE683 0001 TID= GTID=
```

```
FORMATID=003284271494 (decimal) C3C20186 (hexadecimal)
```

GTRID=

BA6A346025B116230000003700000006BA1DDA49113244A40000018000000003090C060B

BQUAL=

BA6A346025B116230000003700000006BA1DDA49113244A40000018000000003090C060B00000001

```
RMNAME=DFHRXDM.SCSCERW1.IBM ROLE=Participant
```

```
FLAGS=10001000 PROTOCOL=PresumeAbort
```

```
StateCheck EXIT RC=Uncalled
```

```
Prepare EXIT RC=00000000
```

```
DistSp EXIT RC=Uncalled
```

```
Commit EXIT RC=00000010
```

```
Backout EXIT RC=Uncalled
```

```
EndUr EXIT RC=Uncalled
```

```
ExitFailed EXIT RC=00000010
```

```
Completion EXIT RC=Uncalled
```

```
OnlyAgent EXIT RC=Uncalled
```

```
RMNAME=BBO.CLHA1.CLUA11.WSA11.IBM ROLE=SDSRM
```

```
FLAGS=10041000 PROTOCOL=PresumeAbort
```

```
StateCheck EXIT RC=Uncalled
```

```
Prepare EXIT RC=00000000
```

```
DistSp EXIT RC=Uncalled
```

```
Commit EXIT RC=00000000
```

```
Backout EXIT RC=Uncalled
```

```
EndUr EXIT RC=00000000
```

```
ExitFailed EXIT RC=00000000
```

```
Completion EXIT RC=Uncalled
```

```
OnlyAgent EXIT RC=Uncalled
```

In this case we have both an RRSTransactional resource manager and an XA resource manager involved. WebSphere Application Server provides the overall transaction management and will use RRS protocols to manage the RRSTransactional RM and XA protocols to manage the XA resource manager. To make things even more interesting, in this scenario the XA resource manager (IMS Connect) is using RRS for its connection.

At first glance it looks like there are two URs being independently controlled. There are two different RRS URIDs. The work manager for the first UR is IMS connect (HWS.IM4BCONN021.SVL.SANJOSE.IBM). The work manager for the second UR is WebSphere Application Server (BBO.CLHA1.CLUA11.WSA11.IBM). Both IMS connect and WebSphere Application Server take an RRS SDSRM role for their respective URs.

What is not so obvious from this RRS archive log is that WebSphere Application Server is communicating with IMS connect using XA protocols to ensure two-phase commit processing for the whole transaction. An X/Open identifier (XID) acts as the identifier for a distributed unit of work. An XID consists of a Global transaction Identifier (GTRID) and a Branch qualifier (BQUAL). RRS supports XIDs and WebSphere Application Server uses the XID to link the two URs. If you check the GTRID and BQUAL for both URs, you will see they are the same.

From an RRS point of view, the two URs are not related; WebSphere Application Server does not use RRS cascaded UR support. If you look in the RRS panels while this transaction is in-flight, you will see two independent URs being managed by two different work managers—each of which has a role of SDSRM.

WebSphere Application Server will use XA protocols to send prepare to IMS connect and issues an Prepare Agent UR to RRS for the CICS UR. Once IMS connect votes OK then WebSphere Application Server tell IMS connect to commit. We see IMS Connect issue an Commit Agent UR to commit the IMS updates and WebSphere Application Server will give the go-ahead to RRS to commit the CICS UR.

8.6.3 Application backout updating CICS and IMS with RRS and XA connectors

For this test we configured the IMS resource adapter to use XA support by defining the IMS connection as remote. The CICS resource adapter remained as RRSTransactional. The WebSphere Application Server application updated both CICS and IMS within the scope of one transaction. We cancelled the CICS system in the middle of the transaction before a commit WebSphere Application Server issued. Example 8-8 shows an extract of this event from the Archive log.

Example 8-8 RRS Archive extract sample

```

SC48      2003/12/03 08:02:57.863262 BLOCKID=00000000413099D1
          URID=BA6A35F77E8F03740000009F010F0000 JOBNAME=IM4BCONN USERID=STC
          PARENT URID=00000000000000000000000000000000
          SURID=N/A
          WORK MANAGER NAME=HWS.IM4BCONN021.SVL.SANJOSE.IBM
          SYNCPOINT=AtraBak RETURN CODE=00000000
          START=UNKNOWN                      COMPLETE=2003/12/03 13:02:57.862936
          EXITFLAGS=02000000
          LUWID=                                TID=                                GTID=

          FORMATID=003284271494 (decimal) C3C20186 (hexadecimal)
          GTRID=
          BA6A35F7CAE0C4030000003700000000CBA1DDA49113244A40000018000000003090C060B
          BQUAL=
          BA6A35F7CAE0C4030000003700000000CBA1DDA49113244A40000018000000003090C060B00000001
          000002D8000000008000000040001
          RMNAME=HWS.IM4BCONN021.SVL.SANJOSE.IBM ROLE=SDSRM
          FLAGS=10000000 PROTOCOL=PresumeAbort
          StateCheck EXIT RC=Uncalled
          Prepare     EXIT RC=Uncalled
          DistSp      EXIT RC=Uncalled
          Commit      EXIT RC=Uncalled
          Backout     EXIT RC=00000000
          EndUr       EXIT RC=Uncalled
          ExitFailed  EXIT RC=Uncalled
          Completion  EXIT RC=Uncalled
          OnlyAgent   EXIT RC=Uncalled
          RMNAME=IMS.IM4B____V081.STL.SANJOSE.IBM ROLE=Participant
          FLAGS=10000000 PROTOCOL=PresumeAbort
          StateCheck EXIT RC=Uncalled
          Prepare     EXIT RC=Uncalled
          DistSp      EXIT RC=Uncalled
          Commit      EXIT RC=Uncalled
          Backout     EXIT RC=00000010
          EndUr       EXIT RC=Uncalled
          ExitFailed  EXIT RC=Uncalled
          Completion  EXIT RC=Uncalled
          OnlyAgent   EXIT RC=Uncalled

SC48      2003/12/03 08:02:57.878463 BLOCKID=0000000041309C23

```

```

URID=BA6A35F77E8F000000000206010F0000 JOBNAME=WSA11 USERID=ASCR1
PARENT URID=00000000000000000000000000000000
SURID=N/A
WORK MANAGER NAME=BB0.CLHA1.CLUA11.WSA11.IBM
SYNCPOINT=AtraBak RETURN CODE=00000000
START=UNKNOWN COMPLETE=2003/12/03 13:02:57.878151
EXITFLAGS=02040000
LUWID=USIBMSC.SCSCERW1 6A35F9647264 0001 TID= GTID=

FORMATID=003284271494 (decimal) C3C20186 (hexadecimal)
GTRID=
BA6A35F7CAE0C4030000003700000000CBA1DDA49113244A40000018000000003090C060B
BQUAL=
BA6A35F7CAE0C4030000003700000000CBA1DDA49113244A40000018000000003090C060B00000001
RMNAME=DFHRXDM.SCSCERW1.IBM ROLE=Participant
  FLAGS=06010000 PROTOCOL=PresumeAbort
  StateCheck EXIT RC=Uncalled
  Prepare EXIT RC=Uncalled
  DistSp EXIT RC=Uncalled
  Commit EXIT RC=Uncalled
  Backout EXIT RC=00000030
  EndUr EXIT RC=Uncalled
  ExitFailed EXIT RC=Uncalled
  Completion EXIT RC=Uncalled
  OnlyAgent EXIT RC=Uncalled
RMNAME=BB0.CLHA1.CLUA11.WSA11.IBM ROLE=SDSRM
  FLAGS=10040000 PROTOCOL=PresumeAbort
  StateCheck EXIT RC=Uncalled
  Prepare EXIT RC=Uncalled
  DistSp EXIT RC=Uncalled
  Commit EXIT RC=Uncalled
  Backout EXIT RC=00000000
  EndUr EXIT RC=00000000
  ExitFailed EXIT RC=00000000
  Completion EXIT RC=Uncalled
  OnlyAgent EXIT RC=Uncalled

```

In this scenario we cancel our CICS system while the application is executing and while the RRS UR states are InFlight. WebSphere Application Server detects the resource adapter error and communicates with IMS connect using XA protocols to tell it to back out. WebSphere Application Server also calls RRS to back out the UR involving CICS.

As in 8.6.2, “Application updating CICS and IMS with RRS and XA connectors” on page 89 using mixed RRSTransactional and XA resource managers, we see that the GTRID and BQUAL ties the two RRS URs together.

Because the CICS system has been cancelled, the return code from the CICS backout exit is x'30' indicating ATRX_LATER. This means that CICS will provide a return code at a later time. When CICS restarts and begins restart processing with RRS, then it will provide RRS with the backout return code.

DB2 for z/OS

In this chapter, we describe how DB2 for z/OS V7 uses RRS.

This chapter covers the following topics:

- ▶ DB2 for z/OS V7 RRS requirements
- ▶ DB2 features that exploit RRS
- ▶ RRS facilities that are exploited
- ▶ Restart and recovery issues

9.1 DB2 RRS requirements

DB2 for OS/390 is the IBM relational database manager for OS/390. DB2 supports access to the data it manages from many different sources—batch jobs, traditional transaction management environments such as CICS and IMS/TM, and also the new environments such as WebSphere Application Server and Internet-based applications.

DB2 supports both single-phase and two-phase commit transactions. The type of commit that is supported for a given transaction is determined by the interface that is used to communicate with DB2. Similarly, the syncpoint manager that is used depends on the application and the environment. DB2 supports the traditional syncpoint managers CICS and IMS/TM, and will use these products as the syncpoint manager for transactions between these products and DB2. For DB2-to-DB2 communication using DDF, DB2 itself acts as the syncpoint manager. And since DB2 for OS/390 V5, it also supports the new RRS component of OS/390.

The resource manager that is responsible for global commit coordination for a given transaction depends on the scope of the transaction, where the transaction originated, and the interfaces and products used. It is vital that you understand all the concepts involved before you read the following discussion about DB2 capabilities. Apart from the programs that use RRSAF, RRS will only be involved in a DB2 transaction if an external syncpoint manager is required—and that requirement generally only exists if there is more than one resource manager involved in a transaction and a part of the transaction executes outside the control of that resource manager.

9.1.1 DB2 RRS Attach facility

The DB2 interface that uses RRS is called the Recoverable Resource Manager Services Attach Facility (RRSAF). As well as providing support for RRS, RRSAF is also the strategic call attachment for DB2—all new enhancements in this area will be in RRSAF. Two enhancements available only through RRSAF are the ability to have more than one connection between an address space and DB2, and the ability to sign on to DB2 using a new authorization ID and an existing plan and connection. RRSAF is described in detail in *DB2 for OS/390 Application Programming and SQL Guide*, SC26-8958.

The RRSAF interface can be used by programs that are written in Assembler, C, COBOL, Fortran, or PL/I. To commit or roll back work when using RRSAF, use the SRRCMIT or SRRBACK commands respectively. If the only recoverable resource being used by your application is within a single DB2 subsystem, then you can also use the DB2 COMMIT and ROLLBACK functions.

Note: You can not use RRSAF from within a CICS or IMS transaction.

To use RRSAF, you must link your application with the RRSAF language interface module, DSNRLI, and specify the appropriate parameters on the CALL DSNRLI statement. Alternately, you can use the ATTACH(RRSAF) option on the precompile. This is just another way of linking DSNRLI. If you use ATTACH(RRSAF), the code produced by the precompiler will call DSNRLI instead of the generic DSNHLI entry point, so you will be sure to pick up RRSAF code without having to worry about the linkage editor statements.

When using RRSAF, DB2 will register an interest in the transaction with RRS. The amount of overhead that is involved in using RRS depends on how many resource managers are involved and what type of commit command is invoked.

If an SQL COMMIT is issued, DB2 will check with RRS to see if any other resource managers are involved. If DB2 is the only resource manager involved in the transaction, DB2 will then coordinate the commit internally. There is no measurable overhead in this case.

If an SRRCMIT is issued, RRS will check to see if DB2 is the only resource manager involved in the transaction. If it is, RRS will invoke the “only agent” exit and request DB2 to do a one-phase commit. The processing associated with the only agent exit is all within the RRS address space. There is more overhead involved in this option than if an SQL COMMIT was used.

Finally, if an SRRCMIT is issued, and DB2 is *not* the only resource manager, then full two-phase commit processing will be invoked and the overhead will be higher due to the additional PREPARE and COMMIT processing.

Generally, the recommendation is to use SQL COMMIT if DB2 is the only resource manager you are using. However if your transaction also uses another resource manager, then you must change the SQL COMMIT to an SRRCMIT. If you do not change the SQL COMMIT, and DB2 determines that there is another resource manager involved in the transaction, the SQL call will fail with a -925/-926 SQL code.

There is no instrumentation in DB2 or DB2PM to tell you how many times RRS was called, or the amount of processing involved as a result of the involvement of RRS.

When using RRSAF, DB2 will register an interest in the transaction with RRS. The amount of overhead that is involved in using RRS depends on how many resource managers are involved and what type of commit command is invoked.

RRSAF-capable interfaces to DB2 include:

- ▶ Batch/TSO
- ▶ WLM managed Stored Procedures (see “DB2 Stored Procedures”)
- ▶ JDBC
- ▶ SQLJ
- ▶ CLI/ODBC

9.1.2 DB2 Stored Procedures

If you use WLM-managed Stored Procedures address spaces, then RRS is a prerequisite. If you use the standard DB2-managed Stored Procedures address space (DSNSPAS), then RRS cannot be involved, and you cannot include other resource managers within a single unit of recovery. If you wish to use Stored Procedures that use RRS or involve other resource managers within the same UR, then you must use WLM-managed Stored Procedures.

If you have an existing Stored Procedure that you wish to start using RRS, all that must be done is to relink the program with DSNRLI and rebind the plan. RRS support is transparent to both the caller of the stored procedure and the stored procedure itself.

The stored procedure itself may not invoke any RRSAF functions directly, nor may it invoke SRRCMIT or SRRBACK.

DB2 stored procedures may update a number of resources including IMS or CICS resources as well as DB2. In this case, DB2 stored procedures may take an RRS DSRM role if it determines that it needs to be the syncpoint manager for a particular UR. See 3.1.1, “RRS distributed syncpoint support” on page 25 for a discussion on distributed syncpoint processing with RRS.

9.1.3 DB2 JDBC/SQLJ driver for OS/390

The DB2 JDBC/SQLJ driver for OS/390 is a type 2 JDBC driver that uses DB2 RRSF attach to allow a Java program connect to DB2 using JDBC or SQLJ. JDBC is an API that allows a Java program to access a database such as DB2. JDBC supports dynamic SQL only.

SQLJ is an API developed by IBM, Oracle® and Tandem to permit Java programs access to a relational database using static SQL. SQLJ can interoperate with JDBC so a Java application program can use both JDBC and SQLJ within one unit of work.

This driver can be used by any Java program running on z/O,S and it is also supported by WebSphere for z/OS V4.01 and later.

This driver is a JDBC type 2 driver. The JDBC standard defines four different type of drivers:

Type 1 driver	This driver provides a bridge to an ODBC driver; it is not commonly used any more.
Type 2 driver	A JDBC driver that calls platform-specific and database-specific code to access the database, these drivers usually offer the best performance—but a specific driver is required for each platform and database combination.
Type 3 driver	This driver provides a bridge to a middleware product that connects to the database. It is usually written in Java and is portable across platforms.
Type 4 driver	This driver connects directly to the database using the a standard protocol supported by the database server. For example, Type 4 drivers for DB2 use the DRDA protocol to connect to a DB2 database server directly. Type 4 drivers are written entirely in Java and are thus much easier to port across platforms.

This driver uses RRSF to connect to DB2 and therefore can only connect to a local DB2.

9.1.4 DB2 Universal JDBC/SQLJ driver

The DB2 universal JDBC driver for z/OS is a new driver supplied with DB2 for z/OS V8. It is also available as a PTF for DB2 for z/OS V7. This driver can be configured as a type 2 or type 4 driver. From an RRS perspective, we are really only interested in this driver when operating as a type 2 driver.

When configured as a type 2 driver, it behaves like the older DB2 JDBS/SQLJ driver for OS/390. It uses RRSF to connect to a local DB2 for z/OS. WebSphere for z/OS V5.02 supports the new DB2 driver. DB2 stored procedures only support this driver when configured in JDBC type 2 mode.

When configured as a type 4 driver, it can operate in two modes: non-XA or XA. The X/Open XA protocol defines how a transaction manager communicates with resource managers with two-phase commit support. In the J2EE platform, the XA protocol is implemented by the JTA XAResource interface. WebSphere for z/OS V5 supports this driver in both XA and non-XA modes of operation. DB2 stored procedures do not support this driver in JDBC type 4 mode.

If configured as a non-XA JDBC type 4 driver then there is no two-phase commit support provided. The DB2 subsystem can be local or remote because the DRDA protocol is used to connect to it.

If configured as an XA JDBC type 4 driver, then there is two-phase commit support provided to both local and remote DB2s. RRS is not used for two-phase commit support when configured as a type 4 driver.

In an environment where both WebSphere Application Server and the DB2 subsystem reside on the same z/OS image, it is preferable to configure the driver as JDBC type 2 for performance reasons. RRSAF attach will always perform better than DRDA.

9.2 DB2 restart and recovery with RRS

On restart DB2, like any other DBMS, performs extensive restart and recovery processing. The DB2 for z/OS Administration Guide discusses DB2 restart and recovery processing in detail. Here, we concentrate on how DB2 interacts with RRS on restart and recovery, and we look at scenarios that may force manual intervention to resolve in-doubt URs with RRS.

The normal restart and recovery phases that DB2 goes through are:

- ▶ **Phase 1: Log initialization**
DB2 identifies the last LOG RBA used before termination so that it can start logging at the next RBA.
- ▶ **Phase 2: Current status rebuild.**
DB2 determines what URs are outstanding and the status of each UR (in-flight, in-commit, in-abort or in-doubt). DB2 also recovers information about coordinator and participants for all outstanding URs.
- ▶ **Phase 3: Forward Log Recovery**
Having determined the outstanding URs in Phase 2, DB2 makes all the database changes for committed work as well as for in-flight, in-doubt and in-abort URs. For in-flight, in-doubt and in-abort URs, DB2 locks the changed data to make it unavailable.
- ▶ **Phase 4: Backward Log Recovery**
- ▶ In this phase, DB2 reverses out changes by in-flight or in-abort URs and releases locks for those URs.

During Phase 2 of DB2 restart processing, DB2 retrieves any outstanding URs from RRS. DB2 calls the `Begin_restart_processing (ATRIBRS)` RRS service to begin restart processing with RRS. RRS returns a list of URs in which DB2 has expressed an interest. DB2 then retrieves information for each outstanding UR using the `Retrieve_interest_data (ATRRID)` RRS call.

9.2.1 DB2 restart if RRS is unavailable

If DB2 restarts and RRS is not available, then it cannot resolve any URs that were in-doubt where RRS was the syncpoint manager and DB2 was a participant. For any URs in which DB2 was coordinator (DB2 WLM-managed stored procedures, for example) or in which the state was in-flight, in-commit or in-abort, DB2 does not need to retrieve UR data from RRS. It can process those UR by backing out any in-flight or in-abort URs.

Any in-doubt URs may result in retained locks for affected data. If there are in-doubt URs, DB2 will issue messages DSN3010I or DSN3011I to indicate that RRS cannot be contacted to resolve these in-doubt URs.

DB2 will resync with RRS once RRS is restarted on the z/OS image. At that stage it can resolve any in-doubt URs according to the final state that RRS supplies.

Note that when DB2 starts and RRS is unavailable, any DB2 facilities that require RRS will also be unavailable. That means any attempt to use RRSAF or WLM managed stored procedures will result in an error. For example, an attempt to start a WLM stored procedure results in the error message shown in Example 9-1.

Example 9-1 WLM stored procedure sample startup messages

```
DSNX982I DSNX9WLM ATTEMPT TO PERFORM RRS ATTACH FUNCTION SPAS_ID  
  FAILED WITH RRS RC = 00000008 RSN = 00F30091 SSN = D7V1  
  PROC= D7V1SPAS ASID = 4F WLM_ENV =WLMENV
```

Support is provided with APAR UQ85390 for DB2 or z/OS V7.1 to display a message once DB2 RRS restart processing is complete; the message is shown in Example 9-2:

Example 9-2 RRS attachment messages sample

