SAP Legacy Connector (SAPLCO)

Applies to:

SAP Netweaver with DB2 for z/OS .

Summary

SAP Legacy Connector (SAP LCO) is a software package which is intended to support projects in the z/OS environment. SAP LCO makes it possible to communicate synchronously between CICS, IMS or Batch applications on IBM mainframes and SAP systems. With SAP LCO, you can also call simple CICS programs or IMS transactions from SAP programs.

Authors: Rainer Pfister
Dr. Christian Schaefer

Company: SAP AG

Created on: 28 October 2008

Version: 17 December 2012

Author Bio

Rainer Pfister is a development architect in the SAP database platform organization. He studied Computer Science at the Mannheim University of Applied Sciences, and joined SAP in 1987. Since then he has worked on various SAP kernel and communication development projects. Rainer is responsible for System z porting projects. In his role as advocate, he is the central contact for cross-platform related issues of third-party products and acquired companies. He can be reached at mailto:rainer.pfister@sap.com.

Dr. Christian Schaefer is Development Architect at the joint SAP/IBM platform team and has over 23 years of experience in SAP on IBM mainframes. He joined the SAP in 1985. He worked on the interfaces of SAP R/2 to CICS and MVS. He holds a DSC in Theoretical Physics from the University of Heidelberg, Germany. Christian can be reached at christian.schaefer@sap.com .

Typographic Conventions

Type Style	Description
Example Text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation
Example text	Emphasized words or phrases in body text, graphic titles, and table titles
EXAMPLE TEXT	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, *`F2`* or *`ENTER`*.

Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see ***Help on Help*** → ***General Information Classes and Information Classes for Business Information Warehouse*** on the first page of any version of ***SAP Library***.

Table of Contents

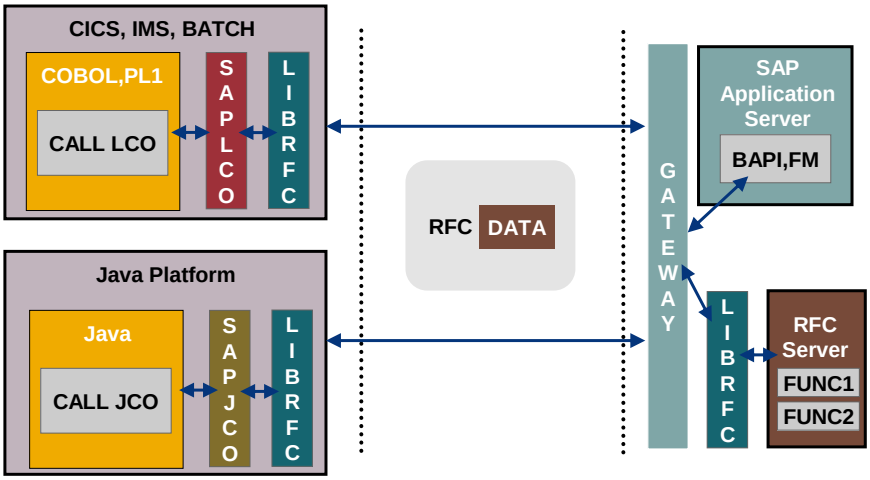
1 Introduction

SAP Legacy Connector (SAP LCO) is a software package which is intended to support projects in the z/OS environment. SAP LCO makes it possible to communicate synchronously between CICS, IMS or Batch applications on IBM mainframes and SAP systems. With SAP LCO, you can also call simple CICS programs or IMS transactions from SAP programs.

The basis of SAP LCO is the **SAP Remote Function Call** (RFC) technology. With this technology, application and metadata are transferred across TCP/IP connections. You can create RFC programs with the **RFC Software Development Kit** (RFC SDK) for C/C++ which is available on all SAP platforms. For more information about the RFC SDK, see the RFC Toolkit.

SAP LCO modules bridge the gap between classic mainframe applications and the **RFC Runtime Library** (`LIBRFC`), since you cannot directly use RFC calls in CICS programs because CICS does not support `POSIX`. The interface uses dynamic program call which allows calls from the programming languages **COBOL** and **PL/I**.

