# XCF Concepts and Coding:
# A Multi-system Application Example

Carl Feinberg
Relational Architects Intl
Feinberg.C@relarc.com
http://www.relarc.com

August 5th, 2010
Session 7797

# Abstract

The increasing importance of Sysplex enabled applications requires an understanding of XCF (Sysplex Services for Communication). This session illustrates the coding and usage of XCF services within a simple, multi-system application. The following topics will be discussed:

- XCF Concepts: Sysplex, Group and Member
- XCF Services: Group, Signaling and Status Monitoring
- Member attributes and states
- An overview of XCF macros
- Walk-through the code of the XCF sample application

This presentation is primarily intended for z/OS systems programmers and system level developers. It assumes some familiarity with assembly language programming.

# Agenda

- Introduction
- Sysplex concepts
- Sysplex services for communication
- XCF Communication Services
- Member Attributes
- Overview of XCF macros
- RXC – the sample multi-system application
- RXC Installation

# Introduction

**RXC** is a package developed to demonstrate XCF operation

- RXC operates as either a single or multi-system application
- The RXC Server performs all XCF related services
- The RXC Server operates in Supervisory State and Key 0

The following IBM publications document XCF:

- z/OS MVS Setting Up a Sysplex SA22-7625
- z/OS MVS Programming Sysplex Services Guide SA22-7617
- z/OS MVS Programming Sysplex Services Reference SA22-7618

# Sysplex Concepts

- The **Base Sysplex** is a collection of MVS systems that cooperate, using certain hardware and software products, to process work. IBM introduced the Sysplex (system complex) in September of 1990 and provided XCF as a standard mechanism for inter-processor communications.
- **Cross-system coupling (XCF)** services allow multiple instances of an application or subsystem, running on different systems in a sysplex, to share status information and communicate with each other.
- A **Parallel Sysplex** supports a greater number of systems and significantly improves communication and data sharing among those systems. It is also implies the presence of a physical or simulated Coupling Facility.

**Sysplex + Coupling Facility = Parallel Sysplex**

- The **Coupling Facility** enables high performance multi-system data sharing
- **Coupling Facility Channels** provide high speed connectivity between the coupling facility and the central processor complexes (CPC)
- **Sysplex Timers** synchronize the time-of-day (TOD) clocks of multiple CPC's in a sysplex

# Sysplex Concepts

- **XCF** - Sysplex Services for Communication permits multiple instances of an application to:
  - Inform others of their status (active, failed, etc...)
  - Obtain information about the status of other instances of the application
  - Send messages to and receive messages from each other
- **ARM** - Sysplex Services for Recovery allows an application to:
  - Request an application restart in the event of application or system failure
  - Wait for another job to restart before restarting
  - Indicate its readiness to accept work
  - Request that automatic restart no longer be permitted
- **XES** - Sysplex Services for Data Sharing allow multiple instances of an application running on different systems in a sysplex to implement high-performance, high-availability data sharing by using a coupling facility. Applications can maintain and access data in three types of structures (list, lock, or cache).
  - Share data organized as a set of lists (list structure)
  - Determine whether a local copy of cached data is valid (cache structure)
  - Automatically notify other users when a data update invalidates their local copies of cached data (cache structure)
  - Implement efficient, customized locking protocols, with user-defined lock states and contention management (lock structure)

# Sysplex Services for Communication

- **What is a multi-system application?**

  A multi-system application is a program that has functions distributed across MVS systems in a multi-system environment

- **What is a group?**

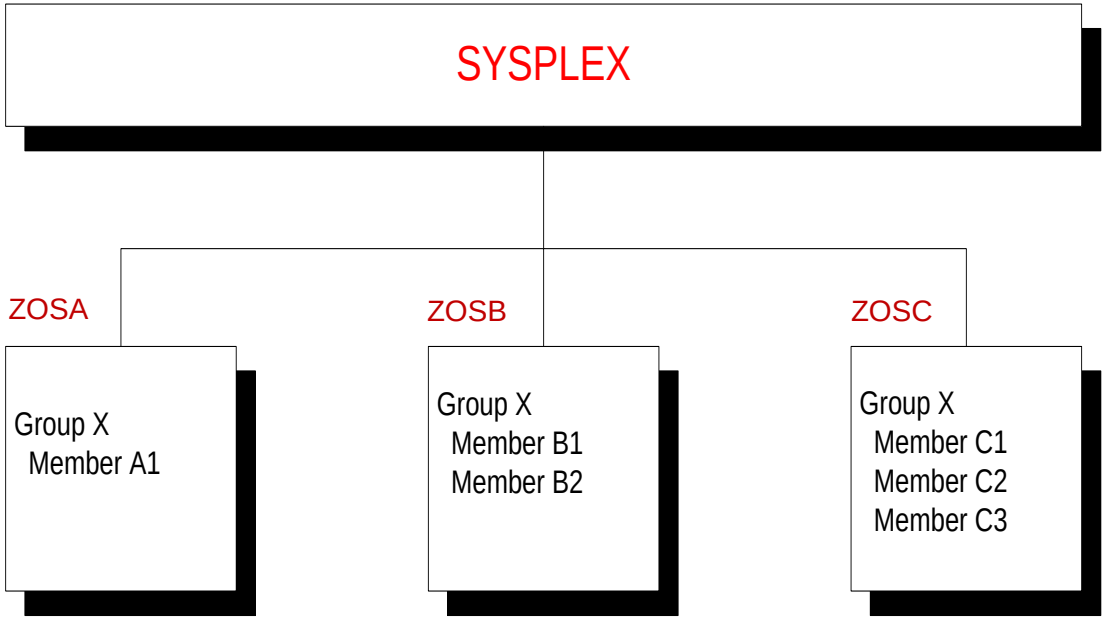
  A group is the set of related members defined to XCF by a multi-system application. A group represents a complete logical entity to XCF