```
DSN3029I csect-name RRS ATTACH PROCESSING IS AVAILABLE
```

9.2.2 DB2 restart on another system

As discussed in Chapter 4, “Implementing RRS” on page 31, prior to z/OS V1.6, RRS will mark a resource manager as having to restart on the same system if there are outstanding URs for that resource manager when it terminated and RRS has remained operational on that system.

If DB2 terminates and there are outstanding URs, then RRS will reject any attempt by DB2 to restart on another system in the same RRS logging group. DB2 will receive an return code of F02 from the Begin_restart_processing (ATRIBRS) RRS call indicating that DB2 cannot register with RRS on this system. If there are in-doubt URs, then DB2 will issue messages DSN3010I or DSN3011I to indicate that RRS cannot be contacted to resolve these in-doubt URs because DB2 is being restarted on the wrong system.

This has the same effect as if RRS is unavailable; in-doubt URs cannot be resolved and RRSAF and WLM stored procedures will be unavailable. The difference in this scenario is that in order to allow DB2 to connect to RRS, DB2 must be stopped and either restarted on the z/OS system it was originally running on, or you must manually remove any outstanding URs in which DB2 has an interest. Then DB2 can be restarted on the new system.

Note that this issue only occurs on releases of z/OS prior to V1.6 and only when RRS has stayed operational on the original system. It is assumed that if DB2 terminates and the z/OS system remains operational, DB2 will be restarted on the same system.

9.3 Sample scenarios for DB2 using RRS

Our test scenario consisted of an MVS batch program written in C that updates both DB2 and MQ in one unit of work. The program uses RRSAF to attach to DB2 and uses the MQ RRS attach support to connect to MQSeries. The program uses the RRS SRRCMIT and SRRBACK calls to perform commit and backout processing.

9.3.1 Normal commit processing scenario

In this scenario, we see RRS panel displays and RRS log extracts showing what happens during a normal run of the batch job where we update both DB2 and MQ and then commit the updates. The C program we are running waits for a period after updating DB2 and MQ but before issuing a SRRCMIT. This gives us time to display panels and so on.

Before we start the batch job we look at the status of the DB2 (subsystem D7V1) and MQ (subsystem MQV1) resource managers by looking at the RRS ISPF resource manager list panel; see Example 9-3.

Example 9-3 RRS resource manager list panel

RRS Resource Manager List

Row 3 to 13

Command ==>

Scroll ==>

Commands: v-View Details u-View URs r-Remove Interest

S	RM Name	State	System	Logging
	CSQ.RRSATF.IBM.MQV1	Run	SC53	WTSCPLX1
	DSN.RRSATF.IBM.DB7A	Run	SC53	WTSCPLX1
	DSN.RRSATF.IBM.DB7J	Run	SC53	WTSCPLX1
	DSN.RRSATF.IBM.DB7L	Run	SC53	WTSCPLX1
	DSN.RRSATF.IBM.D7V1	Run	SC53	WTSCPLX1
	DSN.RRSPAS.IBM.DB7A	Run	SC53	WTSCPLX1
	DSN.RRSPAS.IBM.DB7J	Run	SC53	WTSCPLX1
	DSN.RRSPAS.IBM.DB7L	Run	SC53	WTSCPLX1
	DSN.RRSPAS.IBM.D7V1	Run	SC53	WTSCPLX1
	IMS.IMSC____V081.STL.SANJOSE.IBM	Reset	SC53	WTSCPLX1
	IMS.IMG____V091.STL.SANJOSE.IBM	Reset	SC53	WTSCPLX1

This panel shows the two resource managers that we are interested in; DSN.RRSATF.IBM.D7V1 is the DB2 RRSAF resource manager, and CSQ.RRSATF.IBM.MQV1 is the MQseries RRS attach resource manager. Both RMs have an RRS state of Run, meaning they are registered to RRS on this system (SC53) and have completed RRS restart processing.

We now start our batch job and display the RRS UR details using the RRS ISPF panels; see Example 9-4.

Example 9-4 RRS UR details view

RRS Unit of Recovery List		Row 1 to 1 of 1
RRS Unit of Recovery Details		Row 1 to 2
Command ==>		Scroll ==> P
Commands r-Remove Interest v-View URI Details		
UR identifier : BB67A49B7E5CEDD00000001001010000		
Create time : 2004/06/22 02:44:42.478089		Comments :
UR state : InFlight		UR type : Prot
System : SC53		Logging Group : WTSCPLX1
SURID : N/A		
Work Manager Name : SC53.MURPHYAR.0026		
Display Work IDs		Display IDs formatted
Luwid . . : Not Present		
Eid . . : Not Present		
Xid . . : Not Present		
Expressions of Interest:		
S	RM Name	Type Role
	CSQ.RRSATF.IBM.MQV1	Prot Participant
	DSN.RRSATF.IBM.D7V1	Prot Participant

In this panel we see the details on the UR. Both RMs have an RRS role of Participant. This means that RRS acts as syncpoint coordinator.

We can relate the RRS UR to a DB2 thread by issuing the DB2 command DISPLAY THREAD(*) RRSURID(*); see Example 9-5.

Example 9-5 Display Thread Command sample output

```
-D7V1 DIS THREAD(*) RRSURID(*)
DSNV401I -D7V1 DISPLAY THREAD REPORT FOLLOWS -
DSNV402I -D7V1 ACTIVE THREADS - 447
NAME      ST A  REQ ID      AUTHID  PLAN      ASID TOKEN
RRSAF     T    13          MURPHYA  RRS006    0026 1186
  V480-DB2 IS PARTICIPANT FOR RRS URID=BB67A49B7E5CEDD000000001001010000
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I -D7V1 DSNVDT '-DIS THREAD' NORMAL COMPLETION
```

When the RRSURID keyword is used on the DISPLAY THREAD command, DB2 adds RRS-related information in message DSNV480I if DB2 is a participant. Note that the full message id is not displayed but the output is prefixed with V480. In Example 9-5, we see that DB2 is a participant as we would expect, given the previous RRS panel display.

DB2 issues the RRS information using message DSNV481I if DB2 is the coordinator for this UR.

When the batch program finally commits the updates, the UR goes to a state of forgotten and we can look in the RRS Archive log to see a record of the commit; see Example 9-6.

Example 9-6 RRS Archive log sample

```
RRS/MVS LOG STREAM BROWSE DETAIL REPORT

READING ATR.WTSCPLX1.ARCHIVE      LOG STREAM

SC53      2004/06/21 22:46:42.504857 BLOCKID=0000000079876D89
  URID=BB67A49B7E5CEDD000000001001010000 JOBNAME=MURPHYAR USERID=MURPHYA
  PARENT URID=00000000000000000000000000000000
  SURID=N/A
  WORK MANAGER NAME=SC53.MURPHYAR.0026
  SYNCPOINT=Commit RETURN CODE=00000000
  START=2004/06/22 02:46:42.484323 COMPLETE=2004/06/22 02:46:42.504683
  EXITFLAGS=00800000
  LUWID=                                TID=                                GTID=

  FORMATID=                                (decimal)                (hexadecimal)
  GTRID=

  BQUAL=

  RMNAME=CSQ.RRSATF.IBM.MQV1              ROLE=Participant
  FLAGS=10001000 PROTOCOL=PresumeAbort
  StateCheck EXIT RC=Uncalled
  Prepare     EXIT RC=00000000
  DistSp      EXIT RC=Uncalled
  Commit      EXIT RC=00000010
  Backout     EXIT RC=Uncalled
  EndUr       EXIT RC=Uncalled
  ExitFailed  EXIT RC=Uncalled
  Completion  EXIT RC=Uncalled
  OnlyAgent   EXIT RC=Uncalled
```



```
RMNAME=DSN.RRSATF.IBM.D7V1          ROLE=Participant
FLAGS=10001000 PROTOCOL=PresumeAbort
StateCheck EXIT RC=Uncalled
Prepare     EXIT RC=00000000
DistSp      EXIT RC=Uncalled
Commit      EXIT RC=00000010
Backout     EXIT RC=Uncalled
EndUr       EXIT RC=Uncalled
ExitFailed  EXIT RC=Uncalled
Completion  EXIT RC=Uncalled
OnlyAgent   EXIT RC=Uncalled
```

Here we see that the UR has been successfully committed. RRS has called the Prepare and Commit exits for both MQ and DB2. The return code of x'10' from the Commit exit signifies ATRX_FORGET, the commit was successful and the RM has set the UR state to forgotten.

Archived



CICS Transaction Server

In this chapter, we discuss how CICS/TS V2.2 uses RRS.

This chapter covers the following topics:

- ▶ CICS/TS V2.2 RRS requirements
- ▶ CICS/TS features that exploit RRS
- ▶ RRS facilities that are exploited
- ▶ Restart and recovery issues

10.1 CICS RRS requirements

CICS Transaction Server is an online transaction processing system where transactions are executed. The CICS TS execution environment can be structured as a single layer or it can be structure up to three layers, each specifically dedicated to manage a particular environment:

- ▶ The Terminal Owing Region (TOR) is dedicated to managing the interfaces with the network. The TOR is usually considered the front end for the incoming workload.
- ▶ The Application Owing Region (AOR) is where the application logic resides and where transactions are executed.
- ▶ The File Owing Region (FOR) is where the I/O operations against VSAM files are executed. To retrieve DL1 or DB2 data, the application issues an external CALL to either IMS/DB or DB2 (there is no local DL1 support since CICS Transaction Server).

CICS regions are interconnected through the use of Cross Memory services (MRO) within the same z/OS image, through the use of XCF services in a sysplex, or through the use of ISC communication over VTAM® (ISC) if they are located across multiple z/OS images.

As discussed in the following sections, CICS TS provides a two-phase commit protocol for its unit of work (UOW). What is new in this environment is the external CICS interface (EXCI), which has been enhanced in CICS TS 1.3 to provide resource recovery controlled within the client program. This new external CICS interface facility uses RRS as a two-phase commit coordinator.

10.1.1 Working in CICS

In this section we illustrate different ways of initiating a transaction within a CICS region, and of communicating with other resource managers outside that CICS region.

The term *transaction* has a specific meaning in a CICS environment: it involves all the processing executed for a specific type of request. A request type may represent a whole process (such as making an airline reservation), or a subcomponent of that process (such as assigning a seat on the aircraft). The application design determines the duration of the single transaction and provides this information to the CICS system by using explicit or implicit syncpoint.

Transactions in CICS can be of different types:

- | | |
|------------------------------|---|
| Conversational | This involves more than one input from the terminal, so that transaction and user enter into a conversation |
| Non-conversational | This has only one input. The transaction processes that input, responds to the terminal, and terminates. |
| Pseudo-conversational | <p>This contains a series of non-conversational transactions that look to the user like a single conversational transaction involving several interactions with the user. Each transaction in the sequence handles one input, sends back the response, and terminates.</p> <p>This application design is the most common, since pseudo-conversational transactions normally use less virtual storage and also hold exclusive resources for a shorter time than conversational mode. There are other reasons to choose the pseudo-conversational model, which are explained in detail in <i>CICS Application Programming Guide</i>, SC33-1687.</p> |

One of the issues in transaction execution is the usage of resources in terms of contention, integrity and recoverability in case of a failure. The level of contention might be influenced by the type of transaction, while the integrity and recoverability of the recoverable/protected resource is ensured by CICS.

CICS ensures that changes to the recoverable resources made by a unit of work (UOW) are either made completely, or not at all. A UOW is a sequence of actions that needs to be completed before any of the individual actions can be considered as complete. Usually a UOW is equivalent to a transaction—unless that transaction issues a CICS SYNCPOINT command, in which case the UOW lasts between syncpoints. From a CICS perspective, the UOW is the same as the unit of recovery. If a transaction that consists of multiple UOWs fails, committed UOWs are *not* backed out.

A synchronization point or *syncpoint* is a specific time, either explicitly or implicitly invoked by the transaction, where all the updates to protected resources must either be committed or backed out. CICS TS is capable of ensuring integrity and consistency of resources, both within a single CICS region and distributed over the interconnected systems in a network, by applying the two-phase commit processing at syncpoint time.

During syncpoint processing, CICS TS invokes each local resource manager that has updated recoverable/protected resources during the UOW. The local resource managers then perform the required action. This provides the means of coordinating the actions performed by individual resource managers.

If the execution of a UOW is distributed across more than one system, the CICS syncpoint managers (or their equivalents) in each connected system ensure that the effects of the distributed UOW are atomic. Each CICS issues the requests necessary to effect two-phase syncpoint processing to each of the connected systems with which a UOW may be in conversation.

In each connected system in a network, the CICS syncpoint manager uses interfaces to its local syncpoint manager connectors (RMCs) to communicate with the partner syncpoint manager. The RMCs are the communication resource managers (LU6.2, LU6.1, MRO and RMI) that have the function of understanding the transport protocols and controlling the flows between the connected systems during syncpoint.

As remote resources are accessed during UOW execution, the CICS syncpoint manager keeps track of the data describing the status of its end of the conversation with that RMC. The CICS syncpoint manager also assumes responsibility for the coordination of two-phase syncpoint processing for the RMC. See Figure 10-1 on page 106 for a diagram of the resource managers that CICS communicates with.

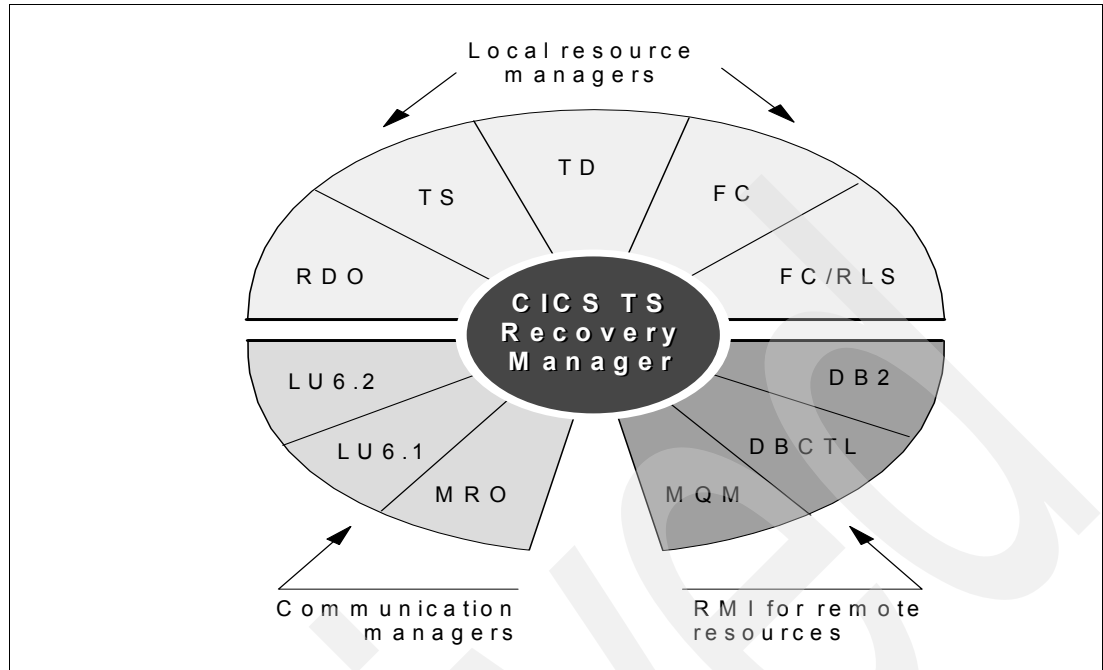


Figure 10-1 The resource managers that CICS communicates with

There is only one exception in providing a two-phase commit protocol, and it is in case of going out of CICS TS by using an LU6.2 protocol to an application environment other than CICS TS (for example, DB2 Stored Procedure or IMS/TM). In this case, in order to have an atomic syncpoint across the distributed UOW, it is necessary that the target application environment registers with RRS as an agent so that RRS can provide synchronization at commit time on the remote site as part of the two-phase commit protocol.

The following section briefly introduces all the existing possibilities for connecting to a CICS subsystem. It also introduces an enhancement to the External CICS interface (EXCI), called Transactional EXCI, in which the syncpoint coordination has been extended outside of CICS to fully support a distributed transaction environment.

10.1.2 Connecting to CICS via EXCI

Here we briefly describe the external interfaces that can be used to start transactions or to execute programs in a CICS environment. Consideration and details are provided only for the EXCI interface, in which starting from CICS TS 1.3, support is provided to fully implement a distributed transaction model by using RRS as an external syncpoint manager. In such a configuration, the syncpoint is initiated outside of the CICS TS environment and CICS acts as a syncpoint agent.

For more detailed information, refer to *CICS Recovery and Restart Guide*, SC33-1698, *CICS Application Programming Guide*, SC33-1687, *CICS Transaction Server for OS/390 External Interface Guide*, SC33-1944, and *CICS Intercommunication Guide*, SC33-1695.

EXCI Interface

The External CICS interface (EXCI) is an application programming interface that enables a non-CICS program (a client program) that is running in a z/OS address space to call a program (a server program) running in a CICS region, and to pass and receive data by means of a communication area. The CICS application program is invoked as if linked to by another

CICS application program. The programming interface allows a user to allocate and open sessions to a CICS region and to pass DPL requests over them.

Before CICS TS 1.3, one of the limitations of EXCI was that syncpoint processing had to be performed upon completion of the CICS program by using the SYNCONRETURN option on the DPL request. Therefore, any CICS resources that were updated were either committed or rolled back when the EXCI client program regained control. Non-CICS resources were not part of the CICS UOW, and updates were not committed upon completion of each DPL request.

CICS TS1.3 enhances the EXCI interface. In addition to the existing support, you are now able to extend a single CICS UOW to span multiple DPL requests to the same CICS region or to several different CICS regions, as well as make calls to other resource managers such as DB2, WebSphere MQ and IMS.

Figure 10-2 shows a diagram of the extended capability of the EXCI with the participation of non-CICS resource managers in the same UOW.

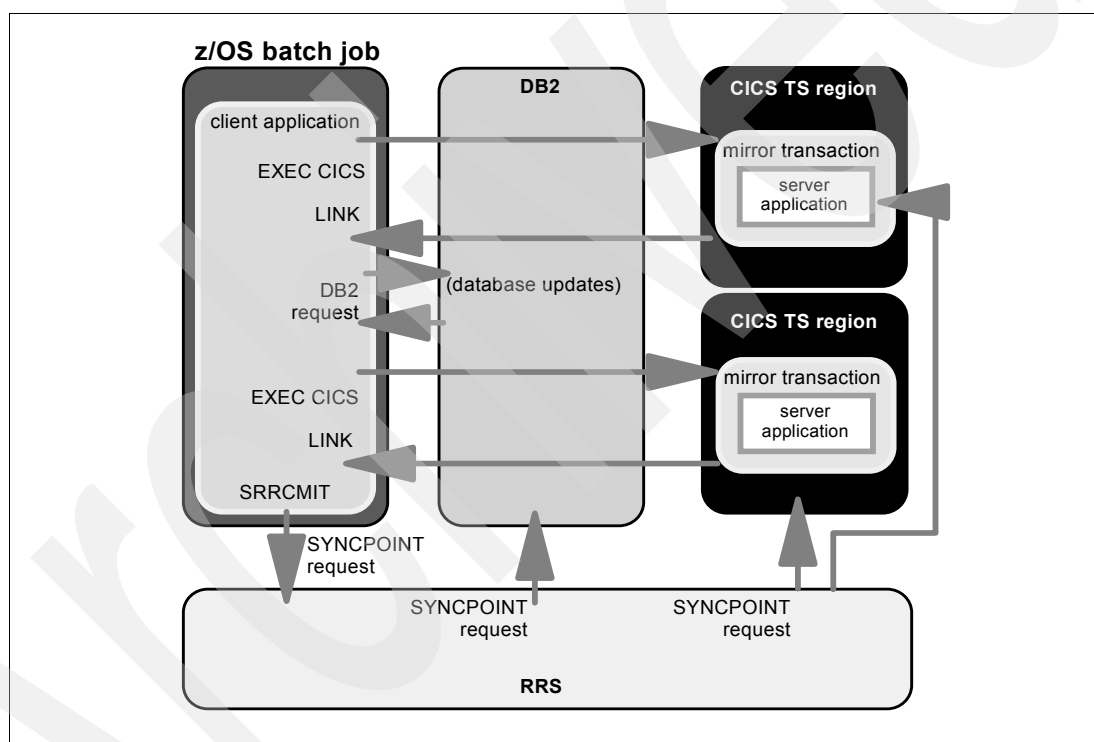


Figure 10-2 EXCI flow

In CICS TS 1.3 environment, a CICS server program that is invoked by an external CICS interface request can update recoverable resources. The client program can determine when syncpointing should occur. There are two options:

- ▶ Resource recovery controlled by CICS server region.
In this case, changes to recoverable resources are committed at the completion of each DPL request, independently of the client program. Also, in addition to the syncpoint taken when the server program returns control to CICS (with the SYNCONRETURN option), the server program can take explicit syncpoints during execution.
- ▶ Resource recovery controlled by the EXCI client program with the support of recoverable resource management services (RRS).

When the client program requests it, updates made by the server program in a single or even in multiple successive DPL requests are committed together.

To support this option, CICS and the EXCI interface both make use of RRS services. CICS acts as a resource manager, and the client program may issue requests to other resource managers and have resources owned by those resource managers committed in the same unit of recovery (UR), where UR represents all the resource managers that are going to be called at syncpoint time.

These options are controlled as follows:

- ▶ By the DPL_opts parameter of the DPL_request
- ▶ By the SYNCONRETURN option, either specified or omitted, on the EXEC CICS LINK PROGRAM command

Specifying SYNCONRETURN, a syncpoint is taken on completion of each DPL request. If SYNCONRETURN is omitted, a syncpoint (implicit or explicit) is taken within the client application program. Implicit syncpoint is taken when the EXCI program ends. Implicit syncpointing is not recommended because there can be situations when the EXCI client program cannot determine the actual result of an implicit syncpoint. Furthermore, in some cases, high level languages suppress errors that should result in backout. Therefore, explicit syncpoint is recommended.

Explicit syncpointing is controlled by coding the RRS API for commit or rollback in the client program. Once the client program decides that the current UR has to be committed, it should use one of the following RRS APIs to either commit or back out the changes:

- ▶ To commit, use **SRRCMIT**
- ▶ To back out, use **SRRBACK**

Application programming models

There are two different application programming models when using EXCI, as explained here:

EXCI call with SYNCONRETURN option

In this model, neither CICS nor the EXCI API use RRS services.

This technique offers the following advantages.

- ▶ The programming model is very simple.
- ▶ The EXCI client and target CICS region can reside in different z/OS images in a sysplex.

The technique also has a disadvantage:

- ▶ CICS commits resource updates every time it returns to the requesting EXCI client. Therefore you cannot implement spanned UOW and potentially sophisticated application code to handle error situations if the application is performing updates across more than EXCI calls.

Note: In this model, the client must be in the same sysplex as the CICS server.

EXCI call without SYNCONRETURN option

Omitting SYNCONRETURN means that you are using RRS services to control syncpointing within the EXCI client. Explicit SYNCPOINT requests should be removed from the server application. The flow is as follows:

1. When the CICS SIT parameter RRMS=YES is specified, CICS TS registers with RRMS as a resource manager at CICS startup.

2. When the EXCI client program issues a DPL_request that does not specify the SYNCONRETURN option, RRS is involved to assign a context token to the EXCI call.
3. The UR identifier is included on the DPL request that is passed to the CICS server region. If the DPL request is the first within the UR, CICS calls RRS and expresses interest in this UR, attaches a new mirror transaction, and links to the server application program. The server program completes its work, which might include updates to recoverable resources and/or daisy-chaining to other CICS regions.
4. When the server program completes, it returns control to the client program.
5. When the client program is ready to commit or back out the changes, the program invokes RRS to begin the two-phase commit protocol.
6. RRS acts as main coordinator and either:
 - a. Asks the resource managers to prepare to commit all updates within the UR. If all the interested resource managers vote yes, RRS tells them to commit the changes. If any resource manager votes no, RRS tells the resource managers to perform backout.
 - b. Tells all the interested resources managers to perform backout of all changes made with the UR.

The UR is now complete and CICS detaches the mirror task.

This technique offers the following advantages.

- ▶ A single UR can consist of multiple DPL calls initiated by multiple EXCI calls.
- ▶ While accessing CICS resources through EXCI, you can also update resources controlled by other resource managers that support the RRS interface in the same UOW.
- ▶ The application logic on the client side decides at which point in time a syncpoint should be taken. In this way it is possible to invoke a single logical commit across different resource managers, including multiple CICS regions.

The technique also has a disadvantage:

- ▶ The target server program has to reside on the same z/OS image as the client program. Even with this restriction, the target server program can be chained to a CICS region in a different image within a sysplex.

Note: Transactional EXCI is supported only by CICS TS 1.3 and later.

Mixing the two options (with SYNCONRETURN and without SYNCONRETURN) in the same EXCI client application is possible. However, each time that an EXCI CALL or LINK with SYNCONRETURN is issued, a new UOW is created and it is separated from the UOW that is created by an EXCI without SYNCONRETURN.

10.2 CICS restart and recovery with RRS

In these sections, we give examples of messages your syslog will show as a result of various failure scenarios.

10.2.1 RRS failure

In this scenario, CICS TS is using RRS to coordinate DPL requests that are issued without the SYNCONRETURN option. In case of RRS failure and unavailability, the CICS TS region will *not* be able to satisfy this type of request; however, all other types of request will be processed by the CICS region.

When RRS fails, the CICS region recognizes that the RRS exit manager is unavailable and and issues the message shown in Example 10-1.

Example 10-1 CICS syslog during RRS failure