Figure 1: SAP LCO Adapter



1.1 Naming Conventions

This guide uses the following terminology:

Abbreviation	Definition
BAPI	Business Application Programming Interface
`DLL`	Dynamic Link Library
`DPL`	Distributed Program Link (CICS)
`<hlq>`	High Level Qualifier of the MVS Dataset
JCL	Job Control Language
LE	Language Environment (CICS)
LRR	Library Routine Retention
OTE	Open Transaction Environment (CICS)
PID	Process ID
POSIX	Portable Operating System Interface: IEEE Standard for UNIX interfaces for platform compatibility
RCB	RFC Control Block
RMI	Resource Manager Interface
SAP LCO	SAP Legacy Connector
TRUE	Task-Related User Exit (CICS)

2 SAP LCO Installation

2.1 Technical Requirements

You must be sure that the following requirements are fulfilled in your system before you begin installing SAP LCO:

Operating system	z/OS 1.6 or higher
IMS	IMS V7 or higher
CICS	CICS/TS 2.2 or higher
Languages/Compiler	*IBM COBOL for OS/390 & VM V2R2* or *IBM Enterprise COBOL for z/OS V3Rx* *IBM Visual Age PL/I for OS/390 V2R2* or *IBM Enterprise PL/I for z/OS V3Rx*
Security/RACF	You must give access rights for the OMVS segment (UNIX System Services) and the HFS or zFS file system to the user IDs of the CICS startup user, the IMS region jobs and of the batch jobs RACF. This must be done by the RACF administrator.
TSO	This user must have upload and download permissions.
SAP	You must have a developing license for at least one SAP system. The SAP user must have uploading and downloading permissions for ABAP lists.

2.2 Installation

SAPLCO is presently being implemented by SAP Consulting at customer sites. The files necessary to implement this method – those being the load library, configuration examples and documentation – are delivered as a “ZIP” file by SAP Consulting.

2.2.1 Procedure

To install the load library:

1. a) Unzip the packaged LCO file into an empty folder onto a PC.

   b) Transfer the sequential file `LCOV2nn` with FTP (or 3270 file transfer) to the host file system of your choice as a sequential dataset with a fixed record format of 80.
   `nn` stands for a two-digit number, which signifies the current patch level version of SAP LCO.

   
   LCOV210

   c) Use the following FTP commands:

   ```
   > cd '<hlq>'
   > binary
   > quote site fixrecfm 80
   > put LCOV210 LCOV210.seq
   > quit
   ```

2. You must create a z/OS load library from the sequential file `LCOV140`.

   a) Enter the TSO `RECEIVE` command:

   ```
   receive indsname('<hlq>.LCOV210.seq')
   ```

   b) TSO displays the following prompt:

   ```
   Enter restore parameters or 'DELETE' or 'END' +
   ```

   c) At the prompt, enter the name of the target dataset:

   ```
   DSN('<hlq>.LCOV210.LOAD')
   ```

   d) Choose `ENTER`, and the partitioned dataset `<hlq>.LCOV2<nn>.LOAD` is created.

2.3 RACF Options

When using `POSIX` functions during RFC runtime, a user needs to have RACF permissions on the **OMVS Segment**, if this user wants to run LCO programs. The following users are must have these permissions:

- Batch job user
- Startup user of a CICS system
- IMS message region user ID

This security context verifies users attempting to access resources in the z/OS **UNIX System Services** environment: for example, when writing trace data into the HFS or zFS file system on the host. The user also needs a `HOME` directory in this file system, to which files are written by default.

To create the `HOME directory`, enter the following:

```
ALTUSER LCOUSR OMVS(UID(123) HOME('/u/saplco') PROGRAM('/bin/sh'))
```

For more information about OMVS Segment, see the IBM RACF documentation at

http://www.ibm.com/servers/eserver/zseries/zos/racf/.