- **What is a member?**

  A member is a specific function of a multi-system application that is defined to XCF and assigned to a group by the multi-system application. XCF allows up to 1023 members in a group.

# Sysplex Services for Communication

## System-Group member relationships in a sysplex



# XCF Communication Services

## *Group services*

- The IXCCREAT macro *defines* a member to XCF. The member is in the "created" state and can later be activated with an IXCJOIN macro
- The IXCLEAVE, IXCQUIES, IXCDELET, and IXCTERM macros *disassociate* members from XCF services
- The IXCSETUS macro *changes* a member's user state value
- The IXCMOD macro changes a member's status-checking interval
- The IXCQUERY macro provides the issuer with information about groups, members, and systems in the sysplex
- The IXCJOIN macro defines a member to XCF and activates it. The following User Exit Routines can be identified: Group, Message, Notify and Status
- The User Group Exit routine can notify group members about changes that occur to members of the group, or systems in the sysplex

# XCF Communication Services

## *Signaling services*

- The IXCJOIN macro lets you identify user routines to do processing on behalf of a member – a Message user exit routine, Group exit and Status exit
- The IXCMSGO macro (message-out service) allows members to *send* messages to other members in their group, as well as to send responses to messages received
- The IXCMSGI macro (message-in service) allows the message user exit routine to *receive* messages from a member and allows the message notify user routine to receive responses from a member
- The IXCMSGC macro (message control service) allows members to save, discard, reprocess or obtain information about messages or responses that have been sent

# XCF Communication Services

## *Status monitoring services*

**provide a way for members to actively participate in determining their own operational status, and to notify other members of their group when that operational status changes. To accomplish this, XCF provides the following:**

- The ability to identify an installation written status user-routine (via the IXCJOIN macro), which determines whether a member is operating normally
- The ability to identify an installation written group user-routine, which allows a member to maintain current information about other members in the group, and systems in the sysplex
- The status user-routine and the group user-routine work together with XCF in the following sense:
  - Specifying a status user-routine, status field, and status checking interval on the IXCJOIN macro causes XCF to begin monitoring a specific field that the member identifies. When the member fails to update the field within the specified time interval, or resumes updating after a failure, XCF schedules the status user-routine to check on the member.
  - When the status user-routine confirms that the member's status changed (either it failed to update its status field or resumed updating), XCF notifies the group user-routines of other members in the group about the change in the member's status. The group user-routines can then take appropriate action.