```
DFHRX0105I SCSCERW1 The Resource Recovery Services (RRS) exit manager ATR.EXITMGR.IBM is now unavailable.
.....
DFHRX0104I SCSCERW1 The Resource Recovery Services (RRS) exit manager ATR.EXITMGR.IBM is now available.
DFHRX0106I SCSCERW1 Restart processing with Resource Recovery Services (RRS) is beginning.
DFHRX0107I SCSCERW1 Restart processing with Resource Recovery Services (RRS) has ended.
```

Although in this scenario the CICS region continues to run and process work, transactions that use RRS to coordinate their updates cannot be successfully executed. When RRS returns a status of available, CICS will re-establish a connection with RRS and you should receive message DFHRX0104 to confirm that the exit manager is available once more.

While RRS is not running, transactional EXCI requests might suffer several errors; the most common errors are:

- ▶ ARXC - CICS either did not register as a resource manager with RRS because system initialization parameter RRMS=NO was specified, or the RX domain made requests to another CICS system).
- ▶ ARXA - RRS might have been shut down after the request was received by CICS.

10.2.2 CICS restart

At startup, if RRMS=YES has been specified in the SIT parameter table, CICS TS registers with RRS. “RX” is the acronym CICS uses to identify the RRMS domain manager. Example 10-2 shows the messages issued during normal CICS startup.

Example 10-2 CICS startup

```
DFHRX0100I SCSCERW3 RX domain initialization has started.
DFHRX0104I SCSCERW3 The Resource Recovery Services (RRS) exit manager ATR.EXITMGR.IBM is
now available.
DFHRX0101I SCSCERW3 RX domain initialization has ended.

DFHRX0106I SCSCERW3 Restart processing with Resource Recovery Services (RRS) is beginning.
DFHRX0107I SCSCERW3 Restart processing with Resource Recovery Services (RRS) has ended.
```

10.2.3 Operator commands

CEMT INQUIRE RRMS

You can issue the CEMT INQUIRE RRMS command to the CICS region to explore the RRS environment. This command can be used to determine whether or not CICS accepts inbound transactional EXCI work. A status of Open means that CICS *will* accept inbound transactional EXCI units of work.

Example 10-3 CEMT Inquiry RRMS

```
CEMT I RRMS
```

```
STATUS:  RESULTS
Rrm Ope
```

CEMT INQUIRE UOW

CEMT INQUIRE UOW can be used to obtain information about a named unit of work, or about all the UOWs currently in the system. It displays the state of the UOW (for example, INDOUBT) and whether it is active, waiting, or shunted.

If you suspect a problem with either a recoverable data set or a connection, you can use INQUIRE UOW to display UOWs that have been shunted due to a connection or data set failure. The command, in some cases, displays the name of the resource that caused the UOW to be shunted, plus the transaction, user, and terminal that started it; see Example 10-4.

Example 10-4 Inquiry for a UOW status

```
CEMT I UOW
STATUS: RESULTS - OVERTYPE TO MODIFY
  Uow(BAABCB5ACE041) Inf Act Tra(CSOL) Tas(0000003)
      Age(00366451)                               Use(CICSTS )
```

```
CEMT I UOW
RESULT - OVERTYPE TO MODIFY
  Uow(BAABCB5ACE041)
  Uowstate( Inflight )
  Waitstate(Active)
  Transid(CSOL)
  Taskid(0000003)
  Age(00366934)
  Termid()
  Netname()
  Userid(CICSTS)
  Waitcause()
  Link()
  Sysid()
  Netuowid(..USIBMSC.SCSCERW1..vE.\...)
  Otstid()
```

10.2.4 CICS example

In our environment, we used a sample application running in the WebSphere Application Server and connecting to CICS TS through the CICS Transaction Gateway. The CICS gateway is indirectly involved, as the WebSphere CICS ECI J2C resource adapter runs CTG-supplied code. From an RRS point of view, during the life of this application, there should be two Resource Managers involved: BBO.CLHA1.CLUA11.WSA11.IBM, representing the WebSphere Application Server and DFHRXDM.SCSCERW1.IBM, representing the CICS region. This CICS region is started with the EXCI programming interface enabled. The syslog shows that the CICS region has been started with RRMS=YES.

Example 10-5 CICS startup syslog

```
DFHRX0100I SCSCERW1 RX domain initialization has started.
DFHRX0104I SCSCERW1 The Resource Recovery Services (RRS) exit manager
DFHRX0101I SCSCERW1 RX domain initialization has ended.

DFHRX0106I SCSCERW1 Restart processing with Resource Recovery Services
DFHRX0107I SCSCERW1 Restart processing with Resource Recovery Services
```

As shown in Example 10-6, the syslog also shows that the RRS connection is open.

Example 10-6 CEMT query to verify RRS service are operative

```
CEMT I RRMS

I RRMS
STATUS: RESULTS
Rrm Ope
```

At this point, the RRS panel shows that CICS registered to RRS as a resource manager. Selecting the option **Display/Update RRS related Resource Manager information** will display the entry FHRXDM.SCSCERW1.IBM, representing the CICS region with a status of Run; see Example 10-7.

Example 10-7 Display Resource Manager Status

Commands: v-View Details u-View URs r-Remove Interest

S	RM Name	State	System	Logging Group
	BBO.CLA11.CLUA11.WSA11.IBM	Reset	SC48	WTSCPLX1
	BBO.CLHA1.BBON001.BBON001.IBM	Run	SC48	WTSCPLX1
	BBO.CLHA1.CLHA1.SC48.IBM	Set	SC48	WTSCPLX1
	BBO.CLHA1.CLHA1DM.WSHA1DM.IBM	Run	SC48	WTSCPLX1
	BBO.CLHA1.CLUA11.WSA11.IBM	Run	SC48	WTSCPLX1
	BBO.CLHA1.WSHA1.WSHA1A.IBM	Reset	SC48	WTSCPLX1
	BBO.WHCELL.WHAGNTA.WHAGNTA.IBM	Run	SC48	WTSCPLX1
	BBO.WHCELL.WHCELL.SC48.IBM	Set	SC48	WTSCPLX1
	BBO.WHCELL.WHDMGR.WHDMGR.IBM	Run	SC48	WTSCPLX1
	BBO.WHCELL.WHSR01.WHSR01A.IBM	Run	SC48	WTSCPLX1
	BBO.WHCELLA.WHSR01.WHSR01A.IBM	Reset	SC48	WTSCPLX1
	CSQ.RRSATF.IBM.MQ4B	Run	SC48	WTSCPLX1
	DFHRXDM.SCSCERW1.IBM	Run	SC48	WTSCPLX1
	DSN.RRSATF.IBM.DB4B	Run	SC48	WTSCPLX1
	DSN.RRSPAS.IBM.DB4B	Run	SC48	WTSCPLX1
	HWS.IM4BCONNV021.SVL.SANJOSE.IBM	Run	SC48	WTSCPLX1
	IMS.IM4B____V081.STL.SANJOSE.IBM	Run	SC48	WTSCPLX1

Now the environment is ready to run the application. From the Web browser, you can issue a request to execute a trade operation (this requires a CICS transaction to be executed in order to update the VSAM file with the trading request). Transaction response time is very fast, so it is difficult to see transactions while they are in-flight in the RRS panel. However, you can track their execution in the RRS ARCHIVE log; Example 10-8 shows a sample log record of a transaction.

Example 10-8 CICS EXCI call reported in RRS ARCHIVE log stream

```
SC48      2004/01/29 17:01:26.163470 BLOCKID=00000000416EDD16
URID=BAB258B77E8C6000000003C5010E0000 JOBNAME=WSA11S  USERID=ASSR1
PARENT URID=00000000000000000000000000000000
SURID=N/A
WORK MANAGER NAME=BBO.CLHA1.CLUA11.WSA11.IBM
SYNCPOINT=Commit RETURN CODE=00000000
START=2004/01/29 22:01:26.158855 COMPLETE=2004/01/29 22:01:26.163262
EXITFLAGS=00840000
LUWID=USIBMSC.SCSCERW1 B258B78F69B8 0001 TID=
GTID=
```

```

FORMATID=003284271494 (decimal) C3C20186 (hexadecimal)
GTRID=
BAB258B739BA78030000004700000011BA1DDA49113244A40000018000000003090C060B
BQUAL=
BAB258B739BA78030000004700000011BA1DDA49113244A40000018000000003090C060B00000001
RMNAME=DFHRXDM.SCSCERW1.IBM          ROLE=Participant
FLAGS=10020000 PROTOCOL=PresumeAbort
StateCheck EXIT RC=Uncalled
Prepare      EXIT RC=Uncalled
DistSp       EXIT RC=Uncalled
Commit       EXIT RC=Uncalled
Backout      EXIT RC=Uncalled
EndUr        EXIT RC=Uncalled
ExitFailed   EXIT RC=00000000
Completion   EXIT RC=Uncalled
OnlyAgent    EXIT RC=00000000

```

This record documents the unit of recovery associated with our application request. The work manager name is our WebSphere Application Server, which is requesting the commit as a result of the transaction completion. The CICS region expressed interest in this UR, which is why it is now associated with a role of Participant. In this case, WebSphere Application Server is initiating the two-phase commit protocol, RRS is the coordinator, and CICS is participating as a resource manager. In our environment we decided to go a step further and put the EXCI request under monitoring. From a CICS session, we entered **CEDX CSMI,ON** to put the mirroring transaction under monitoring. As a result, when a request was sent from WebSphere Application Server to the CICS region, we could follow each major step of the transaction on the terminal that had been set up with the CEDX monitoring facility. At this point, selecting the RRS panel RRS Unit of Recovery showed that there was an in-flight unit of recovery associated with our environment; see Example 10-9.

Example 10-9 Unit of Recovery

Commands: v-View Details c-Commit b-Backout r-Remove Interest f-View UR Family

S	UR Identifier	System	Logging Group
		State	Type Comments
	BAC8FFC87E91A0000000005010E0000	SC48	WTSCPLX1
		InFlight	Prot

A request for the in-flight UR details provided the information shown in Example 10-10.

Example 10-10 In-flight Unit of Recovery details

Commands r-Remove Interest v-View URI Details

```

UR identifier : BAC8FFC87E91A0000000005010E0000
Create time : 2004/02/16 22:26:01.297790      Comments :
UR state : InFlight      UR type : Prot
System : SC48      Logging Group : WTSCPLX1
SURID : N/A
Work Manager Name : BB0.CLHA1.CLUA11.WSA11.IBM
  Display Work IDs      / Display IDs formatted
  Luwid . . : Present
  Eid . . : Not Present
  Xid . . : Present
Expressions of Interest:
S  RM Name      Type Role
   DFHRXDM.SCSCERW1.IBM  Prot Participant

```

A request for the work ID details provided the CICS region application ID, as shown in Example 10-11.

Example 10-11 Work ID details

UR identifier : BAC8FFC87E91A0000000005010E0000

More:

Logical Unit of Work Identifier (LUWID):
USIBMSC.SCSCERW1 C8FFC97F1F06 0001

NetID.LuName : USIBMSC.SCSCERW1
TP Instance : C8FFC97F1F06
SeqNum . . . : 0001

Enterprise Identifier (EID)
TID : (decimal)
GTID :

X/Open Transactions Identifier (XID)

Format ID : 003284271494 (decimal)
C3C20186 (hexadecimal)

GTRID : 00-0F BAC8FFC8 E3B53842 0000004C 00000006	.H.HT.....<....
10-1F BA1DDA49 113244A4 00000180 00000003u.....
20-23 090C060B

BQUAL : 00-0F BAC8FFC8 E3B53842 0000004C 00000006	.H.HT.....<....
10-1F BA1DDA49 113244A4 00000180 00000003u.....
20-27 090C060B 00000001

Finally, details about the Resource Manager involved were also displayed; see Example 10-12.

Example 10-12 Unit of Recovery Resource manager details

UR identifier : BAC8FFC87E91A0000000005010E0000
URI token . . : 7E6900000000004005500025555555
RM name . . . : DFHRXDM.SCSCERW1.IBM
Type : Prot Status . : ACTIVE
Role : Participant State . : InFlight
SURID : N/A

Exit/State	Status	Duration
BACKOUT . . .	: Uncalled	
COMPLETION . .	: Uncalled	
COMMIT	: Uncalled	
DSE/IN_DOUBT .	: Uncalled	
End_UR	: Uncalled	
EXIT_FAILED . .	: Uncalled	
ONLY_AGENT . .	: Uncalled	
PREPARE . . .	: Uncalled	
STATE_CHECK . .	: Uncalled	

IMS

In this chapter, we describe how IMS V8 uses RRS.

The chapter covers the following topics:

- ▶ IMS V8 RRS requirements
- ▶ IMS features that exploit RRS
- ▶ RRS facilities that are exploited
- ▶ Restart and recovery issues

11.1 How IMS/ESA exploits RRS

IMS/ESA has provided RRS support since V6. In this chapter we discuss IMS features that use RRS for connecting to both IMS/DB and IMS/TM.

IMS/ESA supports both single-phase and two-phase commit transactions—but traditionally, IMS/ESA is the commit manager. IMS/ESA TM provides support for distributed two-phase commit where IMS is not the start point of the transaction.

In this case, IMS/ESA is not the Global Sync-Point Manager (GSPM). IMS provides support for what it calls Distributed Sync Point Support. This support allows APPC and Open Transaction Manager Access (OTMA) and application programs and OTMA remote application programs to participate with IMS in protected conversation with coordinated resource updates. This is done by Resource Recovery Services (RRS) managing the syncpoint process on behalf of the conversation participants: the application program and IMS (acting as resource manager).

11.2 Connecting to IMS/ESA

In the following sections, we describe various ways to connect to IMS/ESA DB and TM that use RRS, as follows:

- ▶ ODBA
- ▶ APPC/IMS
- ▶ OTMA

11.2.1 ODBA

Customers need the ability to access IMS-managed databases from application processing environments that are not managed by IMS. Open Database Access (ODBA) is a callable interface that can be used by any OS/390 application program (that uses the Recovery Resource Services of OS/390 as a syncpoint manager) to issue DL/I calls to an IMS DB subsystem. The application and IMS must coexist on the same OS/390 image.

The ODBA interface allows IMS DB and OS/390 application programs to be developed, installed, and maintained independently of each other. This independence provides failure isolation and resource recovery using OS/390 Resource Recovery Services (RRS).

ODBA works on a Recoverable Resource (RRS)-managed application to an IMS DB-supported system; this means on a DBCTL-generated IMS system or a DB/TM IMS system.

The types of programs that can call the ODBA interface are DB2 for OS/390 stored procedures, WebSphere Application Server for z/OS, IMS Object Connector applications, and other OS/390 applications.

The ODBA interface resides in an OS/390 address space. The OS/390 address space is recognized by IMS as an OS/390 application region. The ODBA interface uses the IMS Database Resource Adapter (DRA) to communicate with IMS DB, as shown in Figure 11-1.

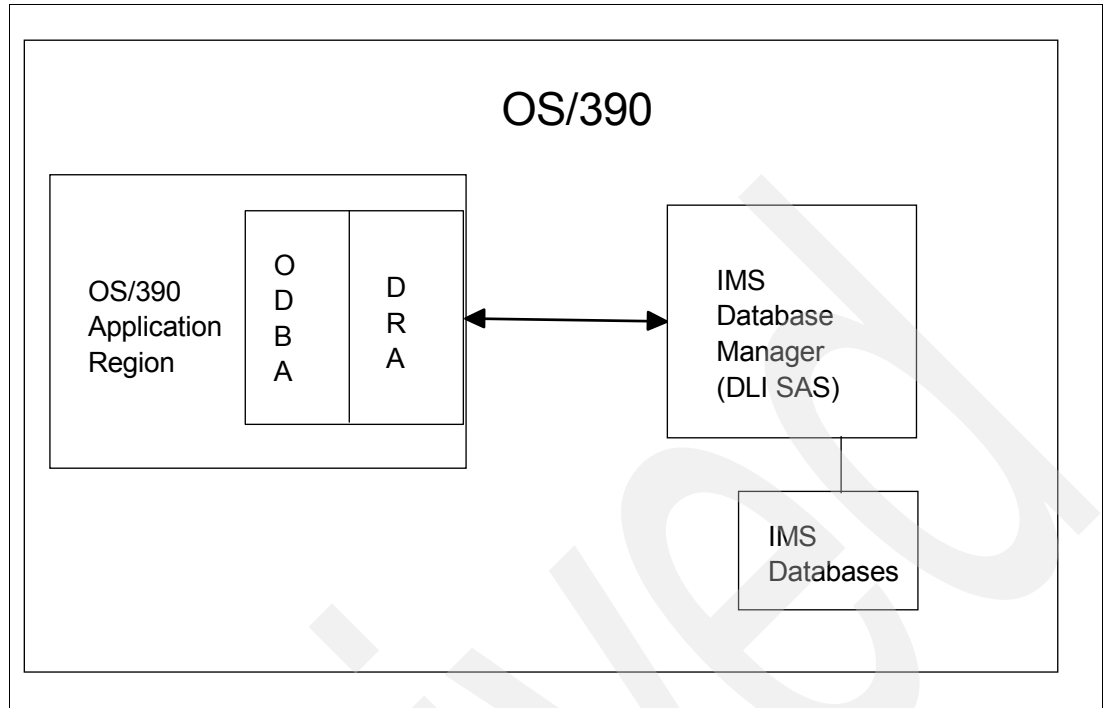


Figure 11-1 ODBA configuration

An example of an ODBA implementation is the JDBC driver provided by IMS/ESA. This driver is used by WebSphere Application Server for z/OS to provide JDBC connectivity to an IMS database.

IMS V8 provides a JCA-compliant resource adapter that provides a JDBC interface to an IMS system. The connection to IMS is via the IMS Open Database Access (ODBA) interface.

From an IMS view, WebSphere Application Server for z/OS is an application region that is using the ODBA interface. As with any other application region using ODBA, it must be registered to RRS as a resource manager. RRS provides the two-phase commit support and may act as the syncpoint coordinator.

From a WebSphere Application Server view, the resource adapter is RRSTransactional and must be run as a part of a global transaction. In this scenario, WebSphere Application Server may take the RRS SDSRM role for this transaction, depending on what other type of resource managers are involved. If all resource managers involved are RRSTransactional, then RRS will be the syncpoint manager. If there are XA resource managers involved, then WebSphere Application Server will take the SDSRM role.

For the IMS JDBC connect with WebSphere Application Server for z/OS, WebSphere Application Server and the ODBA-attached IMS system must reside on the same z/OS image, and RRS must be active on that image.

11.2.2 APPC/IMS

APPC/IMS, a part of IMS TM, lets you use the CPI-C (Common Programming Interface for Communications) interface to build CPI application programs. APPC/IMS allows distributed and cooperative processing between IMS and systems that have implemented APPC. APPC/IMS delivers support for APPC with facilities provided with APPC/MVS. APPC/IMS

supports the CPI resource recovery Commit (SRRCMIT) and Backout (SRRBACK) calls for IMS-managed local resources. These protected resources include:

- ▶ IMS TM message-queue messages
- ▶ IMS DB databases
- ▶ DB2 database

APPC/IMS also supports the existing IMS DL/I application programming interface (API) enabling application programs to use LU 6.2 communications without the function of the CPI Communications interface. This allows most existing applications to continue to function with the APPC/IMS support of LU 6.2.

Recommendation

For APPC/IMS, do the following:

- ▶ IMS standard or modified (mixed mode) application programs may be scheduled on remote IMSs through MSC for messages entered from LU 6.2 devices. Be aware that CPI-C driven application programs cannot be transactions that execute on remote systems.
- ▶ Define your APPC/IMS LUs for use by APPC/MVS, as well as by any APPC application program.
- ▶ Use the LTERM and MODNAME in the first segment of your input message for standard and modified (mixed mode) applications if the application logic depends on them. Output can be sent to an APPC device or a non-APPC device through an Alternate Destination PCB. Expect a MODNAME on output to an APPC device, including the origin device, if the application program chooses a different layout for the message.
- ▶ Use a network-qualified LU name. You do not need to have unique names for the LUs on different systems.

IMS-dependent regions are automatically defined to APPC as subordinate address spaces of the IMS Scheduler. An IMS BMP cannot be defined as an ASCH-controlled application. It may use explicit conversation services through the IMS base LU.

IMS manages the APPC/IMS buffers automatically, so no definition is necessary. No special considerations are needed for EMH. For more details, refer to *IMS/ESA V8 Admin Guide: Transaction Manager*, SC26-8014.

The APPC synchronization level defines the protocol that is used when changing conversation states. APPC and IMS supports the following SYNC_LEVEL values.

NONE	Specifies that the programs do not issue calls or recognize returned parameters relating to synchronization.
CONFIRM	Specifies that the programs can perform confirmation processing on the conversation.
SYNCPT	Specifies that the programs participate in coordinated commit processing on resources that are updated during the conversation under the RRS recovery platform. A conversation with this level is also called a <i>protected conversation</i> .

Allocating a conversation with SYNCLEVEL=SYNCPT requires the Resource Recovery Services (RRS) as the sync-point manager (SPM). RRS controls the commitment of protected resources by coordinating the commit or backout request with the participating owners of the updated resources, the resource managers. IMS is the resource manager for DL/I, Fast Path data, and the IMS message queues. The application program decides whether the data is to be committed or aborted and communicates this decision to the SPM. The SPM then coordinates the actions in support of this decision among the resource managers.

IMS application programs can use the IMS *implicit API* to access LU 6.2 devices. This API provides compatibility with non-LU 6.2 device types so that the same application program can be used from both LU 6.2 and non-LU 6.2 devices. The API adds to the APPC interface by supplying IMS-provided processing for the application program. You can use the *explicit CPI Communications interface* for APPC functions and facilities for new or rewritten IMS application programs.

APPC/IMS supports three different types of application programs:

Standard	No explicit use of CPI-C facilities.
Modified	Uses the I/O PCB to communicate with the original input terminal but can use CPIC as well for outbound conversations.
CPI Communications driven	Uses CPI-C calls to receive the incoming message and to send a reply on the same conversation. Uses the DL/I APSB call to allocate a PSB to access IMS databases and alternate PCBs.

For more information about application programming types, refer to *IMS/ESA V8 Administration Guide: Transaction Manager*.

You must set SYNCLVL=SYNCPOINT on each APPC conversation in a distributed transaction. The protection ring is broken if a conversation is allocated with SYNCLVL=NONE or CONFIRM. Whenever IMS commit processing is invoked in a protected conversation environment, IMS passes control to RRS for commit coordination. IMS issues the OS/390 ATRCMIT for implicit mode applications. Explicit mode applications issue the SRRCMIT verb directly.

IMS registers itself to RRS at startup, and thereafter APPC/MVS establishes a private context for each transaction involving a protected conversation. Only valid work units (units of recovery) are of interest to RRS, so only active IMSs register. Neither alternate nor backup IMSs will register until they become the Active IMS.

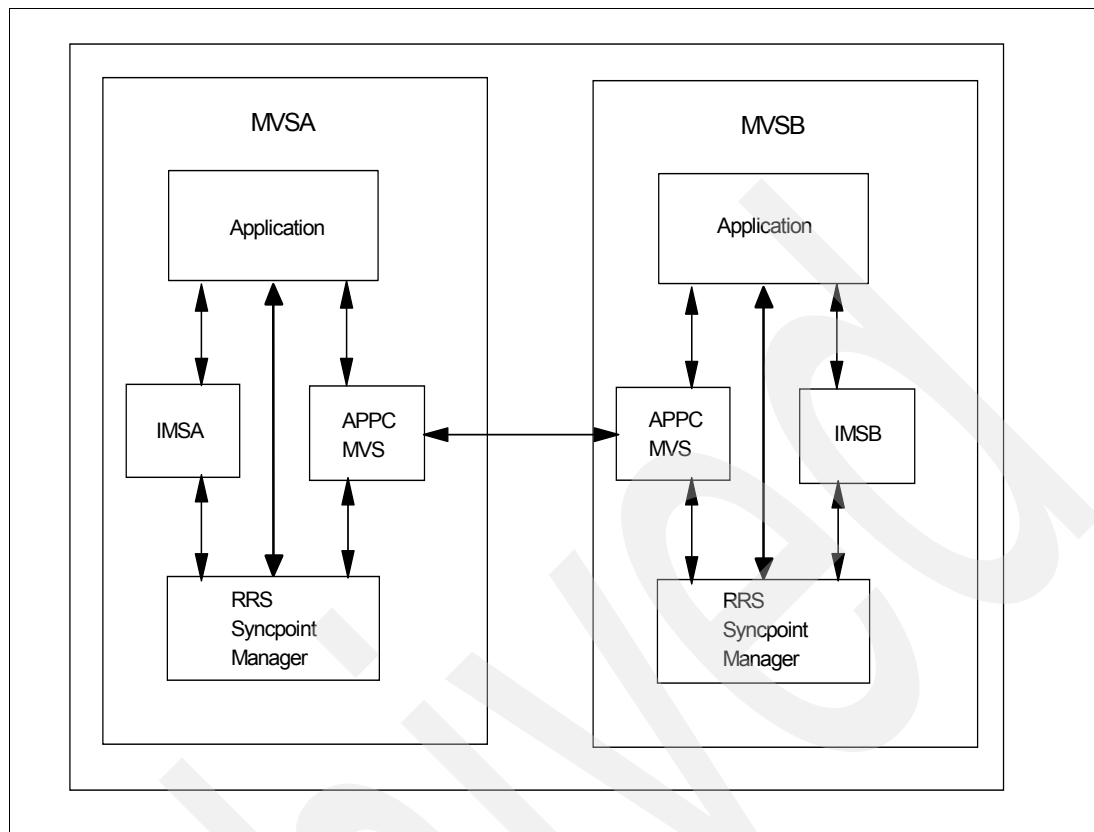


Figure 11-2 APPC/IMS distributed transaction example

Multiple RRSs may be involved in the process where a protected conversation straddles multiple OS/390s, but only one RRS is allowed on each OS/390. APPC/MVS (protected conversations) is responsible for communications between participating commit managers, as illustrated in Figure 11-2. In this instance APPC/MVS is acting as a communications resource manager. Refer to Chapter 5, “RRS operations” on page 45, for a discussion of communication resource managers.

All the resource and commit managers harden their knowledge of the status of each part of the protected conversation so that they can reestablish consistency after any sort of system failure. IMS places the UOR identifier in the x'5611' log record to enable resynchronization with RRS after a restart.

Before you can work with APPC/IMS, you must establish the APPC/IMS support. For detailed information about establishing APPC/IMS support refer to “Establishing APPC/IMS” in *IMS/ESA V8 Admin Guide: Transaction Manager* and *z/OS V1R4 MVS Planning: APPC Management*, SA22-7599.

Example 11-1 IMS VTAM appl

Adding ATNLOSS and SYNCLVL Parameters to the VTAM LU APPL Statements:
To allow APPC/MVS TPs and their partner TPs to establish protected conversations, we added the following to the VTAM LU APPL statements:

```

SCSIM6CA      APPL ACBNAME=SCSIM6CA      C
               APPC=YES,                  C
               PARSESS=YES,                C
               EAS=100,                     C
               ATNLOSS=ALL , <---req. for protected conversations C

```

SYNCLVL=SYNCPT , <---req. for protected conversations	C
SECACPT=NONE,	C
SRBEXIT=YES,	C
DLOGMOD=LU62APPC,	C
MODETAB=SCMODIMS	

APPC/IMS shared queue support

IMS/ESA V8 introduced support for synchronous OTMA and APPC workloads to be distributed and executed on any IMS system in the shared queue group. In “OTMA”, we describe how IMS uses RRS to provide this support.

11.2.3 OTMA

IMS Open Transaction Manager Access (OTMA) is a transaction-based, connectionless client/server protocol. It uses the XCF API and can therefore operate across a sysplex. OTMA is designed to offer high performance client access to IMS without the overhead normally associated with communications protocols (such as SNA).

OTMA provides support for protected conversations (OTMA “send-then-commit” mode with sync_level=synpt, which is also known as Commit Mode 1, or CM1). The two-phase commit support is provided by RRS. In this case, the OTMA client must reside on the same z/OS image as the target IMS system.

OTMA is used by a number of IBM and non-IBM products to access IMS. Examples are IMS Connect V2.1 and transactional RPC. OTMA is documented in *OTMA Guide and Reference*, SC26-8743.

OTMA Client Attach

While all that is necessary for most applications programming involving RRS is to issue the SRRCMIT or SRRBACK calls, with OTMA you are required to delve deeper.

If you wish your application to update protected resources in other resource managers as well as IMS within one unit or work, then the OTMA client must register with RRS as a resource manager and must acquire a private context, which it then includes on the OTMA call to IMS.

The MPR that IMS schedules will then run using the supplied private context, and RRS context services can tie in any work performed within the unit of work. The OTMA client application and IMS will both be resource managers, from an RRS view. When using RRS with OTMA, the client must reside on the same OS/390 image as the target IMS system. This is an RRS restriction.

OTMA shared queue support

IMS V8 introduced support for synchronous OTMA and APPC workloads to be distributed and executed on any IMS system in the shared queue group. Prior to IMS/ESA V8, only asynchronous OTMA messages could be processed on any system in the shared queue group. That meant that only OTMA “commit-then-send” mode was supported, and it was not possible to have protected conversation. The support added in IMS/ESA V8 allows OTMA to use protected conversations (OTMA “send-then-commit” mode with sync_level=synpt) with support for shared queues.

IMS V8 uses RRS multisystem cascaded transactions support provided in z/OS V1.2 to achieve this. Refer to “Distributed RRS” on page 23 for a discussion of RRS cascaded transaction support.

Synchronous Support General Flow

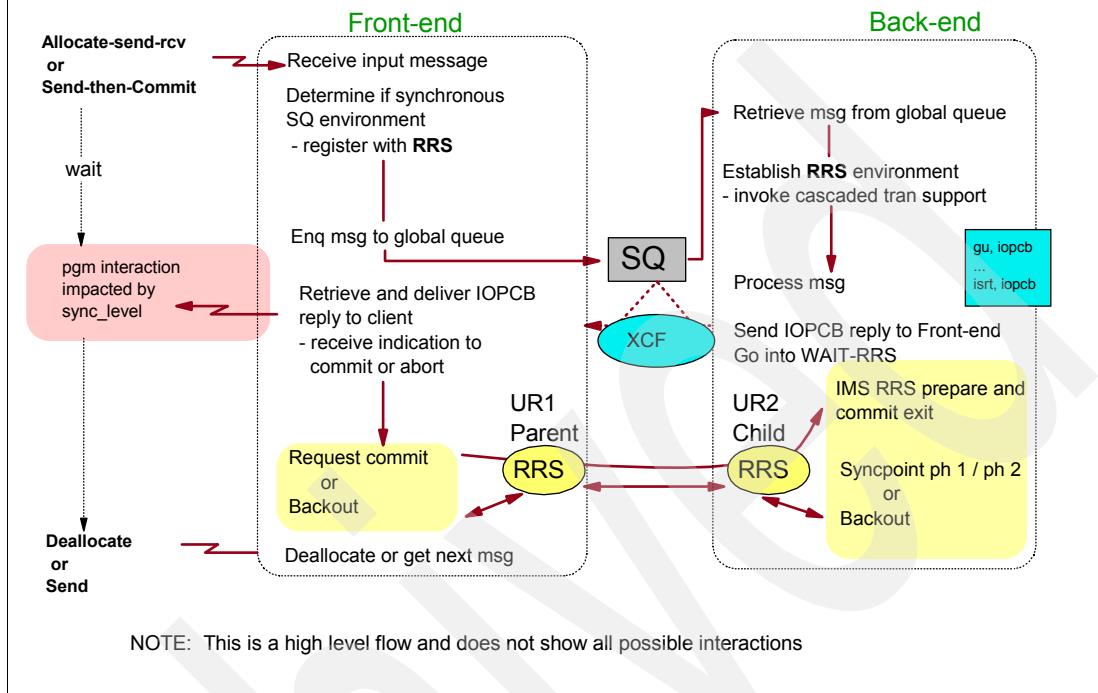


Figure 11-3 OTMA/APPC synchronous shared queue support

Figure 11-3 shows the general flow for an OTMA or APPC protected conversation with shared queue support. RRS cascaded transaction support links the unit of recovery (UR) on the front-end IMS system (UR1, the parent UR in the cascaded transaction) with the UR on the back-end IMS system (UR2 the child UR in the cascaded transaction) and ensures RRS can manage the two URs within one transaction.

IMS has the following requirements for synchronous OTMA/APPC shared queue support:

- ▶ ALL IMS systems in the shared queue group are at IMS V8 or above
- ▶ All IMS control regions are started with RRS=Y
- ▶ RRS is enabled on all z/OS systems where the IMS systems run
- ▶ All z/OS systems are at z/OS V1.2 (plus APAR OW50627)

IMS Connect V2.1

IMS Connect provides a bridge to IMS from TCP/IP attached or local clients using the IMS Java connector. IMS Connect uses the OTMA interface to attach to IMS. IMS Connect will always use RRS because IMS OTMA attach requires that the client (IMS Connect) be an RRS resource manager. The behavior of IMS Connect depends on whether TCP/IP attach or local attach is used; in the following sections, we examine both scenarios.

TCP/IP attach to IMS Connect

The resource adapter for WebSphere Application Server for z/OS supplied by IMS Connect is XA-compliant when operating in TCP/IP attach mode. This means that two-phase commit is supported from WebSphere Application Server transactions to a TCP/IP connected IMS Connect server.

In this environment, IMS Connect will always take on an RRS SDSRM role. This allows IMS Connect to act as a communications resource manager (CRM) and interface to other external transaction managers using XA protocol.

In this environment, the IMS Connect server and the OTMA-attached IMS system must reside on the same z/O image, and RRS must be active on that image. The WebSphere Application Server can be on a remote z/OS image, although it is possible to configure the resource adapter to attach in remote mode even if the WebSphere Application Server resides on the same z/OS images as IMS Connect and the IMS system. However, this is not a recommended configuration. Local attach is more efficient and provides for better restart and recovery because RRS will be used.

WebSphere Application Server local attach to IMS Connect

The resource adapter for WebSphere Application Server for z/OS supplied by IMS Connect is RRSTransactional when operating in local attach mode. This means that two-phase commit is supported from WebSphere Application Server transactions to a locally connected IMS Connect server.

In local mode, the resource adapter communicates to IMS Connect using the MVS Program Call (PC) facility. In this environment IMS Connect does not take on an RRS SDSRM role because it does not need to communicate with an external transaction manager using XA protocol. In this scenario WebSphere Application Server may take the RRS SDSRM role for this transaction, depending on what other type of resource managers are involved. If all resource managers involved are RRSTransactional, then RRS will be the syncpoint coordinator.

In the local attach environment, WebSphere Application Server, the IMS Connect server, and the OTMA-attached IMS system must reside on the same z/OS image and RRS must be active on that image.

11.3 IMS/ESA restart and recovery with RRS

On restart IMS, like any other DBMS, performs extensive restart and recovery processing. For detailed information about IMS restart and recovery processing, refer to *IMS/ESA V8 Admin Guide: Transaction Manager*, SC26-8014. Here, we concentrate on how IMS interacts with RRS on restart and recovery, and we examine scenarios that may force manual intervention to resolve in-doubt URs with RRS.

For an IMS warm or emergency restart, IMS will process its Online Log Data Sets (OLDS) and perform log analysis to determine whether there are any units of work that were in-flight, in-commit, in-abort, or in-doubt. IMS will then proceed to commit or back out these units of work as appropriate. For any in-doubt URs that represent RRS protected conversations, IMS must resync with RRS.

As long as RRS=Y is specified in the IMS Control region parameters, during its restart processing IMS will resync with RRS and retrieve any outstanding URs from RRS. IMS calls the `Begin_restart_processing (ATRIBRS)` RRS service to begin restart processing with RRS. RRS will return a list of URs in which IMS has expressed an interest. IMS then retrieves information for each outstanding UR using the `Retrieve_interest_data (ATRRID)` RRS call.

On completion of the RRS resync, you will see the message shown in Example 11-2.

Example 11-2 RRS connection established message

```
DFS0653I PROTECTED CONVERSATION PROCESSING WITH RRS/MVS ENABLED
```

You will see the messages shown in Example 11-3, indicating in-doubt URs and when they are resolved.

Example 11-3 In-doubt UR message sample

```
DFS0693I RIS ESTABLISHED FOR PSB xxxxxx,PRTKN=yyyyyy, TOKEN=zzzzzzz,
RRS-URID=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

DFS0699I RESYNC (COMMIT|ABORT) COMPLETE for PSB xxxxxx, PRTKN=yyyyyy,
TOKEN=zzzzzzz, RRS-URID=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

You can use the /DIS UOR INDOUBT command to display any outstanding URs that are in-doubt.

11.3.1 RRS failure while IMS is active

If RRS fails while IMS is running, then all attempts to use RRS protected conversations will fail. Application abendU0711 will occur for any application that attempts to use RRS protected conversation. IMS will issue the message shown in Example 11-4.

Example 11-4 IMS error message sample for RRS failure

```
DFS0698W PROTECTED CONVERSATION PROCESSING NOT ENABLED - reason
```

The message will provide a reason for the failure, such as RRS NOT AVAILABLE. After RRS is restarted, IMS will resync with RRS and enable protected conversations again.

11.3.2 IMS restart when RRS is not available

If RRS=Y is specified in the IMS control region parameters and RRS is not available, you will receive the message shown in Example 11-5.

Example 11-5 RRS not active message sample

```
DFS0548A RRS NOT ACTIVE BUT RRS=Y SPECIFIED - REPLY: RETRY,
CONTINUE OR CANCEL
```

You can reply CONTINUE, but RRS protected conversation support will not be enabled until RRS is started on that system. Any attempt by an application to use RRS protected resources will result in ABENDU0711.

Also, IMS will not be able to resolve any in-doubt URs where RRS was the syncpoint coordinator. These URs will remain in-doubt until RRS is started and IMS can resync with RRS.

The IMS operator command /DIS UOR INDOUBT will display any in-doubt RRS URs.

Example 11-6 /DIS UOR INDOUBT sample output

```
/DIS UOR INDOUBT

ST P-TOKEN PSBNAME RRS-URID IMS-TOKEN
RI 00010120 PLAPJK02 12345678901234567890123456789012 SYS1 00000013000000001
```

The status RI, as shown in Example 11-6, indicates a residual in-doubt that is an in-doubt from a prior IMS execution or a dependent region abend. The display shows both the RRS URID as seen in the RRS ISPF panels and RRS logs, and the IMS recovery token.

11.3.3 IMS restart when RRS has been cold-started

If RRS=Y is specified in the IMS control region parameters and RRS has been cold-started since IMS was last active, then when IMS contacts RRS to retrieve any outstanding URs, it will get none returned. IMS may find units of recovery in its own logs that it expects to be able to match with a UR returned from RRS. If this happens, IMS issues message DFS0744A, as shown in Example 11-7.

Example 11-7 IMS restart messages for in-doubt UR

```
DFS0744A IMS HOLDS AN INDOUBT UOR FOR WHICH RRS HAS NO DATA:
URID-xxxxxxxxxx, TOKEN=yyyyyyyyyy
```

In this case, manual recovery is necessary. Use the IMS command /CHANGE UOR ABORT/COMMIT to either back out or commit an in-doubt UR.

There is another, related scenario where IMS is restarted on a system and RRS has been started as a part of a different logging group (GNAME parameter on RRS proc). In this instance IMS will detect that the RRS it is connecting to is not in the same RRS logging group as the RRS it last connected to. IMS treats this situation in the same way as an RRS cold start. IMS will perform log processing and if there are any in-doubt URs that IMS has a record of that require RRS to determine their outcome, IMS issues message DFS0744A and manual intervention will be required.

11.3.4 IMS restart on a different system

As discussed in Chapter 4, “Implementing RRS” on page 31, prior to z/OS V1.6, RRS will mark a resource manager as having to restart on the same system if there are outstanding URs for that resource manager when it terminated and RRS has remained operational on that system. If IMS terminates and there are outstanding URs, then RRS will reject any attempt by IMs to restart on another system in the same RRS logging group.

IMS will receive a return code of F02 from the Begin_restart_processing (ATRIBRS) RRS call indicating that IMS cannot register with RRS on this system. If there are in-doubt URs, then IMS will issue message DFSxxxx to indicate that RRS cannot be contacted to resolve these in-doubt URs because IMS is being restarted on the wrong system.