2.4 Configuring the Batch Environment

To configure the batch environment, carry out the following steps:

1. Add the SAP LCO load library entry as a new line to the `STEPLIB` concatenation of batch programs that contains LCO calls.
2. Define the LCO configuration file `LCOPARM` as follows:

```
//STEPLIB  DD DSN=<hlq>.LCOV2<nn>.LOAD,DISP=SHR
           :
//LCOPARM  DD DSN=..
```

For a complete JCL example, see the section *SAP LCO in IMS and Batch*.

If you have not granted the user OMVS Segment permissions, the resulting errors are collected in the system log. The trace information is written to the normal job log.

For more information about the LCO configuration file `LCOPARM`, see the section *LCO Configuration*.

2.5 Configuring the IMS Environment

To configure the IMS environment, carry out the following steps:

1. Add the SAP LCO load library entry to the `STEPLIB` concatenation of the JCL for the IMS Message Regions in which you want to run LCO programs. You must define the configuration file `LCOPARM`, as in the Batch environment. If the OMVS Segment permission is missing for the IMS Message Region User ID, the trace data is written to the region log. The missing permission is written to the system log.
2. If you want to run a **new** SAP LCO application transaction, additional definitions in the IMS environment are necessary for you to be able to carry out that transaction.
   Additional definitions are not necessary if you want to add LCO access to an existing transaction.
   You can create a test transaction with the SAP LCO program code generator. For more information, see the section *Code Generator*.

Here is an example of the necessary IMS definitions:

IMS generation

```
         APPLCTN PSB=SAPLCOP,SCHDTYP=PARALLEL,
               PGMTYPE=(TP)
         TRANSACT CODE=SAPLCOP,
               PRTY=(05,05,65365),
               MSGTYPE=(SNGLSEG,NONRESPONSE,3),
               PROCLIM=(10,3),
               MODE=SNGL
         APPLCTN PSB=SAPLCOC,SCHDTYP=PARALLEL,
               PGMTYPE=(TP)
         TRANSACT CODE=SAPLCOC,
               PRTY=(05,05,65365),
               MSGTYPE=(SNGLSEG,NONRESPONSE,4),
               PROCLIM=(10,3),
               MODE=SNGL
```

PSB definitions

```
****************************************
*PSB-NAME     : IMSLCOP       PL1 Program
****************************************
         PRINT NOGEN
         PSBGEN LANG=PLI,
               SSASIZE=097,
               CMPAT=YES,
               IOEROPN=451,
               PSBNAME=SAPLCOP
****************************************
*PSB-NAME     : IMSLCOC       COBOL Program
****************************************
         PRINT NOGEN
         PSBGEN LANG=COBOL,
               SSASIZE=097,
               CMPAT=YES,
               IOEROPN=451,
               PSBNAME=SAPLCOC
```

For more information about the LCO configuration file `LCOPARM`, see the section *LCO Configuration*.

2.6 Configuring the CICS Environment

To configure the CICS environment, carry out the following steps:

1. Enter the SAP LCO load library in the `STEPLIB` and `DFHRPL` concatenation of the CICS start job. The `STEPLIB` string makes it necessary to give the SAP LCO load library APF authorization.
2. Define the configuration file `LCOPARM`, with which the parameters of the SAP LCO modules are set. Many of the LCO configuration parameters are expressly used in the CICS environment. For more information about the different parameter values, see the section *LCO Parameters*.

```
//*************************************************************
//*        THE CICS STEPLIB CONCATENATION
//*************************************************************
//STEPLIB  DD DSN=..
//         DD DSN=..
//         DD DSN=<hlq>.LCOV2<nn>.LOAD,DISP=SHR
//*************************************************************
//*        THE CICS LIBRARY (DFHRPL) CONCATENATION
//*************************************************************
//DFHRPL   DD DSN=..
//         DD DSN=..
//         DD DSN=<hlq>.LCOV2<nn>.LOAD,DISP=SHR
//*************************************************************
//LCOPARM  DD DSN=..
```

3. The program `SAPLCOP` reads the configuration file, which is called by the program `SAPLCOI` during the initialization phase. The values found in the file are used to override the predefined standard values that are contained in the load module `SAPLCOZ`.