# Member Attributes

## Members with Permanent Status Recording (P.S.R.)

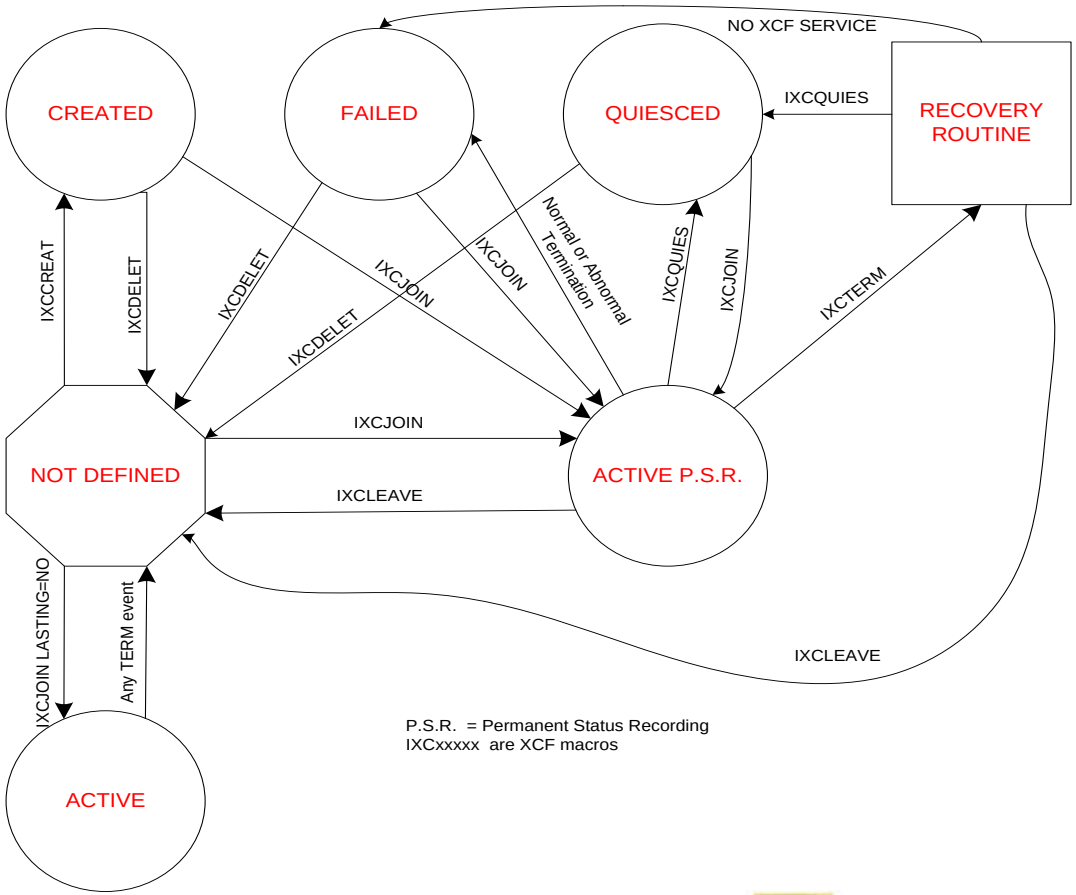
- Maintain a record of the member's existence (including the member's current member state and user state values) even when the member is dormant or has failed.
- Recognizes five discrete states for the member:
  Active, Created, Quiesced, Failed, and Not-Defined
- Use P.S.R. when it is important to know what happened to a member the last time it was running

## Members without Permanent Status Recording

- XCF recognizes only two member states:  Active and Not-defined

# Member Attributes –
## Member states

### Member States



# Member Attributes –
## User Status Field

### User State Field (USF)

A 32-byte user state field is associated with a member. User state information is available to members as follows:

- XCF includes the USF as part of the parameter list it passes to the group user-routines
- XCF includes the USF as part of the data it returns to the caller of the IXCJOIN, IXCCREAT and IXCQUERY macros

### Examples of USF usage:

- As a unique counter
- As a role indicator
- To record steps in a process
- To indicate for a failed member that a restart is in progress
- XCF stores the USF for members with P.S.R., even if they are not in the “Active” state

# Member Attributes –
## Member token

### *The member token*

When you define a member to XCF through IXCCREAT or IXCJOIN, XCF assigns a member token to the member that is unique within the sysplex. The member token that XCF returns on the IXCCREAT or IXCJOIN macros can change. XCF assigns a **new** member token when:

- A created member issues IXCJOIN to become active
- A quiesced or failed member issues IXCJOIN to become active once again
- A member terminates (becomes failed or not-defined) because its associated task, job step task, address space or system terminates and the member is then restarted

Aside from these circumstances, a member token remains the same for the duration of the sysplex. Authorized routines use the member token when requesting XCF services on behalf of a member.

# Member Attributes –
## User Exit Routines

### *The User Routines*

Every member of an XCF group can define one or more of the following user-routines to XCF:

- Message user-routine
- Status user-routine
- Group user-routine
- Message Notify user-routine

# Member Attributes –
## User Exit Routines

### *The Message User-Routine*

The Message user-routine enables an active member of an XCF group to receive messages from other members of the group. Without a Message user-routine, a member cannot receive any messages. The message user-routine can also be used to provide a response to a message, when the member is capable of participating in a protocol that involves sending a response to a sender.

# Member Attributes –
## User Exit Routines

### *The Status User-Routine*

The Status user-routine determines whether a member is operating normally. By identifying a Status user-routine, the member alerts XCF to begin monitoring a field that the member might be updating. If the member fails to update the field within a member-specified time interval, XCF schedules the Status user-routine to determine if a problem exists. (XCF schedules the Status user-routine only for active members.)
If a problem does exist, XCF notifies other active members of the group through their Group user-routines. Code a Status user-routine when you want XCF to monitor the status field of a member.

# Member Attributes –
## User Exit Routines

### *The Group User-Routine*

The Group user-routine enables XCF to notify an active member of a group when there is a change in the operational state of any other member in the group, or a change to the status of any system in the sysplex. If a member does not have a Group user-routine, XCF cannot notify that member of changes that occur.

# Member Attributes –
## User Exit Routines

### *The Message Notify User-Routine*

The Message Notify user-routine enables XCF to notify a sender about the completion of a message. If the sender specifies that a response to a message is required, the Message Notify user-routine can be used to process the collected responses. If the sender specifies that a response is not required, XCF notifies the sender about the status of the message.