This has the same effect as if RRS is unavailable; in-doubt URs cannot be resolved and RRSF and WLM stored procedures will be unavailable. The difference with this scenario is that, in order to allow IMS to connect to RRS, IMS must be stopped and either restarted on the z/OS system it was originally running on or you must manually remove any outstanding URs from RRS in which IMS has an interest. Then IMS can be restarted on the new system.

Note that this issue only occurs on releases of z/OS prior to V1.6 and only when RRS has stayed operational on the original system. It is assumed, if IMS terminates and the z/OS system remains operational, that IMS will be restarted on the same system.

11.4 IMS/ESA sample scenario using RRS

For this test we configured the CICS and IMS resource adapters to use RRSTransactional support. The WebSphere Application Server application updated both CICS and IMS within the scope of one transaction.

Example 11-8 is an extract from the RRS Archive log showing entries related to a successful transaction.

Example 11-8 RRS Archive log sample

```
SC48      2003/12/02 10:49:06.050724 BLOCKID=00000000412FACD1
          URID=BA6919507E8F0000000001F1010F0000 JOBNAME=WSA11S  USERID=ASSR1
          PARENT URID=00000000000000000000000000000000
          SURID=N/A
          WORK MANAGER NAME=BB0.CLHA1.CLUA11.WSA11.IBM
          SYNCPOINT=Commit RETURN CODE=00000000
          START=2003/12/02 15:49:06.038266 COMPLETE=2003/12/02 15:49:06.050490
          EXITFLAGS=00840000
          LUWID=USIBMSC.SCSCERW1 691951BA88FB 0001 TID=          GTID=

          FORMATID=003284271494 (decimal) C3C20186 (hexadecimal)
          GTRID=
          BA691950551DB8230000003600000009BA1DDA49113244A40000018000000003090C060B
          BQUAL=
          BA691950551DB8230000003600000009BA1DDA49113244A40000018000000003090C060B000000001
          RMNAME=IMS.IM4B____V081.STL.SANJOSE.IBM ROLE=Participant
          FLAGS=10021000 PROTOCOL=PresumeAbort
          StateCheck EXIT RC=Uncalled
          Prepare EXIT RC=00000000
          DistSp EXIT RC=Uncalled
          Commit EXIT RC=00000000
          Backout EXIT RC=Uncalled
          EndUr EXIT RC=Uncalled
          ExitFailed EXIT RC=00000000
          Completion EXIT RC=Uncalled
          OnlyAgent EXIT RC=Uncalled
          RMNAME=DFHRXDM.SCSCERW1.IBM ROLE=Participant
          FLAGS=10001000 PROTOCOL=PresumeAbort
          StateCheck EXIT RC=Uncalled
          Prepare EXIT RC=00000000
          DistSp EXIT RC=Uncalled
          Commit EXIT RC=00000010
          Backout EXIT RC=Uncalled
          EndUr EXIT RC=Uncalled
          ExitFailed EXIT RC=00000010
          Completion EXIT RC=Uncalled
          OnlyAgent EXIT RC=Uncalled
```

In this example we see that WebSphere Application Server, which is the work manager from an RRS point of view, has issued a commit. RRS is the syncpoint coordinator and there are two resource managers: IMS.IM4B____V081.STL.SANJOSE.IBM (our IMS system) and DFHRXDM.SCSCERW1.IBM (our CICS system). WebSphere Application Server, in this case, does not take an SDSRM role because in our transaction there are only RRSTransactional RMs involved.

For each resource manager we see that the Prepare, Commit and ExitFailed exits are driven by RRS. The CICS commit exit returns x'10' which signifies ATRX_FORGET. This means CICS has completed commit processing and requests RRS to delete its interest in this UR.

Archived

Archived



WebSphere MQ for z/OS

In this chapter, we describe how WebSphere MQSeries V5.3 for z/OS uses RRS.

This chapter covers the following topics:

- ▶ WebSphere MQ V5.3 RRS requirements
- ▶ WebSphere MQ features that exploit RRS
- ▶ RRS facilities that are exploited
- ▶ Restart and recovery issues

12.1 WebSphere MQ RRS requirements

WebSphere MQ provides two RRS adapters that allow batch applications that connect to WebSphere MQ, as well as to DB2 stored procedures, to use RRS coordinated commit processing. The adapters allow WebSphere MQ to be a full participant in RRS coordination of two-phase commit syncpoints.

The WebSphere MQ RRS adapters can be used in the same way as the Batch/TSO adapter. The WebSphere MQ RRS adapters support simultaneous connections to multiple WebSphere MQ queue managers running on a single z/OS instance from a single task, and they support the ability to switch a WebSphere MQ batch thread between TCBs.

The WebSphere MQ provided RRS adapters are:

- | | |
|-----------------|--|
| CSQBRSTB | This stub allows you to use two-phase commit and backout for applications using RRS callable resource recovery services instead of the MQI calls MQCMIT and MQBACK. CSQBRSTB requires you to use SRRCMIT and SRRBACK. If your program still uses MQCMIT or MQBACK, when linked with CSQBRSTB, you will receive MQRC_ENVIRONMENT_ERROR return code. |
| CSQBRSSI | This stub allows you to use either MQI calls MQCMIT and MQBACK or SRRCMIT and SRRBACK. If you use MQCMIT and MQBACK, WebSphere MQ actually implements these calls as the SRRCMIT and SRRBACK RRS calls. |

The bind step for an application that wants to use RRS *must* include one of these stubs. There is no dynamic support for these stubs. Both stubs are shipped with linkage attributes AMODE(31) and RMODE(ANY).

12.1.1 WebSphere MQ and DB2 stored procedures

If you use DB2 stored procedures with RRS, you should be aware of the following:

- ▶ DB2 stored procedures that use RRS must run under the control of WLM.
- ▶ If a DB2 stored procedure using WebSphere MQ calls is linked with either RRS stub, the MQCONN or MQCONNEX call returns MQRC_ENVIRONMENT_ERROR.
- ▶ If a WLM-managed DB2 stored procedure contains WebSphere MQ calls and is linked with a non-RRS stub, the MQCONN or MQCONNEX call returns MQRC_ENVIRONMENT_ERROR.
- ▶ If your DB2 stored procedure contains WebSphere MQ calls and is linked with a non-RRS stub, WebSphere MQ resources updated in that stored procedure are not committed until the stored procedure address space ends, or until a subsequent stored procedure does an MQCMIT.
- ▶ Multiple copies of the same DB2 stored procedure can execute concurrently in the same address space. You should ensure that your program is coded in a re-entrant manner if you want DB2 to use a single copy of your stored procedure.
- ▶ You must not code MQCMIT and MQBACK in a WLM-managed DB2 stored procedure.
- ▶ All programs must be designed to run in Language Environment® (LE).

12.1.2 WebSphere MQ JMS interface

The WebSphere MQSeries classes for Java provide a JMS-compliant interface for Java programs to invoke messaging services using MQSeries. On z/OS, this interface uses the MQ

RRS adapter, which provides two-phase commit between the Java application and MQSeries. The Java application must use MQ bindings mode; client connection is not supported from a Java program on z/OS. The Java program and the MQ queue manager must reside on the same z/OS image.

The WebSphere MQ JMS driver is used by the WebSphere Application Server to provide the JMS interface to an external messaging system from an EJB application.

12.2 WebSphere MQ restart and recovery issue with RRS

On restart, MQ performs restart and recovery processing. MQ restart looks very similar to DB2 restart. There are four phases:

- Phase 1: Log initialization

MQ identifies the last LOG RBA used before termination so that it can start logging at the next RBA.

- Phase 2: Current status rebuild

MQ determines what URs are outstanding and the status of each UR (in-flight, in-commit, in-abort or in-doubt).

- Phase 3: Forward log recovery

Having determined the outstanding URs in Phase 2 MQ makes all the database changes for committed work as well as for in-flight, in-doubt and in-abort URs. For in-flight, in-doubt and in-abort URs, MQ locks the changed data to make it unavailable.

- Phase 4: Backward log recovery

In this phase, MQ reverses changes by in-flight or in-abort URs and releases locks for those URs.

During Phase 2 of MQ restart processing, MQ will retrieve any outstanding URs from RRS. MQ calls the `Begin_restart_processing (ATRIBRS)` RRS service to begin restart processing with RRS. RRS will return a list of URs in which MQ has expressed an interest.

12.2.1 RRS failure when MQ is running

If RRS fails while MQ is operational, any MQ facilities that use RRS also fail. That means that any applications using the RRS adapter fail. MQ stays operational, but you may see various application failures indicated by RC2012 2012 MQRC_ENVIRONMENT_ERROR because RRS is unavailable.

MQ Queue sharing uses the DB2 RRSAF facility to connect to DB2, so a failure of RRS will remove an MQ queue manager from an MQ queue sharing group. The following message is issued:

```
CSQ5026E -MQV1 CSQ5CONN Unable to access DB2, RRS is not available
```

If RRS is restarted, then MQ reconnects to RRS and applications are able again to connect to MQ using the RRS adapter. In an MQ queue sharing group, you can see the following message when MQ reconnects to DB2:

```
CSQ5001I -MQV1 CSQ5CONN Connected to DB2 D7V1
```

12.2.2 WebSphere MQ restart if RRS is unavailable

If MQ restarts and RRS is not available, then it cannot resolve any URs that were in-doubt where RRS was the syncpoint coordinator and MQ was a participant. For any URs in which the state was in-flight, in-commit or in-abort, MQ does not need to retrieve UR data from RRS. It can process those UR by backing out any inflight or in-abort URs.

Any in-doubt URs may result in retained locks for the affected data. If there are in-doubt URs, then MQ will issue messages CSQ3011I - CSQ3016I to indicate that RRS cannot be contacted to resolve these in-doubt URs.

MQ will resync with RRS once RRS is restarted on the z/OS image. At that stage it can resolve any in-doubt URs according to the final state that RRS supplies.

Note that when MQ starts and RRS is unavailable, any MQ facilities that require RRS are not available. That means any application that attempts to use RRS adapter will get an error. Also, if the MQ subsystem is part of a queue sharing group, then MQ will hang on startup until it can establish its connection to DB2. MQ queue sharing needs to connect to the DB2 system using RRSFAC, and this will not be possible until RRS is started on the system.

MQ issues an error message if it receives an error code from any RRS call made during startup:

```
CSQ3017I csect-name RRS function <call-name> failed, RC=rc
```

The support for this message is provided with APAR PQ67919 for MQ V5.2 and later.

12.2.3 WebSphere MQ restart on another system

As discussed in Chapter 4, “Implementing RRS” on page 31, prior to z/OS V1.6, RRS marks a resource manager as having to restart on the same system if there are outstanding URs for that resource manager when it terminated, and RRS has remained operational on that system. If MQ terminates and there are outstanding URs, then RRS rejects any attempt by MQ to restart on another system in the same RRS logging group. MQ will receive a return code of F02 from the Begin_restart_processing (ATRIBRS) RRS call, indicating that MQ cannot register with RRS on this system.

MQ issues message CSQ3017I to indicate an RRS failure:

```
CSQ3017I -MQV1 CSQ3RRSR RRS function ATRIBRS failed, RC=00000F02
```

If there are in-doubt URs, then DB2 will issue messages CSQ3011I - CSQ3016I to indicate that RRS cannot be contacted to resolve these in-doubt URs because MQ is being restarted on the wrong system.

This has the same effect as if RRS is unavailable; in-doubt URs cannot be resolved and RRSFAC and WLM stored procedures will be unavailable. If the MQ subsystem is a part of a Queue sharing group, then MQ will not start until it can connect to its DB2—and this cannot happen because MQ cannot use the RRSFAC facility to speak to DB2.

The difference in this scenario is that in order to allow MQ to connect to RRS, MQ must be stopped and either restarted on the z/OS system it was originally running on, or you must manually remove any outstanding URs in which MQ has an interest so that RRS will mark MQ as “restart anywhere”. Then MQ can be restarted on the new system.

Note that this issue only occurs on releases of z/OS prior to V1.6 - and only when RRS has stayed operational on the original system. It is assumed that if DB2 terminates and the z/OS system remains operational, MQ will be restarted on the same system.

12.3 Sample scenarios for WebSphere MQ using RRS

Our test scenario consisted of an MVS batch program written in C that updates both DB2 and MQ in one unit of work. The program uses RRSATF to attach to DB2 and uses the MQ RRS attach support to connect to MQSeries. The program uses the RRS SRRCMIT and SRRBACK calls to perform commit and backout processing.

12.3.1 Normal commit processing scenario

In this scenario we see RRS panel displays and RRS log extracts showing what happens during a normal run of the batch job, where we update both DB2 and MQ and then commit the updates. The C program we are running waits for a period after updating DB2 and MQ, but before issuing a SRRCMIT. This gives us time to display panels and so on.

Before we start the batch job, we look at the status of the DB2 (subsystem D7V1) and MQ (subsystem MQV1) resource managers by looking at the RRS ISPF Resource Manager List panel, shown in Example 12-1.

Example 12-1 RRS Resource Manager List panel

```
RRS Resource Manager List          Row 3 to 13
Command ==>                        Scroll ==>

Commands: v-View Details u-View URs r-Remove Interest

S  RM Name                                State      System  Logging G
  CSQ.RRSATF.IBM.MQV1                     Run        SC53    WTSCPLX1
  DSN.RRSATF.IBM.DB7A                      Run        SC53    WTSCPLX1
  DSN.RRSATF.IBM.DB7J                      Run        SC53    WTSCPLX1
  DSN.RRSATF.IBM.DB7L                      Run        SC53    WTSCPLX1
  DSN.RRSATF.IBM.D7V1                      Run        SC53    WTSCPLX1
  DSN.RRSPAS.IBM.DB7A                      Run        SC53    WTSCPLX1
  DSN.RRSPAS.IBM.DB7J                      Run        SC53    WTSCPLX1
  DSN.RRSPAS.IBM.DB7L                      Run        SC53    WTSCPLX1
  DSN.RRSPAS.IBM.D7V1                      Run        SC53    WTSCPLX1
  IMS.IMSC___V081.STL.SANJOSE.IBM Reset      SC53    WTSCPLX1
  IMS.IMSG___V091.STL.SANJOSE.IBM Reset      SC53    WTSCPLX1
```

This panel shows the two resource managers that we are interested in: DSN.RRSATF.IBM.D7V1 is the DB2 RRSATF resource manager, and CSQ.RRSATF.IBM.MQV1 is the MQseries RRS attach resource manager. Both RMs have an RRS state of Run, meaning they are registered to RRS on this system (SC53) and have completed RRS restart processing.

We now start our batch job and display the RRS UR details using the RRS ISPF panel, as shown in Example 12-2.

Example 12-2 RRS UR Details view

```
RRS Unit of Recovery List          Row 1 to 1 of 1
RRS Unit of Recovery Details       Row 1 to 2
Command ==>                        Scroll ==> P

Commands r-Remove Interest v-View URI Details

UR identifier : BB67A49B7E5CEDD00000001001010000
```

```

Create time : 2004/06/22 02:44:42.478089      Comments :
UR state : InFlight      UR type : Prot
System : SC53      Logging Group : WTSCPLX1
SURID : N/A
Work Manager Name : SC53.MURPHYAR.0026
  Display Work IDs      Display IDs formatted
  Luwid . . : Not Present
  Eid . . : Not Present
  Xid . . : Not Present
Expressions of Interest:
S  RM Name      Type  Role
   CSQ.RRSATF.IBM.MQV1      Prot  Participant
   DSN.RRSATF.IBM.D7V1      Prot  Participant

```

This panel shows the UR details. Both RMs have an RRS role of Participant, meaning that RRS will act as syncpoint coordinator.

When the batch program finally commits the updates, the UR moves to a state of Forgotten. We can view the RRS Archive log, as shown in Example 12-3, to see a record of the commit.

Example 12-3 RRS Archive log extract

```

RRS/MVS LOG STREAM BROWSE DETAIL REPORT

READING ATR.WTSCPLX1.ARCHIVE      LOG STREAM

SC53      2004/06/21 22:46:42.504857 BLOCKID=0000000079876D89
URID=BB67A49B7E5CEDD00000001001010000 JOBNAME=MURPHYAR USERID=MURPHYA
PARENT URID=00000000000000000000000000000000
SURID=N/A
WORK MANAGER NAME=SC53.MURPHYAR.0026
SYNCPPOINT=Commit RETURN CODE=00000000
START=2004/06/22 02:46:42.484323 COMPLETE=2004/06/22 02:46:42.504683
EXITFLAGS=00800000
LUWID=      TID=      GTID=

FORMATID=      (decimal)      (hexadecimal)
GTRID=

BQUAL=

RMNAME=CSQ.RRSATF.IBM.MQV1      ROLE=Participant
FLAGS=10001000 PROTOCOL=PresumeAbort
StateCheck EXIT RC=Uncalled
Prepare EXIT RC=00000000
DistSp EXIT RC=Uncalled
Commit EXIT RC=00000010
Backout EXIT RC=Uncalled
EndUr EXIT RC=Uncalled
ExitFailed EXIT RC=Uncalled
Completion EXIT RC=Uncalled
OnlyAgent EXIT RC=Uncalled
RMNAME=DSN.RRSATF.IBM.D7V1      ROLE=Participant
FLAGS=10001000 PROTOCOL=PresumeAbort
StateCheck EXIT RC=Uncalled
Prepare EXIT RC=00000000
DistSp EXIT RC=Uncalled
Commit EXIT RC=00000010
Backout EXIT RC=Uncalled

```

```
EndUr      EXIT RC=Uncalled  
ExitFailed EXIT RC=Uncalled  
Completion EXIT RC=Uncalled  
OnlyAgent EXIT RC=Uncalled
```

Example 12-2 shows that the UR has been successfully committed. RRS has called the Prepare and Commit exits for both MQ and DB2. The return code of x'10' from the Commit exit signifies ATRX_FORGET, the commit has been successful and the RM has requested RRS to set the UR state to Forgotten.

Archived

APPC/MVS

In this chapter, we discuss how APPC/MVS in z/OS V1.4 uses RRS.

This chapter covers the following topics:

- ▶ “APPC/MVS RRS requirements” on page 138
- ▶ “APPC/MVS application restart and recovery with RRS” on page 139
- ▶ “APPC/MVS sample scenario with RRS” on page 140
- ▶ Restart and recovery issues

13.1 APPC/MVS RRS requirements

Advanced Program-to-Program Communication (APPC) is an implementation of the Systems Network Architecture (SNA) LU 6.2 protocol on a given system. APPC allows program-to-program communication within or between systems using SNA as the underlying communication protocol.

APPC/MVS is the IBM implementation of APPC on the z/OS platform. APPC/MVS uses the underlying LU 6.2 support provided by VTAM (the IBM implementation of SNA on z/OS), and extends it to provide full APPC support. In particular, it provides the transactional services discussed in the opening section of this chapter.

Note: Do not confuse VTAM/APPC and APPC/MVS.

- ▶ VTAM provides APPC support that may be used directly by programs on z/OS without any involvement by APPC/MVS. For example, CICS has supported APPC-attached clients since early releases using the LU 6.2 support in VTAM. CICS does not use APPC/MVS.
- ▶ APPC/MVS is a resource manager that manages conversations between transaction programs using SNA as the underlying communications protocol. APPC/MVS is used by a number of products on z/OS to provide APPC support. IMS/TM uses the APPC/MVS to provide APPC support for clients connections. WebSphere MQ also uses APPC/MVS to provide support for APPC connections.

There are three levels of commit protocol defined in the LU 6.2 architecture and implemented in APPC/MVS:

SYNCLEVEL=NONE There is no coordination between the partners through automatic APPC flows.

SYNCLEVEL=CONFIRM DEALLOC causes confirmation-of-receipt flows.

SYNCLEVEL=SYNCPT This provides protected conversations. A *protected conversation* links separate pieces of a distributed application into a single transaction. All resource managers participating in the protected conversation either commit or back out together. This support requires RRS to be active.

We are interested in protected conversations. To identify a conversation as protected, the transaction programs (TP) allocate the conversations with a synchronization level of syncpt. When one of the TPs is ready to commit or back out its changes for a particular unit of work, the TP issues either the SRRCMIT or SRRBACK callable service to begin a syncpoint operation. During this operation, the local and partner LUs work with system syncpoint managers to coordinate the changes; RRS is the system syncpoint manager for APPC/MVS LUs.

As discussed in 2.1, “Introduction to two-phase commit” on page 10, RRS is an exit manager. It performs its functions by driving exits provided by a resource manager in response to certain events. When APPC/MVS registers LUs as resource managers, it sets its exits to communicate with partner LUs to pass on PREPARE, COMMIT, or ROLLBACK requests. APPC/MVS on the partner LU will communicate with RRS on the local system to instruct it to commit or roll back units of work.

In these scenarios, APPC/MVS can act as an RRS communications resource manager. APPC/MVS will take on an RRS distributed server resource manager (DSRM) role to allow it act as a part of a distributed two-phase commit. Refer to Chapter 3, “Distributed RRS” on page 23, for a discussion of RRS distributed two-phase commit support.

13.1.1 Transaction flow using APPC/MVS protected conversations

An example of transaction flow is provided in the discussion of the peer-to-peer CRM model in 3.1.1, “RRS distributed syncpoint support” on page 25.

13.1.2 APPC/MVS system requirements for protected conversations

RRS must be enabled on all z/OS systems that are participating in the distributed transaction. Refer to Chapter 4, “Implementing RRS” on page 31, for details on implementing RRS.

APPC/MVS requires the partner LU to be parallel session-capable. The APPC/MVS LU must have its VTAM APPL definition with the SYNCLVL parameter set to SYNCPT, and the ATNLOSS parameter set to ALL.