4. The file `CSDINFO` contains all necessary information about programs and transactions that must be defined in the `CSD` file in the CICS environment. If necessary, you can modify the group `SAPLCO`:

```
DEFINE PROGRAM(SAPLCOC) L(A) GROUP(SAPLCO) DA(A)
       DATALOCATION(ANY)
       DE(Synchronous Call of Interface via SAPLCOIN)

DEFINE PROGRAM(SAPLCOE) L(A) RES(Y) GROUP(SAPLCO) DA(A)
       EXECK(C) DATALOCATION(ANY)
       DE(Task Related User Exit)

DEFINE PROGRAM(SAPLCOI) L(A) GROUP(SAPLCO) DA(A)
       EXECK(C) DATALOCATION(ANY)
       DE(Initialize and Detach LCO environment)

DEFINE PROGRAM(SAPLCOL) L(A) GROUP(SAPLCO) DA(A)
       DATALOCATION(ANY)
       DE(Test Interface - Determine System Info)

DEFINE PROGRAM(SAPLCOM) L(A) GROUP(SAPLCO) DA(A)
       EXECK(C) DATALOCATION(ANY)
       DE(Display LCO Monitoring Data)

DEFINE PROGRAM(SAPLCOP) L(A) GROUP(SAPLCO) DA(A)
       EXECK(C) DATALOCATION(BELOW)
       DE(Read LCO Parameterization File)

DEFINE PROGRAM(SAPLCOZ) L(A) RES(Y) GROUP(SAPLCO) DA(B)
       EXECK(C) DATALOCATION(ANY)
       DE(LCO main anchor - table like function)

DEFINE TRANS(LCOE) PROG(SAPLCOI) PRI(100) ALIAS(lcoe)
       GROUP(SAPLCO) TASKDATAL(A) TASKDATAK(C)
       DE(Detach LCO environment)

DEFINE TRANS(LCOI) PROG(SAPLCOI) PRI(100) ALIAS(lcoi)
       GROUP(SAPLCO) TASKDATAL(A) TASKDATAK(C)
       DE(Initialize LCO environment)

DEFINE TRANS(LCOL) PROG(SAPLCOL) PRI(100) ALIAS(lcol)
       GROUP(SAPLCO) TASKDATAL(A) TASKDATAK(U)
       DE(Synchronous Interface Call via SAPLCOL and SAPLCOIN)

DEFINE TRANS(LCOM) PROG(SAPLCOM) PRI(100) ALIAS(lcom)
       GROUP(SAPLCO) TASKDATAL(A) TASKDATAK(C)
       DE(Display LCO Monitoring Data)
```

3 LCO Interface

3.1 Interface Programs

The communication between an application program and the SAP LCO layer is implemented as a dynamic program call. All necessary parameters are passed by a **control block**.

Depending on the compile options that you are using, there are three different programs available for the dynamic call of the LCO interface that all expect to receive the address of the RCB control block for the parameters.

SAPLCOA

An interface for applications that were complied with the options `DYNAM` and `DLL`. This module is a **Dynamic Link Library** (`DLL`) with the exported function name `SAPLCOA`

For PL/I programs, the IBM PTF `UK14224` influences which module has to be called with the fetch command: either `SAPLCOA` or `SAPLCOB`.

```
DCL SAPLCO EXT ENTRY(POINTER       BYVALUE)
               RETURNS(FIXED BIN(31) BYVALUE)
               OPTIONS(FETCHABLE);
    FETCH SAPLCO TITLE('SAPLCOB');
    RC = SAPLCO( ADDR(RCB) );
```

SAPLCOB

An interface for applications that were complied with the options `DYNAM` and `NODLL`. The module is generated as a fetchable module with the name `SAPLCOB`.