# Member Attributes –
## Member Association

### *Member Association*

- Member association is specified via the MEMASSOC parameter on the IXCJOIN macro. This allows you to handle normal and abnormal member termination when IXCDELET, IXCTERM, IXCLEAVE and IXCQUIES macros are not issued.

### *The member remains active:*

- until the task that issued the IXCJOIN macro terminates (MEMASSOC=TASK)
- until the job step task under which the IXCJOIN macro was issued terminates (MEMASSOC=JOBSTEP)
- until the address space in which the IXCJOIN macro was issued terminates (MEMASSOC=ADDRSPACE)

# Member Attributes –
## XCF managed response collection

### *XCF managed response collection*

Your application can request that XCF is to manage the collection of responses to messages you send. Once collected, XCF presents the set of responses to the sender for individual processing. To exploit this feature, both sending and receiving members must reside on systems running the appropriate z/OS level. The sending member, when sending a message, specifies **GETRESPONSE=YES** on the IXCMSGO macro. In addition, the receiving member must have specified **CANREPLY=YES** on the IXCJOIN macro to be able to provide a response.

# Overview of XCF macros

| | | |
|---|---|---|
| **IXCCREAT** | Defines a member to XCF, but the member cannot use signaling and status monitoring services (yet. Possible when “Active”) | Ndef -> Cre |
| **IXCDELET** | Disassociates a “Created”, “Quiesced” or “Failed” member from XCF | Cre,Qui,Fail -> Ndef |
| **IXCJOIN** | Enables a member to join an XCF group. Member status becomes “Active” | Ndef,Cr,Qui, Failed->Act |
| **IXCLEAVE** | Disassociates an “Active” member from XCF | Act->Ndef |
| **IXCMG** | Provides tuning and capacity planning data | N/A |
| **IXCMOD** | Changes the status checking interval | N/A |
| **IXCMSGC** | Message control service | N/A |

# Overview of XCF macros

| | | |
|---|---|---|
| **IXCMSGI** | Receive a message | N/A |
| **IXCMSGO** | Send a message to one or more members | N/A |
| **IXCQUERY** | Return info about members | N/A |
| **IXCQUIES** | Disassociate an “Active” member from the group but retain its P.S.R. information | Active -> Quiesced |
| **IXCSETUS** | Change a member’s User Status Field (USF) | N/A |
| **IXCSYSCL** | Notify the system a member completed cleanup processing | N/A |
| **IXCTERM** | Terminate a member. Abnormally end a task-associated member | Active -> Failed or Not-Defined |

# RXC Software Package –
## RXC Environment

### *RXC environment*

RXC was developed and tested on a 4 LPAR System z:

- ZOSA LPAR-A running z/OS V1.11
- ZOSB LPAR-B running z/OS V1.11
- ICFA Internal Coupling Facility
- ICFB Internal Coupling Facility

Both ICFA and ICFB are configured symmetrically with I/O paths to both the ZOSA and ZOSB systems

# RXC Software Package –
## RXCAPPL Program

### *The RXCAPPL application*

**RXCAPPL reads or writes the first record in the RXCCNTL file**

```
R1  -> OS parameters:
       H'll',CL1'code',CL8'ddname',CL8'record'
       where: ll      = Length of the parameter string
                        (set by the Operating System)
              code    = 'G' Get control record
                      = 'P' Put control record
              ddname  = DDNAME of an allocated file
              record  = contents of record to be updated (put)
```

**The following JCL runs RXCAPPL in standalone mode:**

```
//RXCAPPL EXEC PGM=RXCAPPL,PARM='GRXCCNTL '
//STEPLIB  DD  DSN=RXC.LOAD,DISP=SHR
//RXCCNTL  DD  DSN=RXC.CNTLFILE,DISP=OLD
```

**In multi-system mode, RXCXCF loads and calls RXCAPPL**

# RXC Software Package –
## $USER Dialog

### *$USER is an ISPF dialog to access the RXCAPPL program*

The REXX exec $USER invokes the REXX function RXCUSER to access the RXCAPPL program, either directly or via the RXCXCF server. To start the dialog from ISPF Option 6, type the following command: **EX ‘&HLQ.PDS($USER)’**

```
                    XCF Communications Demonstration Dialog
Command ===> ____________________________________________________________

Application mode . . .  M (S=Single-system; M=Multi-system)

Select desired function _

P - Put (update) record in the control file
G - Get record from the control file
Q - Check if the local RXCXCF server is active
T - Request to terminate the local RXCXCF server

Record Put or Get from control file ________
```

# RXC Software Package – RXCUSER

## *REXX Function – RXCUSER*

RXCUSER is a load module and REXX function called by the $USER exec. Its syntax is:

```
CALL RXCUSER mode, func, record

mode        'S' = Single-system application invocation
            'M' = Multi-system application invocation
func        'G' = Get control record
            'P' = Put (update) control record
            'Q' = Query if RXCXCF is active on the local system
            'T' = Terminate RXCXCF on the local system

Record      a CHAR(8) data record to be used with func='P'
```