APPC/MVS requires its own system logger log stream named ATBAPPC.LU.LOGNAMES. This log is used to hold the names of local and partner LUs and the negotiated syncpoint capabilities for all protected conversations. This information must be available to allow for restart and recovery processing after a failure. APPC/MVS supports DASD-only log streams, but in a Parallel Sysplex environment, the APPC/MVS log stream should be defined in the Coupling Facility if more than one system processes protected conversations.

For a detailed discussion about this topic, see “Using APPC/MVS Protected Conversations Support” in *OS/390 MVS Planning: APPC/MVS Management*, GC28-1807.

13.1.3 Managing APPC/MVS resources for protected conversations

A number of new APPC/MVS commands allow you to display information about protected conversations:

- ▶ The **DISPLAY APPC,LU,ALL** command contains the resource manager name for LUs that are registered with RRS, and indicates whether the LU is syncpoint-capable.
- ▶ The **DISPLAY APPC,TP,ALL** command contains the logical work unit identifier and indicates whether the conversation is protected and whether a syncpoint operation is in progress.
- ▶ The **DISPLAY APPC,UR** command displays each unit of recovery associated with a protected conversation. The command response contains the RRS URID that can be used in the RRS ISPF panels.

RRS also provides ISPF panels that allow you to display information about units of work. An APPC/MVS LU registered as a resource manager has a name in the form of *ATB.netid.luname.IBM*. You can use the RRS panels to display information for resource managers beginning with ATB* to display all APPC/MVS resource managers. Refer to Chapter 5, “RRS operations” on page 45 for examples of using the RRS ISPF panels to display information.

13.2 APPC/MVS application restart and recovery with RRS

On restart, APPC/MVS will process its logstream to determine all protected conversations that were ongoing before it terminated. It will then resync with RRS to determine the state of any URs in which it has expressed an interest.

During its restart processing, an APPC/MVS LU will resync with RRS and retrieve any outstanding URs from RRS. APPC/MVS calls the *Begin_restart_processing* (ATRIBRS) RRS service to begin restart processing with RRS. RRS will return a list of URs in which

APPC/MVS has expressed an interest. APPC/MVS then retrieves information for each outstanding UR using the Retrieve_interest_data (ATRRID) RRS call.

Example 13-2 shows the APPC/MVS message that is issued for every LU that provides protected conversation support:

Example 13-1 Sample APPC message issued for each Protected Conversation LU

```
AATB227I LOCAL LU luname IS log-status AS A RESOURCE MANAGER WITH RRS/MVS. LOCAL LOG:
logname
```

In this message, *log-status* will be either COLD STARTING or WARM STARTING.

During the resynchronization phase, APPC/MVS will issue a number of messages indicating its process and any errors that have occurred. These messages all begin with ATB2*.

13.2.1 RRS failure while the APPC/MVS application is active

If RRS fails while an APPC/MVS application that uses protected conversations is active, then that LU will remain active but be unable to process any protected conversations. Any attempt to establish a new conversation will fail.

Example 13-2 shows the APPC/MVS error message ATB206I issued to indicate the LU cannot process protected conversations because RRS exits have been unset.

Example 13-2 APPC message after RRS failure

```
AATB208I LOGICAL UNIT luname FOR TRANSACTION SCHEDULER schedname WILL REJECT ALL
PROTECTED CONVERSATIONS. THE RESOURCE MANAGER EXITS HAVE BEEN UNSET. NOTIFICATION EXIT
REASON=rsncode.
```

Example 13-3 shows that RRS is available again; the APPC/MVS LU should be able to process protected conversations.

Example 13-3 APPC message after RRS becomes available

```
ATB201I LOGICAL UNIT luname FOR TRANSACTION SCHEDULER schedname NOW ACCEPTS PROTECTED
CONVERSATIONS.
```

13.3 APPC/MVS sample scenario with RRS

Example 13-4 shows an extract from the RRS Archive log where we have an APPC/MVS application updated DB2 over a protected conversation.

Example 13-4 Extract from RRS Archive log

```
BROWSE      JOHNBTZ.ATR.REPORT                                Write failed with abend
Command ==>                                                    Scroll ==> CSR
***** Top of Data *****
RRS/MVS logstream BROWSE DETAIL  REPORT

READING ATR.APPC.ARCHIVE          logstream

N9A      2003/08/07 00:11:37.787861 BLOCKID=0000000000000001
URID=B0DD95907E9A2C780000005601010000 JOBNAME=SP303  USERID=*
SYNCPPOINT=Commit RETURN CODE=00000000
START=2003/08/07 04:11:37.772092 COMPLETE=2003/08/07 04:11:37.787469
EXITFLAGS=00800000
LUWID=USIBMT6.MF1LUA07 958E44394D09 0001 TID=                  GTID=
```



```

FORMATID=                (decimal)          (hexadecimal)
GTRID=

EQUAL=
RMNAME=ATB.USIBMT6.MF2PROT3.IBM          ROLE=DSRM
  FLAGS=100E0000 PROTOCOL=PresumeNothing
  StateCheck EXIT RC=00000000
  Prepare     EXIT RC=00000000
  DistSp      EXIT RC=00000000
  Commit      EXIT RC=00000000
  Backout     EXIT RC=Uncalled
  EndUr       EXIT RC=00000000
  ExitFailed  EXIT RC=Uncalled
  Completion  EXIT RC=00000000
  OnlyAgent   EXIT RC=Uncalled
RMNAME=DSN.RRSATF.IBM.RDF2              ROLE=Participant
  FLAGS=10000000 PROTOCOL=PresumeNothing
  StateCheck EXIT RC=Uncalled
  Prepare     EXIT RC=00000010
  DistSp      EXIT RC=Uncalled
  Commit      EXIT RC=Uncalled
  Backout     EXIT RC=Uncalled
  EndUr       EXIT RC=Uncalled
  ExitFailed  EXIT RC=Uncalled
  Completion  EXIT RC=Uncalled
  OnlyAgent   EXIT RC=Uncalled

```

In this example, we see that the APPC/MVS application (resource manager ATB.USIBMT6.MF2PROT3.IBM) has taken on the RRS DSRM row. The DISTSP exit has been enabled. In this case, DB2 has returned x'10' from its prepare exit, meaning ATRX_FORGET. DB2 is telling RRS to remove its interest in this UR.

Archived

DFSMStvs

In this chapter, we describe how DFSMStvs in z/OS V1.4 uses RRS.

This chapter covers the following topics:

- ▶ DFSMStvs RRS requirements
- ▶ DFSMStvs features that exploit RRS
- ▶ RRS facilities that are exploited
- ▶ Restart and recovery issues

14.1 DFSMStvs features that exploit RRS

DFSMStvs is an enhancement to VSAM RLS. It allows batch applications update access to recoverable data sets. As with RLS, DFSMStvs is a mode of access to data sets, not a new form of data set. VSAM RLS continues to provide both read integrity and write integrity for data sets using Coupling Facility-based locking and caching.

DFSMStvs is an extension to VSAM RLS locking and therefore uses the same locking structure as other users of VSAM RLS. This allows applications using VSAM RLS and applications using DFSMStvs to read and write recoverable data sets concurrently.

DFSMStvs also supports the logging of updates to VSAM recoverable data sets in the same forward recovery log that CICS uses. If the VSAM file is opened in DFSMStvs mode, the following DFSMStvs functions will be enabled:

Resource locking

This function is implemented in VSAM RLS. DFSMStvs uses the existing locking and deadlock detection functions of VSAM RLS unchanged to ensure compatibility between concurrent accesses in RLS mode and DFSMStvs mode.

A new type of read access, consistent read explicit (CRE), is added by DFSMStvs for use in batch applications. It is the same as the repeatable read that only CICS transactions have in VSAM RLS. Since the same locking structure is used, when a lock is held in one mode (whether RLS or DFSMStvs), an application is unable to obtain the locked record in the other mode in an inconsistent state.

For example, if a lock is held as shared in RLS mode, it could not be obtained as exclusive in DFSMStvs mode—this would be such an inconsistency. This serialization prevents VSAM RLS users and DFSMStvs users from attempting to modify the same record at the same time.

Resource recovery logging

DFSMStvs enables forward recovery for VSAM files. This function enables a resource manager to use a redo log to keep a record of the changes made to recoverable resources by applications. In the VSAM RLS environment, redo logging was provided by CICS. CICS and DFSMStvs use the same redo log. The redo log is implemented in DFSMStvs by using the services provided by the system logger. The data provided in this redo log stream can be used by a forward recovery product, such as CICS/VR, at recovery time to reconstruct a VSAM file after a failure.

Two-phase commit and backout

CICS already provides these functions and now DFSMStvs adds support for them at the file system level. Two-phase commit and backout in DFSMStvs are supported by RRS.

In the following sections we describe how DFSMStvs is involved in the two-phase commit protocol, and explain its interactions with RRS.

14.1.1 Resource recovery participants

There are three parties involved in basic resource recovery protocols:

- ▶ The application program
- ▶ The resource manager(s)
- ▶ The syncpoint manager

In the following sections we discuss these entities in more detail. Note that for DFSMStvs, however, DFSMStvs itself is the resource manager and RRS acts as the syncpoint manager.

The application program

The *application* is the program that accesses protected resources, for example a banking transaction that requires money to be transferred from a savings account to a checking account. There are two separate operations that must occur to satisfy this transaction: money must be subtracted from the savings account, and an identical amount must be added to the checking account. From the bank's point of view, it is important that the accounts never be left in a state where the transfer amount is in both accounts. From the customer's point of view, it is important that the accounts never be left in a state where the transfer amount is in neither account.

After the individual operations have taken place, the application can take one of two actions to ensure that changes to the protected resources are synchronized. This is a sync point in the application. It can either:

- ▶ Invoke a commit service to make the change permanent
- ▶ Invoke a backout service to undo the changes and restore the protected resources to their previous stable state (also referred to as a rollback or abort)

Notice that only the application has sufficient information to determine when it is time to commit or to back out. The system must have a way of ensuring that once the application indicates commit or backout, the transaction either completes in its entirety or nothing at all changes.

The resource manager

The second participant in resource recovery is the resource manager. The *resource manager* is the program that provides interfaces through which applications manipulate protected resources. It is also responsible for carrying out resource-specific actions to commit changes. Resource managers can control either protected or unprotected resources. A resource manager that controls protected resources (often termed a *protected resource manager*) has the following characteristics:

- ▶ It provides an application programming interface (API) that applications use to manipulate resources.
- ▶ It logs changes to data before making the changes permanent.
- ▶ It logs the state of the unit of recovery.
- ▶ It reacts to invocations of the commit/backout services.
- ▶ It has failure recovery mechanisms that allow it to restore data to the state of its previous unit of recovery.

Unprotected resource managers generally do not log data changes or react to the commit/backout services. DFSMStvs runs as a protected resource manager.

The syncpoint manager

The final participant in resource recovery is the *syncpoint manager*, which coordinates the commitment of multiple protected resources of potentially different resource types (for example, DB2 databases or VSAM data sets). The role of the syncpoint manager is to take a snapshot of the set of resource managers that the application is using prior to a commit. It also provides a protocol to ensure that the transaction is executed completely or not at all.

14.1.2 Commit flow with DFSMStvs

When an application program has completed the set of record updates that it considers to be an atomic unit, it invokes commit to make those changes permanent. During commit, the commit coordinator, RRS, invokes the commit exits of each of the interested parties (the resource managers, who are also known as commit participants).

During phase 1 of commit, the commit participants harden their changes and logs and determine if it is possible to commit. If all of the commit participants agree to commit, processing continues with phase 2 of commit.

The flow of actions for the commit process in two-phase commit is shown in Figure 14-1:

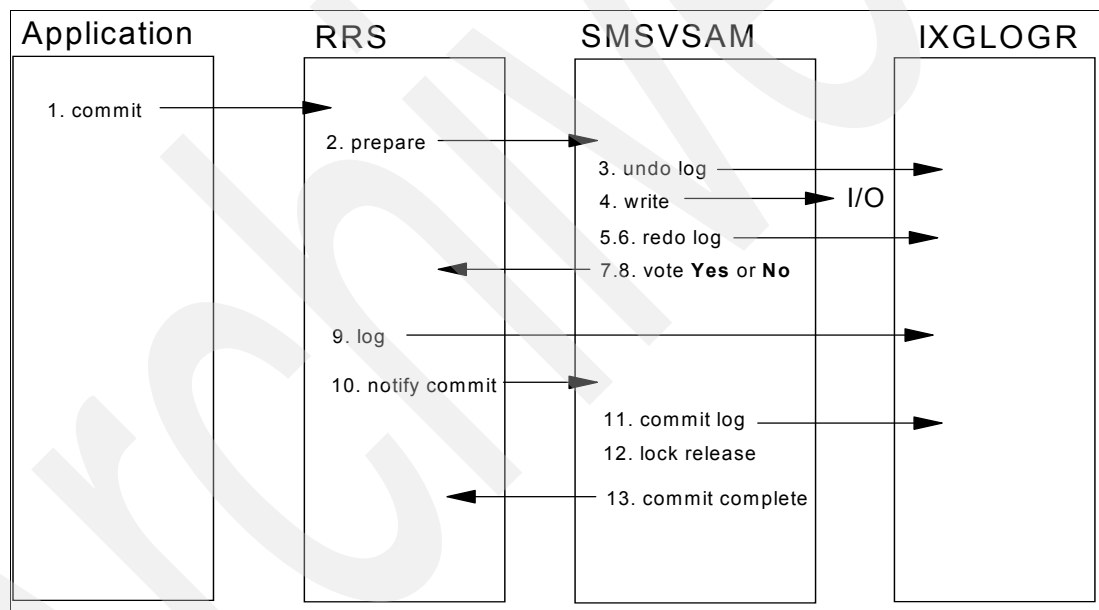


Figure 14-1 DFSMStvs commit flow

Following is the flow of actions for the commit process in two-phase commit:

1. The application program requests commit by invoking the RRS commit service.
2. RRS drives the prepare exit of each recoverable resource commit participant that has commit responsibility for this unit of recovery.
3. DFSMStvs ensures that all undo log records for the unit of recovery are hardened to the system log stream managed by the system logger.
4. DFSMStvs ensures that all I/O is complete and that any modifications to recoverable data sets made by this unit of recovery are written to disk. This includes changes made using sequential, skip sequential, and direct note string position (NSP) requests. DFSMStvs issues an ENDREQ against all RPLs that hold position within VSAM RLS data sets for this unit of recovery. The ENDREQs change the status of the unit of recovery's exclusive

record locks such that the unit of recovery may now access the records for backout using a different RPL.

5. DFSMStvs ensures that all redo log records for the unit of recovery are hardened to the forward recovery log stream managed by the system logger.
6. DFSMStvs does not write a log record showing phase 1 of commit processing for the unit of recovery has completed because such a log record would have no effect on DFSMStvs processing.
7. DFSMStvs determines if any errors occurred while processing the unit of recovery that make it ineligible for commit. If so, it votes no.
8. If there were no such errors, DFSMStvs votes yes.
9. When all participants have voted yes, RRS writes a log record to non-volatile storage showing the UR's status as in-commit.
10. RRS then drives the commit exit of each participant.
11. DFSMStvs creates a commit complete log record before it releases the locks for the unit of recovery, and writes the commit complete record to the system log.
12. DFSMStvs releases all the locks held by the unit of recovery.
13. DFSMStvs then returns commit complete to RRS.

14.1.3 Backout flow with DFSMStvs

When a unit of recovery backs out, the commit participants are responsible for backing out all changes to recoverable resources made by the unit of recovery. An exclusive lock is held for each record update, add, and delete performed by the unit of recovery to recoverable VSAM data sets.

These locks are held with the unit of recovery ID (URID) of the owner. They inhibit other units of recovery from changing or reading with integrity (using either consistent read or consistent read explicit) the records. While these locks inhibit access to the records by other units of recovery, they do not inhibit access by the unit of recovery that is backing out. The commit participant accesses these records to perform the backout operations.

VSAM requests issued by the commit participant during the backout may return an error status. This can cause the backout to fail. The flow of control shown in Figure 14-2 assumes that there were no VSAM request errors during the backout.

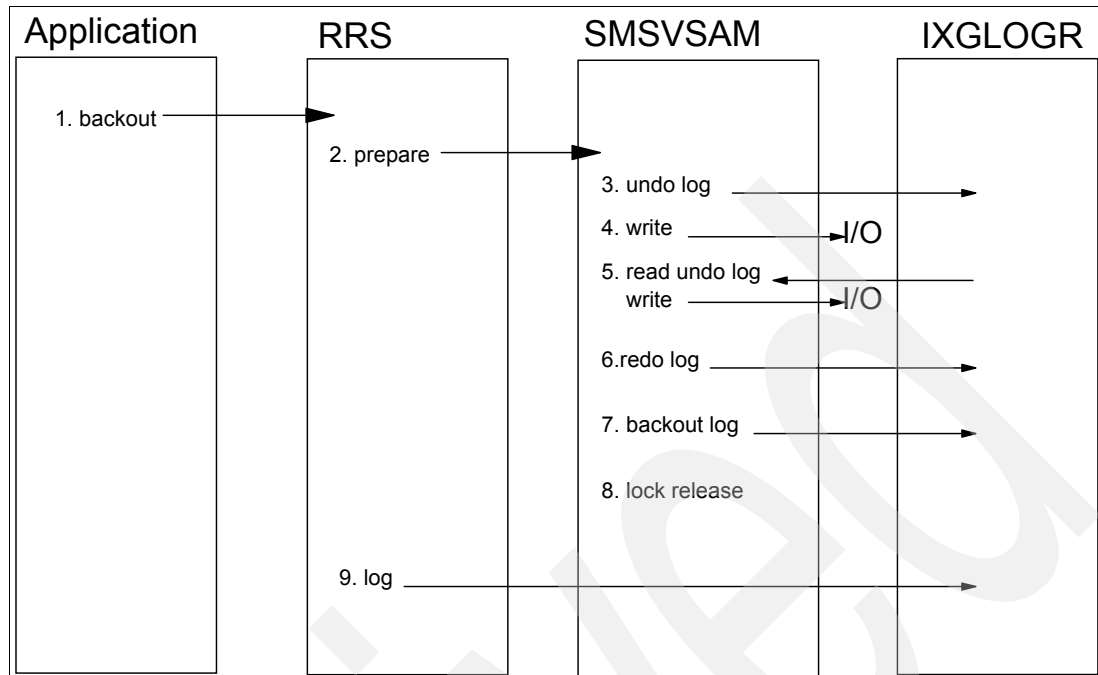


Figure 14-2 DFSMStvs backout flow

The backout flow occurs as follows:

1. The application requests backout by invoking the RRS backout service, or RRS initiates backout as a result of abnormal termination of the unit of recovery.
2. RRS drives the backout exit of each recoverable resource commit participant that has commit responsibility for this unit of recovery.
3. DFSMStvs ensures that all undo log records for the unit of recovery are hardened to the system log stream managed by the system logger.
4. DFSMStvs ensures that all I/O is complete and that any modifications to recoverable data sets made by this unit of recovery are written to DASD. This includes changes made using sequential, skip sequential, and direct NSP (note string position) requests. DFSMStvs issues an ENDREQ against all RPLs that hold position within VSAM RLS data sets for this unit of recovery. The ENDREQs change the status of the unit of recovery's exclusive record locks so that the unit of recovery may now access the records for backout using a different RPL.
5. DFSMStvs reads its system log to determine the changes that must be backed out. It undoes any updates, deletes and adds (except to an ESDS).
6. DFSMStvs writes before image records to the redo log (if applicable).
7. DFSMStvs creates a backout complete log record before it releases the locks for the unit of recovery and writes the backout complete record to the system log.
8. DFSMStvs releases all of the locks held by the unit of recovery.
9. RRS writes a log record indicating that the unit of recovery has been backed out.

14.1.4 Handling of undo records when in-doubt with DFSMStvs

A unit of recovery can become in-doubt during the period between a yes response from the resource managers to an RRS request for prepare and the time RRS invokes commit.

Once a commit is issued and all participants in the commit respond positively to the prepare request, the following occurs:

- ▶ The unit of recovery on the system that issued the commit becomes In_Commit.
- ▶ The unit of recovery on other systems becomes in-doubt until the distributed syncpoint resource manager receives the prepare response from the system that initiated the commit.

14.1.5 Handling long-running units of recovery with DFSMStvs

A *long-running unit of recovery* is one that makes a request which causes the unit of recovery to become in-flight and then does not issue a syncpoint request (commit or backout) for a long period of time. This can cause the unit of recovery to hold a large number of locks, as well as to write a large number of log records.

The undo log records for in-doubt and long-running units of recovery cause a problem for management of the space within backout log streams. Ideally, the undo records in a backout log stream have a short life cycle. This enables the deletion of obsolete entries from the log stream, thus avoiding offload of the log data by the system logger from the Coupling Facility to disk data sets. Units of recovery that do not reach a syncpoint within a short period interfere with the deletion of obsolete entries.

Two possible conditions could occur when attempting to write records to a log stream using the z/OS System Logger if too much old data is left in the log:

- ▶ The z/OS System Logger could return a return code and reason code indicating that the Coupling Facility storage limit has been reached. When this occurs, the z/OS System Logger begins offloading data to DASD. DFSMStvs cannot write any further information to its log streams until the problem has been resolved. DFSMStvs periodically retries the request to write data until the request finally succeeds.
- ▶ The z/OS System Logger can return a return code and reason code indicating that the staging data set storage limit has been reached. Again, when this occurs, the z/OS System Logger begins offloading data to disk. DFSMStvs cannot write any further information to its log streams until the problem has been resolved.