Applications that use the option `NODYNAM` can force a dynamic call with `CALL` by specifying the module name as the content of a variable field with a length of 8, and not as a fixed character string.

```
01   SAPLCO       PIC X(8) VALUE 'SAPLCOB '
       :
CALL SAPLCO USING BY REFERENCE RCB.
```

SAPLCOC

An interface for CICS programs that synchronously calls the LCO interface by a **Program Link**. The `RCB` is directly passed as a `COMMAREA` parameter.

```
  EXEC CICS LINK PROGRAM(SAPLCOC) COMMAREA(RCB)
```

You must carry out the LCO call in the CICS environment with a program link in the local CICS environment. The reason for this method is that the RCB control block uses local addresses.

Beginning with LCO version 2.1, you can also carry out the LCO call by using `CALL` in `SAPLCOB` in CICS programs.

3.2 RFC Control Block

The RFC control block (RCB) is used for calling an RFC API function. The RCB represents the context for the administration of the RFC connection.
You must complete the following fields before calling the RFC API interface:

function	Address of the z-string with the name of the RFC function, for example `RfcCallReceive`. This string must be null-terminated (x'00')
parmcnt	Number of parameters passed for this function
parm	Address of a list with the addresses of each parameter
status	Reserved. You must initialize it with x'00'

The RCB must either be defined by the application that is calling it, or be allocated dynamically before the first call. The structure must be initialized before you use RCB for the first time.

You must neither release nor reinitialize the RCB during the entire transaction, since several fields are used for internal LCO administrative purposes.

The following fields contain return information for the application:

rc/retcode	Return code of the RFC function that was carried out
errtxt	Address of the error message for internal LCO errors
pgmname	Name of the application that is calling the RFC function
attrib(utes)	Pointer to a space that contains the attributes of the connection. This area is described in the section *Connection Attributes*.
statistic	**CICS only:** Address of a space with time stamps that are collected while LCO is processing. The space is described in section *Monitoring SAP LCO in CICS*.

The calling application may not override any other fields. The content of these fields is not relevant for the application, but is used internally.

The following is the structure of the RCB in COBOL and PL/I:

COBOL: RFC Control Block

```
01  RFC-CB.
    05 rcb-eyecatcher                  PIC X(16).
    05 rcb-function                    POINTER.
    05 rcb-parmcnt                     PIC S9(9) BINARY.
    05 rcb-parm                        POINTER.
    05 rcb-retcode                     PIC S9(9) BINARY.
    05 rcb-retaddr REDEFINES rcb-retcode POINTER.
    05 rcb-errtext                     POINTER.
    05 rcb-status                      PIC S9(9) BINARY.
    05 rcb-pgmname                     PIC X(8).
    05 rcb-attributes                  POINTER.
    05 rcb-statistic                   POINTER.
    05 rcb-reserved                    PIC X(240).
```

PL/I: RFC Control Block

```
Define Structure
  1 RFC_CONTROL_BLOCK,
    2 eyecatcher                      CHAR(16),
    2 function                        PTR,
    2 parmcnt                         FIXED BIN(31),
    2 parm                            PTR,
    2 retcode                         FIXED BIN(31),
    2 errtxt                          PTR,
    2 status                          FIXED BIN(31),
    2 pgmname                         CHAR(8),
    2 attributes                      PTR,
    2 statistic                       PTR,
    2 reserved                        CHAR(240);
```

The following examples show RFC in the different supported programming languages. The first example shows a normal RFC call. The second and third examples show the same call with SAP LCO:

RFC SDK C/C++

```
rc = RfcOpenEx( rfc_logon_data, rfc_error_info_ex );
rfc_handel = rc;
```

COBOL

```
01 RfcOpenEx        PIC X(10) value Z'RfcOpenEx'.
      :                 :

set rcb-function     to address of RfcOpenEx
move 2               to rcb-parmcnt
set rcb-parm-ptr(1)  to address of rfc-logon-data
set rcb-parm-ptr(2)  to address of rfc-error-info-ex
perform SAPLCO-CALL
move rcb-retcode     to rfc-handle
```