The RXCUSER function sets the RESULT and RC variables on return

# RXC Software Package –
## Example of RXCUSER invocation

### Example 1

In this example, the caller receives the control record in the RESULT variable

```
Call RXCUSER 'M','G'
Say 'RC='rc
Say 'Record='result
```

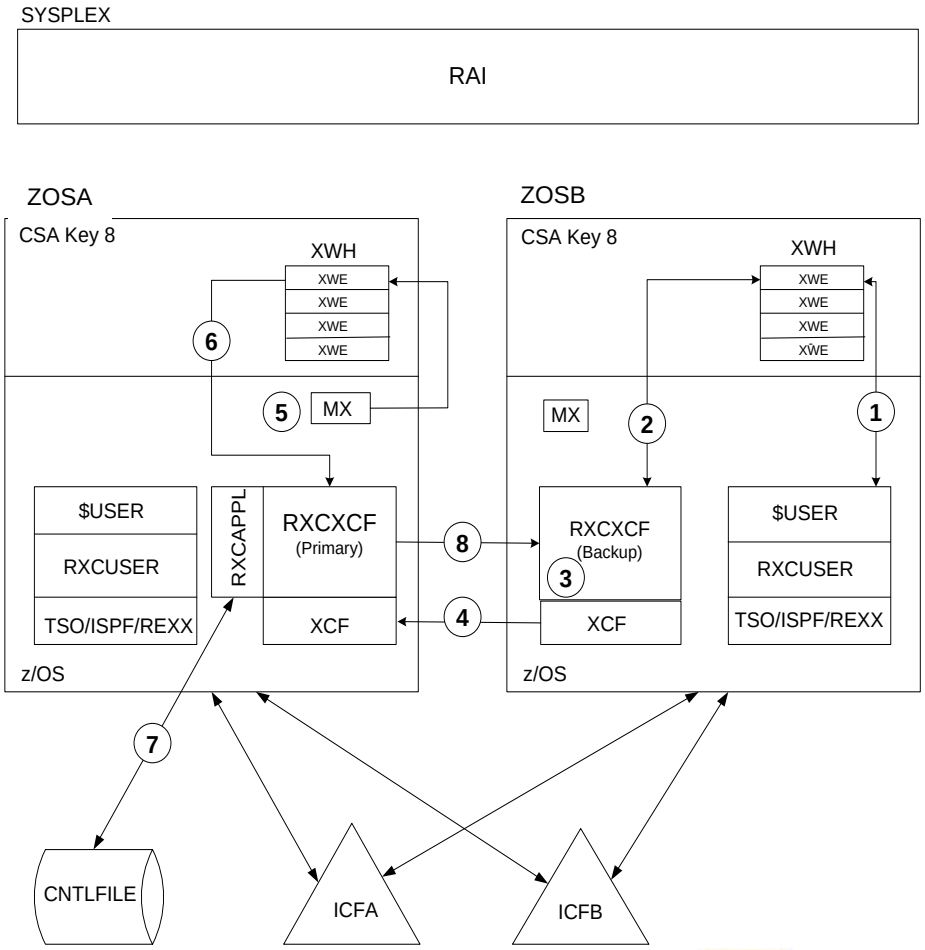
### Example 2

In this example, the caller updates the control record with the value ‘08052010’

```
Call RXCUSER 'M','P',’08052010’
Say 'RC='rc
```

# RXC Software Package

## RXC Remote processing



# RXC Software Package

0. Run RXCXCF on ZOSA and ZOSB as started tasks or submitted jobs
1. The user executes the command: `'EX '&HLQ.PDS($USER)'`

   The user specifies 'Application Mode' as M (multi-system) and function as 'G' to get a control record. The $USER REXX exec issues:

   ```
   CALL RXCUSER 'M','G'
   ```

   The RXCUSER function uses the Name-Token service to locate a pointer to the XWH structure in CSA, key 8. It then reserves a free slot of the XWH area (an XWE control block) and formats it. Next, RXCUSER issues a WAIT macro using the ECB defined in the XWE area.

2. The RXCXCF program monitors the XWH queue for new work
3. RXCXCF detects a new XWE request on the XWH queue. Because it is not the Primary, RXCXCF sends the XWE to the primary system via the IXCMSGO macro.

# RXC Software Package

4. XCF on system ZOSB sends the message to XCF on system ZOSA. XCF on system ZOSA schedules the message user-routine RXCXM through which the incoming message is processed (on behalf of the RXCXCF member on ZOSA).

5. The RXCXM user-routine issues an IXCMSGI macro to receive the message into a local area. It then reserves an XWE slot in the XWH on the ZOSA system and copies the received message into the reserved slot. Finally, RXCXM issues a POST to signal RXCXCF of the new queued element.

6. RXCXCF is dispatched because of the POST issued by RXCXM. The new XWE is used to build a parameter list for RXCAPPL. RXCXCF then calls RXCAPPL to Get a control record.