DFSMStvs uses a special log stream called the *secondary system log stream (or shunt log)* as a place for keeping the undo records of in-doubt and long-running units of recovery. The undo records are first recorded in the primary system log stream. Later, when the unit of recovery's status changes to in-doubt or DFSMStvs determines that this is a long-running unit of recovery, its previously logged undo records are moved from the primary to the secondary system log stream. This frees the space in the primary stream.

DFSMStvs also uses the secondary log for units of recovery that it is unable to complete (for example due to an I/O error or unavailability of a resource, such as a volume or a cache) and for long-running units of recovery.

14.2 TVS restart and recovery with RRS

DFSMStvs depends upon RRS to provide transactional support to its unit of work. In the following sections, we describe how these subsystems affect each other in restart/recovery situations.

14.2.1 RRS failure

If RRS fails, then DFSMStvs stops accepting requests. DFSMStvs requires RRS to provide its transactional support and cannot function without it. RRS must be restarted on the image before DFSMStvs processes new requests on this image. As soon as RRS is restarted, DFSMStvs reconnects to the subsystem. Example 14-1 is a syslog sample showing the interaction between the two subsystems during an RRS failure.

Example 14-1 Syslog of RRS failure

```
ASA2960I RRS SUBSYSTEM FUNCTIONS DISABLED. COMPONENT ID=SCRRS
ATR167I RRS RESMGR PROCESSING COMPLETED.
IGW471I DFSMS VSAM RLS REQUEST TO DISABLE
TRANSACTIONAL VSAM INSTANCE IGWTV002 IS ACCEPTED.
DISABLE REASON: TRANSACTIONAL VSAM DETECTED RRS IS UNAVAILABLE
IGW471I DFSMS VSAM RLS COMMAND PROCESSOR ON SYSTEM: #@$2
      IS WAITING FOR A RESPONSE
      FROM TRANSACTIONAL VSAM: IGWTV002
      COMMAND REQUESTED:
      DISABLE TRANSACTIONAL VSAM: IGWTV002
IGW473I DFSMS VSAM RLS COMMAND PROCESSOR ON SYSTEM: #@$2
IS WAITING FOR A RESPONSE FROM TRANSACTIONAL VSAM: IGWTV002
COMMAND REQUESTED: DISCONNECT FROM LOGSTREAM: #@$C.CICSVR.LGOFLOGS
IGW473I DFSMS VSAM RLS COMMAND PROCESSOR ON SYSTEM: #@$2
IS WAITING FOR A RESPONSE FROM TRANSACTIONAL VSAM: IGWTV002
COMMAND REQUESTED: DISABLE LOGSTREAM: IGWTV002.IGWLOG.SYSLOG
IGW474I DFSMS VSAM RLS IS DISCONNECTING FROM
TRANSACTIONAL VSAM LOGSTREAM IGWTV002.IGWLOG.SYSLOG
SYSTEM NAME: #@$2
TRANSACTIONAL VSAM INSTANCE NAME: IGWTV002
IGW474I DFSMS VSAM RLS IS DISCONNECTING FROM
TRANSACTIONAL VSAM LOGSTREAM IGWTV002.IGWSHUNT.SHUNTLOG
SYSTEM NAME: #@$2
TRANSACTIONAL VSAM INSTANCE NAME: IGWTV002
IGW473I DFSMS VSAM RLS COMMAND PROCESSOR ON SYSTEM: #@$2
HAS BEEN POSTED BY TRANSACTIONAL VSAM: IGWTV002
COMMAND REQUESTED: DISABLE LOGSTREAM: IGWTV002.IGWLOG.SYSLOG
IGW474I DFSMS VSAM RLS IS DISCONNECTING FROM
TRANSACTIONAL VSAM LOGSTREAM #@$C.CICSVR.LGOFLOGS
SYSTEM NAME: #@$2
TRANSACTIONAL VSAM INSTANCE NAME: IGWTV002
IGW473I DFSMS VSAM RLS COMMAND PROCESSOR ON SYSTEM: #@$2
HAS BEEN POSTED BY TRANSACTIONAL VSAM: IGWTV002
COMMAND REQUESTED: DISABLE LOGSTREAM: IGWTV002.IGWSHUNT.SHUNTLOG
IGW471I DFSMS VSAM RLS COMMAND PROCESSOR ON SYSTEM: #@$2
HAS BEEN POSTED BY TRANSACTIONAL VSAM: IGWTV002
COMMAND REQUESTED: DISABLE TRANSACTIONAL VSAM: IGWTV002
IGW471I DFSMS VSAM RLS COMMAND PROCESSOR ON SYSTEM: #@$2
HAS CALLED THE DFSMS COMMAND COMPLETE PROCESSOR
COMMAND REQUESTED: DISABLE TRANSACTIONAL VSAM: IGWTV002
IGW473I DFSMS VSAM RLS COMMAND PROCESSOR ON SYSTEM: #@$2
IS WAITING FOR A RESPONSE FROM TRANSACTIONAL VSAM: IGWTV002
COMMAND REQUESTED: DISABLE LOGSTREAM: IGWTV002.IGWSHUNT.SHUNTLOG
IGW471I DFSMS VSAM RLS REQUEST TO DISABLE
TRANSACTIONAL VSAM INSTANCE IGWTV002 IS COMPLETED.
TRANSACTIONAL VSAM INSTANCE IGWTV002 IS NOW DISABLED.
TRANSACTIONAL VSAM LOGSTREAM IGWTV002.IGWLOG.SYSLOG IS NOW DISABLED
TRANSACTIONAL VSAM LOGSTREAM IGWTV002.IGWSHUNT.SHUNTLOG IS NOW DISABLED
ATR222I LOG TAKEOVER FOR SYSTEM #@$2 HAS COMPLETED SUCCESSFULLY.
```

D SMS,TRANVSAM
 IGW800I 10.47.37 DISPLAY SMS,TRANSACTIONAL VSAM

DISPLAY SMS,TRANSACTIONAL VSAM - SERVER STATUS

System	TVSNAME	State	Rrs	#Urs	Start	AKP	QtimeOut
#@\$2	IGWTV002	DISED	UNREG		0 WARM/COLD	1000	300

DISPLAY SMS,TRANSACTIONAL VSAM - LOGSTREAM STATUS

LogStreamName	State	Type	Connect Status
IGWTV002.IGWLOG.SYSLOG	Disabled	UnDoLog	DisConnected
IGWTV002.IGWSHUNT.SHUNTLOG	Disabled	ShuntLog	DisConnected
#@\$C.CICSVR.LGOFLGS	Enabled	LogOfLogs	DisConnected

S RRS

ATR221I RRS IS JOINING RRS GROUP #@\$#PLEX ON SYSTEM #@\$2
 IXL014I IXLCONN REQUEST FOR STRUCTURE RRS_RMDATA_1
 WAS SUCCESSFUL. JOBNAME: IXGLOGR ASID: 0014
 CONNECTOR NAME: IXGLOGR_#@\$2 CFNAME: FACIL03
 IXL014I IXLCONN REQUEST FOR STRUCTURE RRS_MAINUR_1
 WAS SUCCESSFUL. JOBNAME: IXGLOGR ASID: 0014
 CONNECTOR NAME: IXGLOGR_#@\$2 CFNAME: FACIL03
 IXL014I IXLCONN REQUEST FOR STRUCTURE RRS_DELAYEDUR_1
 WAS SUCCESSFUL. JOBNAME: IXGLOGR ASID: 0014
 CONNECTOR NAME: IXGLOGR_#@\$2 CFNAME: FACIL03
 IXL014I IXLCONN REQUEST FOR STRUCTURE RRS_RESTART_1
 WAS SUCCESSFUL. JOBNAME: IXGLOGR ASID: 0014
 CONNECTOR NAME: IXGLOGR_#@\$2 CFNAME: FACIL03
 IXL014I IXLCONN REQUEST FOR STRUCTURE RRS_ARCHIVE_1
 WAS SUCCESSFUL. JOBNAME: IXGLOGR ASID: 0014
 CONNECTOR NAME: IXGLOGR_#@\$2 CFNAME: FACIL03
 IGW472I DFSMS VSAM RLS REQUEST TO ENABLE
 TRANSACTIONAL VSAM INSTANCE IGWTV002 ACCEPTED.
 ENABLE REASON: TRANSACTIONAL VSAM DETECTED RRS IS AVAILABLE
 IGW471I DFSMS VSAM RLS COMMAND PROCESSOR ON SYSTEM: #@\$2
 IS WAITING FOR A RESPONSE FROM TRANSACTIONAL VSAM: IGWTV002
 COMMAND REQUESTED: ENABLE TRANSACTIONAL VSAM: IGWTV002
 IGW473I DFSMS VSAM RLS COMMAND PROCESSOR ON SYSTEM: #@\$2
 IS WAITING FOR A RESPONSE FROM TRANSACTIONAL VSAM: IGWTV002
 COMMAND REQUESTED: ENABLE LOGSTREAM: IGWTV002.IGWLOG.SYSLOG
 IGW473I DFSMS VSAM RLS COMMAND PROCESSOR ON SYSTEM: #@\$2
 IS WAITING FOR A RESPONSE FROM TRANSACTIONAL VSAM: IGWTV002
 COMMAND REQUESTED: ENABLE LOGSTREAM: IGWTV002.IGWSHUNT.SHUNTLOG
 ASA2011I RRS INITIALIZATION COMPLETE. COMPONENT ID=SCRRS
 021 IGW879A TRANSACTIONAL VSAM COLD START REQUESTED REPLY 'COLD', 'WARM', OR 'DISABLE'
 R 21,WARM
 IEE600I REPLY TO 021 IS;WARM
 IGW860I TRANSACTIONAL VSAM HAS SUCCESSFULLY REGISTERED WITH RLS
 IGW888I TRANSACTIONAL VSAM PERMITNONRLSUPDATE EXIT NOT LOADED FOR
 INSTANCE IGWTV002 ON SYSTEM #@\$2
 IGW848I 1229200310.48.52 SYSTEM UNDO LOG IGWTV002.IGWLOG.SYSLOG INITIALIZATION HAS STARTED
 IXL014I IXLCONN REQUEST FOR STRUCTURE LOG_IGWLOG_001 WAS SUCCESSFUL.
 JOBNAME: IXGLOGR ASID: 0014
 CONNECTOR NAME: IXGLOGR_#@\$2 CFNAME: FACIL04
 IGW474I DFSMS VSAM RLS IS CONNECTING TO
 TRANSACTIONAL VSAM LOGSTREAM IGWTV002.IGWLOG.SYSLOG
 SYSTEM NAME: #@\$2
 TRANSACTIONAL VSAM INSTANCE NAME: IGWTV002

```

IGW848I 12292003 10.48.53 SYSTEM UNDO LOG IGWTV002.IGWLOG.SYSLOG
INITIALIZATION HAS ENDED
IGW848I 12292003 10.48.53 SYSTEM SHUNT LOG IGWTV002.IGWSHUNT.SHUNTLOG
INITIALIZATION HAS STARTED
IXL014I IXLCONN REQUEST FOR STRUCTURE LOG_IGWSHUNT_001
WAS SUCCESSFUL. JOBNAME: IXGLOGR ASID: 0014
CONNECTOR NAME: IXGLOGR_#@$2 CFNAME: FACIL03
IGW474I DFSMS VSAM RLS IS CONNECTING TO
TRANSACTIONAL VSAM LOGSTREAM IGWTV002.IGWSHUNT.SHUNTLOG
SYSTEM NAME: #@$2
TRANSACTIONAL VSAM INSTANCE NAME: IGWTV002
IGW848I 12292003 10.48.55 SYSTEM SHUNT LOG IGWTV002.IGWSHUNT.SHUNTLOG
INITIALIZATION HAS ENDED
IGW848I 12292003 10.48.55 LOG OF LOGS #@$C.CICSVR.LGOFLOGS
INITIALIZATION HAS STARTED
IXL014I IXLCONN REQUEST FOR STRUCTURE CIC_GENERAL_001
WAS SUCCESSFUL. JOBNAME: IXGLOGR ASID: 0014
CONNECTOR NAME: IXGLOGR_#@$2 CFNAME: FACIL03
IGW474I DFSMS VSAM RLS IS CONNECTING TO
TRANSACTIONAL VSAM LOGSTREAM #@$C.CICSVR.LGOFLOGS
SYSTEM NAME: #@$2
TRANSACTIONAL VSAM INSTANCE NAME: IGWTV002
IGW848I 12292003 10.48.56 LOG OF LOGS #@$C.CICSVR.LGOFLOGS
INITIALIZATION HAS ENDED
IGW865I TRANSACTIONAL VSAM INITIALIZATION IS COMPLETE.
IGW471I DFSMS VSAM RLS COMMAND PROCESSOR ON SYSTEM: #@$2
HAS BEEN POSTED BY TRANSACTIONAL VSAM: IGWTV002
COMMAND REQUESTED: ENABLE TRANSACTIONAL VSAM: IGWTV002
IGW473I DFSMS VSAM RLS COMMAND PROCESSOR ON SYSTEM: #@$2
HAS BEEN POSTED BY TRANSACTIONAL VSAM: IGWTV002
COMMAND REQUESTED: ENABLE LOGSTREAM: IGWTV002.IGWLOG.SYSLOG
IGW473I DFSMS VSAM RLS COMMAND PROCESSOR ON SYSTEM: #@$2
HAS BEEN POSTED BY TRANSACTIONAL VSAM: IGWTV002
COMMAND REQUESTED: ENABLE LOGSTREAM: IGWTV002.IGWSHUNT.SHUNTLOG
IGW473I DFSMS VSAM RLS COMMAND PROCESSOR ON SYSTEM: #@$2
HAS CALLED THE DFSMS COMMAND COMPLETE PROCESSOR
COMMAND REQUESTED: ENABLE LOGSTREAM: IGWTV002.IGWSHUNT.SHUNTLOG
IGW886I 0 RESTART TASKS WILL BE PROCESSED DURING TRANSACTIONAL VSAM RESTART PROCESSING
IGW866I TRANSACTIONAL VSAM RESTART PROCESSING IS COMPLETE.
IGW472I DFSMS VSAM RLS REQUEST TO ENABLE
TRANSACTIONAL VSAM INSTANCE IGWTV002 IS COMPLETED.
TRANSACTIONAL VSAM INSTANCE IGWTV002 IS NOW ENABLED.
TRANSACTIONAL VSAM LOGSTREAM IGWTV002.IGWLOG.SYSLOG IS NOW ENABLED.
TRANSACTIONAL VSAM LOGSTREAM IGWTV002.IGWSHUNT.SHUNTLOG IS NOW ENABLED.
TRANSACTIONAL VSAM IGWTV002 WILL NOW ACCEPT NEW WORK
IGW467I DFSMS TVSNAME PARMLIB VALUE SET DURING
SMSVSAM ADDRESS SPACE INITIALIZATION ON SYSTEM: #@$2
TVSNAME: IGWTV002
CURRENT VALUE: ENA-ED 1
IGW467I DFSMS TRANSACTIONAL VSAM UNDO LOG PARMLIB VALUE SET DURING
SMSVSAM ADDRESS SPACE INITIALIZATION ON SYSTEM: #@$2
UNDO LOGSTREAM NAME: IGWTV002.IGWLOG.SYSLOG
CURRENT VALUE: ENA-ED 1
IGW467I DFSMS TRANSACTIONAL VSAM SHUNT LOG PARMLIB VALUE SET DURING
SMSVSAM ADDRESS SPACE INITIALIZATION ON SYSTEM: #@$2
SHUNT LOGSTREAM NAME: IGWTV002.IGWSHUNT.SHUNTLOG
CURRENT VALUE: ENA-ED 1

```

Note: In our sample, DFSMSUvs was interacting with the operator at startup time since COLD is specified in our IGDSMS parmlib member and the UnDo log stream is not empty. When this

situation occurs, DFSMStvs verifies with the operator whether it should complete the initialization in COLD start or WARM start.

14.2.2 DFSMStvs restart

During restart processing, DFSMStvs requests a list of the units of recovery for which VSAM has retained locks. DFSMStvs releases retained locks for any units of recovery for which its undo (system) log contains no backout log records.

Restart processing of a failed instance of DFSMStvs includes backout of changes to VSAM recoverable data sets. These changes were made by in-flight (active and not yet committed) units of recovery that were forced to terminate when the instance of DFSMStvs failed.

Restart processing resolves any in-doubt units of recovery; however, DFSMStvs cannot take action on in-doubt units of recovery until it is told to do so by RRS. Therefore, restart processing may complete but leave some outstanding in-doubt units of recovery that are waiting for RRS to determine whether they should be committed or backed out.

When DFSMStvs completes backout processing for the unit of recovery required for restart, it releases all active and retained locks held by the unit of recovery with the exception of any locks that are marked for retention due to errors encountered during the attempted backout. Any locks that are marked for retention become retained locks.

14.2.3 DFSMStvs peer restart

If the SMSVSAM address space fails, all in-flight units of recovery that are using the DFSMStvs instance at the time of its failure are backed out during the restart process of the failed SMSVSAM. This happens automatically.

However, in the case of a z/OS system failure, restart can take a comparatively long time, and the process is not automatic. As a result, in-flight units of recovery can remain for a long time. To prevent this situation, DFSMStvs provides a function known as *peer recovery*.

When peer recovery occurs, another SMSVSAM instance in the sysplex (a peer) performs the backout for in-flight units of recovery that were being processed by the SMSVSAM address space that failed. Peer recovery starts automatically if there is an ARM policy (created using IXCMIAPU) that includes DFSMStvs and any other resource managers that might be involved in a unit of recovery in a restart group.

If you do not use ARM, you should change your operational procedures to issue the following command in another system as soon as possible in case of a system failure:

```
VARY SMS,TRANSVSAM(tvsnam),PEERRECOVERY,ACTIVE
```

Example 14-2 is a sample of the ARM definitions to activate peer recovery. In our case, we want to activate peer recovery on system #@\$3 because it is the only other system in our sysplex that has the right software level for DFSMStvs—and DFSMStvs is the only element in the restart group since our workload is only batch.

Example 14-2 ARM policy definitions

```
DEFINE POLICY NAME(ARM01)
  RESTART_GROUP(TVS)
  TARGET_SYSTEM(#@$3)
  ELEMENT(IGWTV002)
  RESTART_ATTEMPTS(3,120)
  RESTART_TIMEOUT(60)
  READY_TIMEOUT(900)
```

```
TERMTYPE(ALLTERM)
RESTART_METHOD(SYSTEM,STC,
'VARY SMS,TRANVSAM(002),PEERRECOVERY,ACTIVE')
```

14.2.4 Operator commands

There are some changes to operator commands for DFSMStvs. In the following sections, we describe the changes to the SMS command and show sample output from these commands.

Display SMS

The new options for the Display SMS command shown in Example 14-3 are used to capture information about the DFSMStvs environment. Specifically, you can display the following resources or verify their status:

- ▶ Check that a particular unit of recovery (UR) is currently active within the sysplex.
- ▶ Verify all units of recovery currently active on the system on which the command was issued and on whose behalf DFSMStvs has performed any work.
- ▶ Discover entries currently contained in the shunt logs of the systems in the sysplex.
- ▶ Display information about log streams that DFSMStvs is using on one of the systems in the sysplex. If ALL is specified, information is displayed about all log streams in use on the system on which the command is issued. The output includes:
 - Status of the log stream (failed or available)
 - Type of log (undo, shunt, forward recovery, or log of logs)
 - Job name and URID of the oldest unit of recovery using the log
 - List of all DFSMStvs instances using the log
- ▶ Understand which configuration options are in use for this instance of DFSMStvs.

Example 14-3 New options for the DISPLAY SMS command

```
DISPLAY SMS ,TRANVSAM {,ALL}
,JOB(jobname)
,URID(urid|ALL)
,SHUNTED, {SPHERE(sphere)|
URID(urid|ALL)}
,LOG(logstream| ALL)
,DSNAME(dsn)
,OPTIONS
```

Vary SMS

As shown in Example 14-4, you can use the VARY SMS command to change the status of the following resources:

- ▶ The VARY SMS,TRANVSAM command is used to enable, quiesce, or disable the specified DFSMStvs instance, or all DFSMStvs instances in the sysplex. It is routed to all systems in the sysplex and affects either all DFSMStvs instances, or the DFSMStvs instance with the name specified.

- ▶ The VARY SMS,LOG command is used to enable, quiesce, or disable DFSMStvs access to the specified log stream. Quiescing or disabling the DFSMStvs undo or shunt log stream is equivalent to quiescing or disabling DFSMStvs processing.
- ▶ The VARY SMS,SMSVSAM,SPHERE command is used to quiesce or unquiesce a data set for RLS or DFSMStvs access. The VARY SMS,SMSVSAM,SPHERE(sphere) command was provided in RLS to support unquiescing a data set when CICS was unavailable. This clears the VSAM-quiesced state for the specified sphere. This command is being extended to support quiescing a data set.
- ▶ The VARY SMS,TRANVSAM(tvsnam),PEERRECOVERY command is used to start or stop peer recovery processing for a failed instance of DFSMStvs. It applies only to the system on which it is issued. That system will then be responsible for performing all peer recovery processing for the failed DFSMStvs instance.