PL/I

```
DCL RfcOpenEx       CHAR(9)  VARZ INIT('RfcOpenEx');
       :                 :

rcb.function         = Addr(RfcOpenEx);
rcb.parmcnt          = 2;
rcb_parm(1)          = Addr(rfc_logon_data);
rcb_parm(2)          = Addr(errinfox);
rc = SAPLCO_CALL();

rfc_handle           = rcb.retcode;
```

3.3 Connection Attributes

After having established a connection, the RCB field *attributes* contains the address of the structure `RFC_ATTRIBUTES`, which is displayed below. The fields are described in the RFC SDK documentation. Character strings are written as a string ending in x'00' to the output.

The structure contains fields with information about the code pages.
Many SAP systems use SAP code page 1100 (ISO8859-1 or IBM-819), or a Unicode code page. An LCO program that is based on the EBCIDIC version of RFC Runtime uses by default SAP code page 0100 (IBM-273). You can change this by modifying the RFC login data or by making a separate RFC call.

For more information about SAP character strings, see the SAP transaction `SM59`.

Note the following for RFC:
Character-oriented data is automatically converted by the RFC protocol into the relevant code page of the communication partner.

Another field in this structure is `CPIC_Convid`. The session identifier is used to identify the connection between the calling program and the SAP gateway.

In addition, the environment variable `USER` is set. This identifier is passed to the SAP gateway to check the user. You can display this identifier in transaction `SMGW`. In Batch and IMS environments, the job name is used as a user identifier. In a CICS environment, a combination of CICS name and TCB number is used. In the event of an error, this helps to determine which LCO client is active.

This information can be important when analyzing an error.

The following are example structures of `RFC_ATTRIBUTES` in COBOL and PL/I:

Structure RFC_ATTRIBUTES:

```
      *
      *    RFC Attributes   (COBOL)
      *

       01  rfc-attributes.
           02  attr-dest                  PIC X(65).
           02  attr-own-host              PIC X(101).
           02  attr-partner-host          PIC X(101).
           02  attr-systnr                PIC X(03).
           02  attr-sysid                 PIC X(09).
           02  attr-client                PIC X(04).
           02  attr-user                  PIC X(13).
           02  attr-language              PIC X(02).
           02  attr-trace                 PIC X.
           02  attr-ISO-language          PIC X(03).
           02  attr-own-codepage          PIC X(05).
           02  attr-partner-codepage      PIC X(05).
           02  attr-rfc-role              PIC X.
           02  attr-own-type              PIC X.
           02  attr-own-rel               PIC X(05).
           02  attr-partner-type          PIC X.
           02  attr-partner-rel           PIC X(05).
           02  attr-kernel-rel            PIC X(05).
           02  attr-CPIC-convid           PIC X(09).
           02  attr-password-sate         PIC X.
           02  attr-own-codepage-pcs      PIC X(05).
           02  attr-pcs                   PIC X(02).
           02  attr-real-partner-codepage PIC X(05).
           02  attr-progname              PIC X(41).
           02  attr-reserved              PIC X(161).
```

```
/*
 *     RFC Attributes   (PL/I)
```

```
*/

Define Structure
    1 RFC_ATTRIBUTES,
      2 dest                        CHAR(64)  VARZ,
      2 own_host                    CHAR(100) VARZ,
      2 partner_host                CHAR(100) VARZ,
      2 systnr                      CHAR(2)   VARZ,
      2 sysid                       CHAR(8)   VARZ,
      2 client                      CHAR(3)   VARZ,
      2 user                        CHAR(12)  VARZ,
      2 language                    CHAR(1)   VARZ,
      2 trace                       CHAR(1),
      2 ISO_language                CHAR(2)   VARZ,
      2 own_codepage                CHAR(4)   VARZ,
      2 partner_codepage            CHAR(4)   VARZ,
      2 rfc_role                    CHAR(1),
      2 own_type                    CHAR(1),
      2 own_rel                     CHAR(4)   VARZ,
      2 partner_type                CHAR(1),
      2 partner_rel                 CHAR(4)   VARZ,
      2 kernel_rel                  CHAR(4)   VARZ,
      2 CPIC_convid                 CHAR(8)   VARZ,
      2 password_sate               CHAR(1),
      2 own_codepage_pcs            CHAR(4)   VARZ,
      2 pcs                         CHAR(1)   VARZ,
      2 real_partner_codepage       CHAR(4)   VARZ,
      2 progname                    CHAR(40)  VARZ,
      2 reserved                    CHAR(160) VARZ;
```