7. RXCAPPL receives control and reads a control record from the CNTLFILE and then returns the record and control to RXCXCF

# RXC Software Package

8. **RXCXCF on ZOSA issues an IXCMSGO macro to send the completed XWE to the requesting system. XCF on system ZOSB schedules the RXCXM Message user-routine. As in 5, RXCXM places the received XWE on the XWH queue and posts RXCXCF to process the work. Meanwhile, XCF on system ZOSA schedules the RXCXN Message Notify user-routine that will free the processed XWE element.**

   **Finally, RXCXCF on ZOSB is dispatched as a result of the preceding POST and issues a Cross Memory POST for the ECB of the $USER dialog. RXCUSER is dispatched and obtains the returned control record from the XWE. RXCUSER then releases the XWE slot. The control record value is placed in the REXX variable RESULT and RXCUSER returns control to the $USER dialog. Lastly, the $USER dialog displays the returned record on an ISPF panel.**

# RXC Software Package –
## RXCXCF messages

```
>>> Submit job RAI@XCFA on ZOSA
RXCJOIN - Member joined the group - RC=04 Reason=0004
RXCQUERY- Primary is active
RXCXCF  - Member assumes Primary role
RXCXCF  - System Name-Token Created.  XMB=00006E80 XWH=08D2EE88
RXCXG   - Member=ZOSB Type=01 OldSt=00 NewSt=03 FLG2=84 US=B
RXCXG   - NotAct -> Active. Role=B

>>> Execute '&HLQ.PDS($USER)' from ZOSB with func = 'G'
RXCXM   - Msg#=00000024 Flags=000000
RXCXM   - Prepare for MSGI
RXCXM   - Receive successful
RXCXM   - Check mode
RXCXM   - Request
RXCXM   - XWE copied
RXCXM   - REQUEST processed
RXCXM   - Free message block
RXCXM   - POST issued
RXCAPPL - GET Completed
RXCXCF  - RXCAPPL executed
RXCMSGO - ORIGIN=01000005001D0001 TARGET=02000003001D0002 MSG#=00000024
RXCXN   - Enter: Type=01 Flags=0000 Rec#=00000001
RXCXN   - Work Queue Element released
RXCMSGO - IXCMSG0 Issued: Primary -> User (response)
```

# RXC Software Package –
## RXCXCF messages

```
>>> Execute '&HLQ.PDS($USER)' from ZOSA
RXCXS   - Stat=08-*S.U.M.   << Status Update Missing >>
RXCXS   - Stat=00- S.U.R.   << Status Update Resumed >>
RXCXG   - Member=ZOSBType=07 OldSt=03 NewSt=03 FLG2=A4 US=B
RXCXG   - Member=ZOSBType=09 OldSt=03 NewSt=03 FLG2=84 US=B
RXCAPPL - GET Completed
RXCXCF  - RXCAPPL executed
RXCXCF  - XPOST to USER was issued

>>> P RAI@XCFA – shutdown RXCXCF on ZOSA
RXCXS   - Stat=08- S.U.M.
RXCXCF  - Exit EVENTS loop
RXCXCF  - Enter PASSBACK
RXCXG   - Member=ZOSBType=02 OldSt=03 NewSt=03 FLG2=84 US=P
RXCXCF  - Pass Primary role to Backup successful
RXCLEAV - IXCLEAVE completed successfully

>>> RAI@XCFA now inactive
```

# RXC Software Package –
## RXCXCF messages

```
>>> Submit job RAI@XCFB on ZOSB

>>> Execute ‘&HLQ.PDS($USER)’ with func = ‘G’ on ZOSB
RXCQUERY- Primary is active
RXCQUERY- Backup is active
RXCJOIN - Member joined the group - RC=00 Reason=0000
RXCXCF  - System Name-Token Created.  XMB=00006E80 XWH=08DF7E88
RXCMSGO - ORIGIN=02000003001D0002 TARGET=01000005001D0001 MSG#=00000024
RXCMSGO - IXCMSG0 Issued: User -> Primary (request)
RXCXM   - Msg#=00000024 Flags=000000
RXCXM   - Prepare for MSGI
RXCXCF  - XPOST to USER was issued

>>> Execute ‘&HLQ.PDS($USER)’ with func = ‘G’ on ZOSA
no messages – this system is unaware of ZOSA activities
```