Example 14-4 New options for the VARY SMS command

```
VARY SMS ,TRANVSAM(tvsnam|ALL), {QUIESCE | Q}
      {DISABLE | D}
      {ENABLE | E}
,LOG(logstream), {QUIESCE | Q}
      {DISABLE | D}
      {ENABLE | E}
SMSVSAM,SPHERE(sphere), {QUIESCE | Q}
      {ENABLE | E}
,TRANVSAM(tvsnam), {ACTIVE |
PEERRECOVERY,  ACTIVEFORCE |
      INACTIVE}
```

SETSMS

The SETSMS command can be used, as shown in Example 14-5 on page 156, to change some of the DFSMStvs configuration variables. For example:

- ▶ AKP - the activity keypoint trigger value, which is the number of logging operations between the taking of keypoints. Valid values are 200 to 65535. The default is 1000.
- ▶ The QTIMEOUT that specifies the quiesce exit timeout value in seconds. This is the amount of time the DFSMStvs quiesce exit allows to elapse before it concludes that a quiesce cannot be completed successfully. Changing the value of QTIMEOUT affects only those quiesces that are submitted after the change is made. It has no effect on quiesces that are already in progress. Valid values are 60 to 3600. The default is 300.
- ▶ MAXLOCKS - this parameter specifies two values:
 - The maximum number of unique lock requests that a single unit of recovery may make before warning message IGW859I is issued to the system console and message IGW10074I is issued to the job log
 - An increment value - the warning messages will be issued every time the number of unique lock requests over and above the maximum increases by a multiple of the increment
- ▶ The RLSTMOUT parameter specifies the maximum time in seconds that a VSAM RLS or DFSMStvs request is to wait for a required lock before the request is assumed to be in deadlock and aborted. RLSTMOUT is specified as a value in seconds in the range of 0 to

9999. The default is 0 and means that the VSAM RLS or DFSMSStvs request has no timeout value; the request will wait for as long as necessary to obtain a lock.

Example 14-5 SETSMS sample command

```
SETSMS AKP(nnn | 1000)
QTIMEOUT(nnn | 300)
MAXLOCKS(max | 0, incr | 0)
RLSTMOUT(nnn|0)
```

SHCDS

The SHCDS IDCAMS command has been enhanced to be able to retrieve DFSMSStvs information that is useful to understand the status of a file, to perform diagnosis and recovery tasks in a failure scenario. Example 14-6 is a sample of the updated SHCDS command for DFSMSStvs.

Example 14-6 Updated SHCDS IDCAMS command for DFSMSStvs

```
SHCDS LISTDS(base-cluster) {JOBS}
LISTSHUNTED SPHERE(base-cluster)
LISTSHUNTED URID(urid|ALL)
RETRY SPHERE(base-cluster)
RETRY URID(urid)
PURGE SPHERE(base-cluster)
PURGE URID(urid)
```

For example:

- ▶ The RETRY and PURGE subcommands are used to take action on work that DFSMSStvs shunted. Units of recovery are shunted when DFSMSStvs is unable to finish processing them (for example, due to an I/O error). As long as a shunted log entry exists, the locks associated with that entry are retained. You can take action either on a particular unit of recovery, or on all records in the shunt log that apply to a particular data set.
- ▶ The LISTDS subcommand has been modified to accept a new, optional JOBS keyword. When this keyword is specified, this command also returns a list of the jobs currently accessing the data set in DFSMSStvs mode.
- ▶ The LISTSHUNTED subcommand lists information about work that was shunted due to an inability to complete a syncpoint (commit or backout) for a given data set or UR, or for all shunted URs when the ALL keyword is specified.

The output includes:

- The unit of recovery identifier(s)
- The data set name(s)
- The job with which the unit of recovery was associated
- The step within the job with which the unit of recovery was associated
- The disposition of the unit of recovery (whether it will be committed or backed out if it is retried)

14.3 DFSMStvs examples

In our environment, we had DFSMStvs active on one of our systems. The name of the subsystem was IGWTV002 and it was active on system #@\$2. Issuing the **D SMS, TRANVSAM** command, we can see the current status of the subsystem and the log streams used by this instance of DFSMStvs; see Figure 14-3.

DISPLAY SMS, TRANSACTIONAL VSAM - SERVER STATUS							
System	TVSNAME	State	Rrs	#Urs	Start	AKP	QtimeOut
#@\$2	IGWTV002	ACTIVE	REG	0	WARM/COLD	1000	300

DISPLAY SMS, TRANSACTIONAL VSAM - LOGSTREAM STATUS			
LogStreamName	State	Type	Connect Status
IGWTV002.IGWLOG.SYSLOG	Enabled	UnDoLog	Connected
IGWTV002.IGWSHUNT.SHUNTLOG	Enabled	ShuntLog	Connected
#@\$C.CICSVR.LGOFLOGS	Enabled	LogOfLogs	Connected
#@\$C.CICSVR.LOG001	Enabled	FrLog	DisConnected

Figure 14-3 Display DFSMStvs status

At this point, you can verify that DFSMStvs registered with RRS as a resource manager by looking through the RRS panel. In the main menu of the RRS panels, selecting the Display/Update RRS related Resource Manager information option should display the DFSMStvs subsystem listed as a resource manager, as shown in Figure 14-4.

This figure shows multiple entries for the IGWTV002 subsystem with the RESET status; these are the old entries for the subsystem. The entry you are interested in has a status of Run, and it represents the current instance of DFSMStvs running on #@\$2.

Commands: v-View Details u-View URs r-Remove Interest				
S	RM Name	State	System	Logging Group
—	IGWTV002020031870834097040620000	Reset	#@\$2	#@\$#PLEX
—	IGWTV002020031891417274421330000	Reset	#@\$2	#@\$#PLEX
—	IGWTV002020031960915105983550000	Reset	#@\$2	#@\$#PLEX
—	IGWTV002020032301225315079870000	Run	#@\$2	#@\$#PLEX

Figure 14-4 Display Resource Manager status

After verifying the subsystem was active we started the workload, which consisted of a set of batch jobs accessing VSAM files. The programs associated with these batch jobs were updated to use the RRS application interface through the SSRRCMIT and SSRBACK calls for commit and backout, respectively.

Note: You can monitor the effect of these interactions on the RRS panels while the workload is running, as follows:

- ▶ From the RRS main menu, under the option Display/Update RRS related Work Manager information you can see the multiple batch jobs that have URIDs and express interest in work requests.
- ▶ From the RRS main menu, under the option Display/Update RRS Unit of Recovery information, you can see the current in-flight URID.

Next, we introduced a failure in order to see what kind of information can be retrieved from the RRS environment that can help recover a DFSMStvs environment. We caused DFSMStvs to fail on #@\$2 while the workload was running.

It is up to the application program to deal with the DFSMStvs failure and return code; normally we would expect a batch job to warn about the error with a WTO on the log, then either loop waiting for DFSMStvs to come back or, more likely, to end.

On the RRA panel Display/Update RRS related Resource Manager information, the status of the resource manager changed from Run to Reset.

On the RRA panel Viewing the Display/Update RRS Unit of Recovery information option, shown in Figure 14-5, we found 8 URIDs that are in-flight and have not committed.

Commands: v-View Details c-Commit b-Backout r-Remove Interest f-View UR Family				
S	UR Identifier	System State	Logging Group Type	Comments
—	BA8B49067E740000000000D401030000	#@\$2 InFlight	#@\$#PLEX Unpr	X
—	BA8B49057E7403740000000CE01030000	#@\$2 InFlight	#@\$#PLEX Unpr	X
—	BA8B49057E740A5C0000000CA01030000	#@\$2 InFlight	#@\$#PLEX Unpr	X
—	BA8B48FF7E74114400000009401030000	#@\$2 InFlight	#@\$#PLEX Unpr	X
—	BA8B49067E7414B800000007001030000	#@\$2 InFlight	#@\$#PLEX Unpr	X
—	BA8B49057E74182C00000004601030000	#@\$2 InFlight	#@\$#PLEX Unpr	X
—	BA8B49057E741BA000000002E01030000	#@\$2 InFlight	#@\$#PLEX Unpr	X
—	BA8B49067E741F140000000E01030000	#@\$2 InFlight	#@\$#PLEX Unpr	X

Figure 14-5 Display URID

Figure 14-6 on page 159 shows the detailed report of the URID from the previous panel. A portion of the Work Manager Name (in this case, TVSV07C) reported in this panel corresponds to the batch job name and address space number that is associated with the in-flight URID that needs to be recovered once that DFSMStvs is restarted.

The batch jobs listed in this panel are the only ones that have a unit of recovery that will be backed out once that DFSMStvs restarts. The other jobs belonging to our workload are not processing new DFSMStvs requests, but at the same time they do not have any pending unit of recovery that needs to be cleared up at DFSMStvs restart time.

So, for example, if you know that DFSMStvs will not be able to restart for a long time and you need to run those jobs that do not have pending URIDs, you can restart them on other DFSMStvs instances. The jobs with in-flight URID could theoretically be restarted from the last committed record, but they will not be able to process the record(s) associated with the in-flight URs till DFSMStvs perform the backout. The records associated with the in-flight URs should be marked with retained locks to prevent further changes.

Note: If the instance on #@\$2 system does not come back in a short time, consider performing a peer recovery restart of the DFSMStvs 002 instance on a different system to perform the backout on these 8 URIDs, then restart these jobs from their last committed syncpoint. Batch jobs that access VSAM files through DFSMStvs must be able to restart from the last syncpoint. For further details on job restartability, refer to *DFSMStvs Application Migration Guide*, SG24-6072.

```
Commands r-Remove Interest v-View URI Details

UR identifier : BA8B49057E74182C00000004601030000
Create time : 2003/12/29 20:23:33.527356      Comments : X
UR state : InFlight      UR type : Unpr
System : #@$2      Logging Group : #@$#PLEX
SURID : N/A
Work Manager Name : #@$2.TVSV07C.002F
- Display Work IDs      - Display IDs formatted
  Luwid . : Not Present
  Eid . . : Not Present
  Xid . . : Not Present
Expressions of Interest:
S   RM Name                                     Type Role
***** Bottom of data *****
```

Figure 14-6 URID detailed report

To easily find the batch jobs that have any in-flight URID, use the RRS panel option Display/Update RRS related Work Manager information. Figure 14-7 shows the list of batch jobs that have in-flight URIDs that will require DFSMStvs to go through a backout process.

```
Commands: u-View URs

S   WM Name                                     System      Logging Group
-   #@$2.TVSV07C.002F      #@$2        #@$#PLEX
-   #@$2.TVSV08C.0030      #@$2        #@$#PLEX
-   #@$2.TVS001C.0033      #@$2        #@$#PLEX
-   #@$2.TVS005C.0035      #@$2        #@$#PLEX
-   #@$2.TVS006C.0036      #@$2        #@$#PLEX
-   #@$2.TVS007C.0037      #@$2        #@$#PLEX
-   #@$2.TVS032C.002B      #@$2        #@$#PLEX
-   #@$2.TVS034C.002C      #@$2        #@$#PLEX
***** Bottom of data *****
```

Figure 14-7 Work Managers Display

From this panel the UR associated with a specific batch job can be listed, as shown Figure 14-8.

(For more detailed information, use option “v”; this will display the information shown in Figure 14-6.)

```
Commands: v-View Details c-Commit b-Backout r-Remove Interest f-View UR Family

S   UR Identifier                                     System      Logging Group
-   BA8B49057E74182C00000004601030000      #@$2        #@$#PLEX
                                     State      Type      Comments
                                     InFlight  Unpr      X
***** Bottom of data *****
```

Figure 14-8 Work Manager associated URID

Using the IDCAMS SHCDS LISTDS command as illustrated in Figure 14-9 can show which VSAM files have retained locks and need recovery actions from DFSMStvs to back out the in-flight URs. Unfortunately, there are no filter keywords to extract only the data sets associated with retained locks; you need to execute the SHCDS LISTDS command against each data set associated to your application and find which ones have retained locks by scanning the report output.

```

SHCDS LISTDS('TVSV.VF01D.ACCUNTDB')
DATA SET NAME----TVSV.VF01D.ACCUNTDB
  CACHE STRUCTURE-----
  RETAINED LOCKS-----YES      NON-RLS UPDATE PERMITTED-----NO
  LOST LOCKS-----NO          PERMIT FIRST TIME-----NO
  LOCKS NOT BOUND-----NO      FORWARD RECOVERY REQUIRED-----NO
  RECOVERABLE-----YES

                                SHARING SUBSYSTEM STATUS
SUBSYSTEM      SUBSYSTEM      RETAINED      LOST      NON-RLS UPDATE
NAME           STATUS         LOCKS         LOCKS     PERMITTED
-----
IGWTV002      ONLINE--FAILED   YES           NO        NO

SHCDS LISTDS('TVSV.VF01D.BANKACCT')
DATA SET NAME----TVSV.VF01D.BANKACCT
  CACHE STRUCTURE-----
  RETAINED LOCKS-----NO      NON-RLS UPDATE PERMITTED-----NO
  LOST LOCKS-----NO          PERMIT FIRST TIME-----NO
  LOCKS NOT BOUND-----NO      FORWARD RECOVERY REQUIRED-----NO
  RECOVERABLE-----YES

IDC31891I SUBSYSTEM NOT LISTED

```

Figure 14-9 Extract from IDCAMS SHCDS LISTDS report

After locating all the information, we restarted DFSMStvs; see Example 14-7. In our system, DFSMStvs asks for confirmation on the type of start, cold or warm; this occurs because we specified a startup option of COLD in our parmlib member.

Example 14-7 DFSMStvs syslog at restart time

```

IGW865I TRANSACTIONAL VSAM INITIALIZATION HAS STARTED.

*024 IGW879A TRANSACTIONAL VSAM COLD START REQUESTED  REPLY 'COLD',
'WARM', OR 'DISABLE'

R 24,WARM
IEE600I REPLY TO 024 IS;WARM
IGW860I TRANSACTIONAL VSAM HAS SUCCESSFULLY REGISTERED WITH RLS
IXL014I IXLCONN REQUEST FOR STRUCTURE LOG IGWLOG_001
WAS SUCCESSFUL.  JOBNAME: IXGLOGR ASID: 0014
CONNECTOR NAME: IXGLOGR_#@$2 CFNAME: FACIL04
.....
.....
IGW865I TRANSACTIONAL VSAM INITIALIZATION IS COMPLETE.
IGW882I 9 INFLIGHT UNITS OF RECOVERY WERE RECONSTRUCTED
TRANSACTIONAL VSAM  LOGSTREAM #@$C.CICSVR.LOG001
SYSTEM NAME:          #@$2
TRANSACTIONAL VSAM INSTANCE NAME:  IGWTV002
IGW866I TRANSACTIONAL VSAM RESTART PROCESSING IS COMPLETE.
.....
.....

```

You can verify that in-flight URIDs have been recovered by looking at the RRS panels, or by checking for the following messages:

```
IGW10103I JOB TVS001C STEP G001 120  
UNIT OF RECOVERY BA8D9AE87E7406E80000612F01030000  
FAILED. BACKOUT WAS SUCCESSFUL
```

These batch jobs can now be restarted from their last committed syncpoint.

Archived

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 163. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *System Programmer's Guide to z/OS System Logger*, SG24-6898
- ▶ *DFSMSdfs Application Migration Guide*, SG24-6072
- ▶ *WebSphere for z/OS V5 Connectivity Handbook*, SG24-7064

Other publications

These publications are also relevant as further information sources:

- ▶ *z/OS MVS Initialization and Tuning Reference*, SA22-7592
- ▶ *z/OS V1R4 MVS Planning: APPC Management*, SA22-7599
- ▶ *z/OS MVS Programming: Resource Recovery*, SA22-7616
- ▶ *z/OS MVS Setting Up a Sysplex*, SA22-7625
- ▶ *IMS/ESA V8 Admin Guide: Transaction Manager*, SC26-8014
- ▶ *OTMA Guide and Reference*, SC26-8743
- ▶ *CICS Recovery and Restart Guide*, SC33-1698
- ▶ *CICS Intercommunication Guide*, SC33-1695
- ▶ *CICS Application Programming Guide*, SC33-1687
- ▶ *CICS External Interface Guide*, SC33-1944
- ▶ *OS/390 MVS Planning: APPC/MVS Management*, GC28-1807
- ▶ *DB2 for OS/390 Application Programming and SQL Guide*, SC26-8958

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Archived

Index

Numerics

2-phase commit
sample 11

A

ACID 14
APPC 138
APPL definition 139
atomic 21

B

backout 19
DFSMS flow 147

C

cascaded transaction
definition 26
CEMT command 111
CICS
EXCI 15, 106
transaction types 104
two-phase commit 14
commit 19
DFSMS flow 146
connector
CICS 78
DB2 79
IMS 76
MQ 81
context
differences 18
Coupling Facility
sizing 59
CSQBRSSI 130
CSQBRSTB 130
CTG 78
CTG Gateway 79

D

DASDONLY 42
DB2
JDBC driver 96
restart 97
RRS attach facility 94
sample scenarios 98
Stored Procedures 95
Display SMS 154
distributed syncpoint 25
distributed syncpoint resource manager 25
distributed unit of work
CICS 14
DRDA 15

DSRM 25
DUPLEXMODE 42

E

ECI 79
exploiters 7

H

HLQ 39, 42

I

IMS
APPC 117
OTMA 121
resource adapter 117
sample scenarios 126
shared queue 121
IMS connect 76
in-doubt 10, 148
ISPF panel 48

J

J2EE
terminology 74
J2EE Connector Architecture 75
JCA 78
JDBC 75
JDBC driver
Type 1 80, 96
Type 2 80, 96
Type 3 80, 96
Type 4 80, 96
JMS 75, 130

L

local JDBC driver 80
log stream
availability considerations 58
AVGBUFSIZE 37
CF definitions 35
CFRM policy 35
content 32
definition 35
DFSMS definitions 35
DUPLEXMODE 58
HIGHOFFLOAD 39
logger definitions 36
LOWOFFLOAD 39
LS_SIZE 40
MAXBUFSIZE 37
offloading 59
performance considerations 59

- SHAREOPTIONS(3,3) 35
- sizing 34
 - CF Sizer tool 34
- SMF type 88 34
- SMS constructs 35
- STG_DUPLEX 58
- STG_DUPLEX=(YES) 40
- STG_SIZE 40
- System Logger policy 35
- types 35
- log takeover 62
- logging group 33, 62
- logon procedure 43

M

- MAXBUFSIZE 42
- multisystem cascaded transactions 26

N

- native context 17
- non-conversational 104

O

- ODBA 116
- Open Database Access 116
- Open Transaction Manager Access 121
- OTMA 121

P

- peer recovery 153
- privately-managed context 17
- protected conversations 138
 - APPC 139
- protected resources 6
- pseudo-conversational 104

R

- recovery
 - peer-level 64
- recovery manager
 - CICS 14
- Redbooks Web site 163
 - Contact us x
- resource manager
 - definition 6
 - restart 63
 - restart restrictions 63
 - startup 63
- restart 123
 - multiple resource managers 64
 - restrictions 63
 - sample 66
 - sample scenario with DB2 66
- restart processing 62
- RRS
 - ATRRRS procedure 43
 - automation 43

- cold start 47
- configuration 32
- context services 17
- exploiters ix, 7, 172
- failure 82, 109, 124, 131, 140, 150
- IEFSSN definition 42
- invocation 18
- ISPF panels 43
- log streams 32
- registration services 16
- security definition 44
- services 16
- SMF records 60
- startup 46
- warm start 47
- WebSphere Application Server for z/OS 75
- Workload Manager definition 42
- RRSAF 94

S

- SDSRM 25, 76
- server distributed syncpoint resource manager 25
- services
 - context services 17
 - registration services 16
- setup 32
- SIT 110
- SMF 70-78 60
- SMF 88 60
- SRRBACK 15, 94
- SRRCMIT 15, 94
- staging data set size
 - sizing 59
- STG_DUPLEX 42
- Stored Procedures 95
- SYNCLEVEL=CONFIRM 138
- SYNCLEVEL=NONE 138
- SYNCLEVEL=SYNCPT 138
- syncpoint 13, 105
 - atomic 20
- syncpoint coordinator 19, 24
- syncpoint manager ix, 6, 172

T

- transaction
 - ACID 5
 - ACID definition 5
 - atomic 4–5
 - consistent 5
 - definition 5
 - durable 5
 - isolated 5
- two-phase commit 10
 - CICS 14
 - DB2 15
 - definition 6
 - distributed environment 24
 - IMS 15
 - legacy resource managers 13

phase 1 10, 21
phase 2 10
RRS 19
WebSphere Application Server 75

U

universal JDBC driver 80

V

VARY SMS 154

W

WebSphere Application Server
adapters 82
CICS connectors 78
DB2 connectors 79
IMS connectors 76
local attachment 77
MQ connector 81
WebSphere Application Server for z/OS
connectors 76
RRS exploitation 75

Archived



Systems Programmer's Guide to Resource Recovery Services (RRS)



Managing, optimizing and sizing the RRS environment

Restart and recovery with RRS

How exploiters can get the most out of RRS

This IBM Redbook gives you a broad understanding of the Resource Recovery Services (RRS) environment. RRS provides a global syncpoint manager that any resource manager on z/OS can exploit. It enables transactions to update protected resources managed by many resource managers.

RRS is increasingly becoming a prerequisite for new resource managers, and for new capabilities in existing resource managers. Rather than having to implement their own two-phase commit protocol, these products can use the support provided by RRS.

Since older transaction managers like CICS already offered many of the benefits of RRS for processing their own data, not many people rushed to exploit RRS when it was first introduced. However, as more transaction managers have become RRS resource managers, and as the complexity of the exchanges of transactional data increases, more and more systems and application programmers will need to use RRS.

This redbook provides information that will help you install, tailor, and manage the RRS environment. It covers RRS exploiters, helping you to understand the connections between RRS and its exploiters, how they work together, and how the installation should behave in recovery/restart situations.

**INTERNATIONAL
TECHNICAL
SUPPORT
ORGANIZATION**

**BUILDING TECHNICAL
INFORMATION BASED ON
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6980-00

ISBN 0738490768