# RXC Software Package –
## RXCXCF messages

```
>>> P RAI@XCFA – Stop Primary on ZOSA
RXCXM   - Receive successful
RXCXM   - Check mode
RXCXM   - XWE=00000001808D37F000F8FE80001000C7F0F7F1F6F2F0F0F3
RXCXM   - RESPONSE processed
RXCXM   - Free message block
RXCXM   - POST issued
RXCXN   - Enter: Type=01 Flags=0000 Rec#=00000001
RXCXG   - Member=ZOSA Type=07 OldSt=03 NewSt=03 FLG2=A4 US=P
RXCXG   - Member=ZOSA Type=09 OldSt=03 NewSt=03 FLG2=84 US=P
RXCXS   - Stat=08-*S.U.M.    << Status Update Missing >>
RXCXS   - Stat=00- S.U.R.    << Status Update Resumed >>
RXCXS   - Stat=08- S.U.M.    << Status Update Missing >>
RXCXG   - Member=ZOSB Type=02 OldSt=03 NewSt=03 FLG2=84 US=P
RXCXG   - Member=ZOSA Type=01 OldSt=03 NewSt=00 FLG2=84 US=P
RXCXG   - Active -> NotAct. Role=P
RXCXCF  - Takeover request
RXCXCF  - Member assumes Primary role
>>> P RAI@XCFB – Stop Backup on ZOSB
RXCXS   - Stat=08- S.U.M.
RXCXCF  - Exit EVENTS loop
RXCXCF  - Enter PASSBACK
RXCSETUS- Failed in IXCSETUS macro - RC=04 Reason=0008
RXCLEAV - IXCLEAVE completed successful
>>> RAI@XCFB is inactive
```

# RXC Software Package –
## Console display

```
>>> D XCF,GRP,RXCGROUP,ALL
IXC333I  15.50.46  DISPLAY XCF 659
   INFORMATION FOR GROUP RXCGROUP
   MEMBER NAME:        SYSTEM:     JOB ID:    STATUS:
   ZOSA                ZOSA        RAI@XCFA   ACTIVE
   ZOSB                ZOSB        RAI@XCFB   ACTIVE

 INFORMATION FOR GROUP RXCGROUP MEMBER ZOSA

 MEMTOKEN: 01000004 001D0001      ASID: 0018

 SIGNALLING SERVICE
   MSGO ACCEPTED:          1  NOBUFFER:          0
   MSGO XFER CNT:          1  LCL CNT:          0  BUFF LEN:   956

   MSGI RECEIVED:          1  PENDINGQ:          0
   MSGI XFER CNT:          1  XFERTIME:       4595

   EXIT 02AB8F00: 02/25/2005 15:50:43.195741 MC 00:00:00.000357

 GROUP SERVICE
   EVNT RECEIVED:          3  PENDINGQ:          0

   EXIT 02AD49B0: 02/25/2005 15:49:49.259460 09 00:00:00.000234
```

# RXC Software Package –
## RXC Datasets

| | |
|---|---|
| **&HLQ.PDS** | This distribution library contains the RXC source objects: JCL, REXX exec and assembler language CSECTs and macros |
| **&HLQ.NCAL** | This library contains the assembled RXC CSECTs that are link-edited with the NCAL option. This library is created at installation time from the assembler source modules of the **&HLQ.PDS** library. |
| **&HLQ.LOAD** | This library contains the RXC load modules created by linking members of the **&HLQ.NCAL** library |
| **&HLQ.CNTLFILE** | This sequential dataset is used to store a control record. This library is allocated at product installation time. |

# RXC Software Package –
## Members of the &HLQ.PDS library

### JCL

| | |
|---|---|
| **$$ALLOC** | Allocate the RXC libraries |
| **$$ASM** | Assemble the CSECTS and link-edit the load modules |
| **$APPL** | Execute the RXCAPPL program as a standalone batch job |
| **$XCF** | Execute the RXCXCF server as a batch job |

### JCL Procedures

| | |
|---|---|
| **ALLOC** | Allocate the RXC target libraries |
| **AL** | Assemble an entry point CSECT and link-edit an RXC load module |
| **ALN** | Assemble an RXC CSECT and link-edit it with the NCAL option |

### REXX program

| | |
|---|---|
| **$USER** | A REXX exec and ISPF dialog that invokes the REXX function named RXCUSER as an interface to the RXCAPPL (and RXCXCF) programs |

# RXC Software Package –
## Macros

| | |
|---|---|
| **#ENTRY** | CSECT entry code |
| **#EXIT** | CSECT exit code |
| **#PLIST** | Build parameter list |
| **#SUBENT** | Subroutine entry code |
| **#SUBEXIT** | Subroutine exit code |
| **@DFT** | System defaults |
| **@DSA** | Map of Dynamic Save Area |
| **@DSAEND** | End of Dynamic Save Area |
| **@REGS** | Register equates |
| **@XAP** | Parameters passed to RXCAPPL |
| **@XMB** | RXC Main Anchor Block |
| **@XMC** | Message Control Information (see IXCMSGO) |
| **@XNT** | Name-Token Pair values |
| **@XWE** | Work Queue Element |
| **@XWH** | Work Queue Header |

# RXC Software Package –
## Assembler CSECTS

| | |
|---|---|
| **RXC$DFT** | System defaults |
| **RXCAPPL** | Application to Get or Put a record from the Control file |
| **RXCCREAT** | IXCCREAT macro code |
| **RXCDELET** | IXCDELET macro code |
| **RXCINIT** | RXC initialization and termination |
| **RXCJOIN** | IXCJOIN macro code |
| **RXCLEAVE** | IXCLEAVE macro code |
| **RXCMG** | IXCMG macro code |
| **RXCMOD** | IXCMOD macro code |
| **RXCMSGO** | IXCMSGO macro code |
| **RXCQUERY** | IXCQUERY macro code |
| **RXCQUIES** | IXCQUIES macro code |
| **RXCREL** | Release Work Queue Element (XWE) |
| **RXCRES** | Reserve Work Queue Element (XWE) |
| **RXCSETUS** | IXCSETUS macro code |
| **RXCSYSCL** | IXCSYSCL macro code |
| **RXCTERM** | IXCTERM macro code |
| **RXCUSER** | Load module and REXX function that interfaces with RXCXCF |
| **RXCWTO** | Issue WTO (Write To Operator) messages |
| **RXCXCF** | RXC server and main XCF interface |
| **RXCXG** | XCF Group user exit routine |
| **RXCXM** | XCF Message user exit routine |
| **RXCXN** | XCF Message Notify user exit routine |
| **RXCXS** | XCF Status user exit routine |

# RXC Software Package –
## Load modules

### *Members of the &HLQ.LOAD library*

| | |
|---|---|
| **RXCAPPL** | an application program to Get or Put a control record.<br>Can run standalone or in multi-system mode |
| **RXCXCF** | RXC server and XCF interface |
| **RXCUSER** | REXX function that provides an interface to the RXCAPPL program (via RXCXCF in multi-system mode) |

# RXC Software Package - Configuration

## *RXC configuration defaults*

```
RXC$DFT  CSECT
RXC$DFT  AMODE 31
RXC$DFT  RMODE ANY
         DC    CL8'RXCGROUP'    XCF Group Name
         DC    H'4'             Maximum number of members
         DC    H'10'            Number of Work Queue Elements
         DC    AL4(180*100)     XCF status update interval
         DC    AL4(2*100)       STIMERM wakeup interval
         DC    AL4(1000)        STIMER pops before terminating
         DC    C'Y'             Issue IXCJOIN: Y or N
         DC    CL3' '
         END   RXC$DFT
```

# RXC Installation

***RXC is distributed as a single file (in TSO TRANSMIT command format) whose name is RXC.XMI. The installation steps are as follows:***

1. Choose a High Level Qualifier (&HLQ) for the RXC datasets and a DASD volume (&VOLSER) on which to allocate them

```
&HLQ       = ______________________________
&VOLSER    = _________

Dataset name  . . . : &HLQ.XMIT
Device type . . . . : 3390
Organization  . . . : PS
Record format . . . : FB
Record length . . . : 80
Block size  . . . . : 3120
1st extent blocks . : 90
Secondary blocks  . : 5
```

# RXC Installation

2. Upload the RXC.XMI file to your z/OS system into the previously allocated &HLQ.XMIT dataset. Use BINARY file transfer – ***without data conversion of any kind***

3. Once the file is uploaded, execute the following TSO command:

```
RECEIVE INDSN('&HLQ.XMIT')
```

When prompted:

```
INMR901I Dataset dsname from user-id on N1
INMR906A Enter restore parameters or 'DELETE' or 'END' +
```

Respond:

```
DSN('&HLQ.PDS') VOLUME(&VOLSER)
```

On completion, RECEIVE allocates the &HLQ.PDS dataset on the specified DASD volume

# RXC Installation

### 4. *Create the RXC libraries*

- Modify the JCL in the '&HLQ.PDS($ALLOC)' member
- Specify a valid job card
- Submit the job
- The highest completion code expected is CC 0000

Once the job completes, the following datasets should be allocated:

```
&HLQ.CNTLFILE
&HLQ.LOAD
&HLQ.NCAL
&HLQ.PDS
```

# RXC Installation

5. ***Assemble and link edit the RXC modules***
   - Review and modify the members AL and ALN of '&HLQ.PDS' to specify the &HLQ to be assigned to the RXC datasets
   - Modify the JCL in the '&HLQ.PDS($$ASM)' member
   - Specify a valid job card
   - Modify the PROCLIB DDNAME to reference the '&HLQ.PDS' dataset
   - Submit the job - the highest completion code expected is CC 0004

6. ***APF authorize the &HLQ.LOAD library on each LPAR in the Sysplex***

   The RXCXCF load module requires APF authorization.

   From the MVS console, enter the following command:

```
SETPROG APF,ADD,DSNAME=&HLQ.LOAD,VOLUME=&VOLSER
```

7. ***Once installed, edit and submit the JCL in the $XCF member of &HLQ.PDS on each LPAR in the Sysplex – one as Primary, another as Backup and the rest as User.***

# RXC Installation

**The RXC software and documentation download RXC.ZIP:**

- Source and object code in TSO TRANSMIT format
- This SHARE presentation
- RXC documentation
- From the home page of www.relarc.com, click “Download” on the bottom banner, then click “**XCF Concepts and Coding”**
- Direct link: http://www.relarc.com/index.php?page_id=form_rxc

**Contact:**

**Carl Feinberg**

**Relational Architects Intl**

Feinberg.C@relarc.com

http://www.relarc.com