3.4 Call Sequence

Use the following call sequence to obtain transactional scope of the jobs started by RFC calls in your SAP system:

1. **`RfcOpenEx`**`(...)`
2. RFC call for communication (for example: **`RfcCallReceive`**`(...)`) and to create local memory structures (for example: **`ItCreate, ItAppend, ItDelete`**).
3. **`RfcClose`**`(...)`

As of SAP LCO Version 2.1, you can call another API as your first RFC call with `RfcOpenEx` (see section *LCO Configuration*, parameter `OPEN`).

With `RfcOpenEx`, SAP LCO uses an existing SAP connection with the same login data, or a new connection is created. The session exists until `RfcClose` is reached. If the session pooling option is used (parameter `POOL`), the session continues to exist even after `RfcClose` is reached.

Memory that is allocated with the RFC function (`ItCreate`, ..) must be explicitly released at application level (`ItDelete`,..).

In CICS, you can use `OPEN=YES` to force the system to only allow transactions that begin RFC communication with `RfcOpenEx`. If this is not the case, an error message is displayed and the job is canceled.

The interval between `RfcOpenEx` and `RfcClose` should be as short as possible in the CICS environment, since an LCO TCB is reserved exclusively for the job in question. To serialize the TCBs, a CICS enqueue is

used. This enqueue is set at the first call `(RfcOpenEx)` and released at `RfcClose` (CICS DEQueue). All calls in between are processed by the same LCO TCB. Long processing times can cause bottlenecks in other transactions that also have LCO calls if all TCBs are busy.

As a result of this technology, there must not be a CICS syncpoint between `RfcOpenEx()` and `RfcClose()`. A syncpoint would cancel the serialization of the RFC threads, which can lead to several errors and even cause the transaction to be canceled.

If the calling application does have a CICS syncpoint between `RfcOpenEx()` and `RfcClose()`, SAP LCO must be configured so that CICS holds the CICS Enqueue until the end of the calling task, if an `RfcClose` is not explicitly called. This can have a negative effect on throughput (see section *LCO Configuration*, parameter `ENQT`).

3.5 Storage Protection in CICS (STGPROT)

If you use active storage protection in CICS, many RFC functions can only be partially used, and some not at all. This especially applies to those RFC functions that return addresses from the heap memory of the RFC runtime environment. These include the following:

```
ItGetLine, ItAppLine, ItUpdLine, RfcAllocString/XString
```

Accessing these addresses while storage protection is active causes the transaction to be canceled with a memory protection error (S0C4). In COBOL, `SET` operations from POINTERS to these addresses are allowed – a `MOVE` operation is not.

Exception strings of the functions `RfcReceive` and `RfcCallReceive` are not affected (see RFC RC=2 or 3). SAP LCO runtime copies these strings internally, so that they are located in the address space of the CICS application.

When processing internal tables, you should not directly use the pointer of an address line. Instead, you must copy the data into an existing field or structure.

```
ItGetLine + MOVE    instead of   ItCpyLine
ItAppLine + MOVE    instead of   ItAppLine + ItPutLine
ItUpdLine + MOVE    instead of   ItUpdLine + ItPutLine
```

Another possibility is to use the SAP LCO functions `memcpy` und `strcpy`. Their syntax corresponds to that of the C functions of the same name:

```
memcpy(toAddr,fromAddr,Length)     =  Copy memory
strcpy(toAddr,fromAddr)            =  Copy Z-String
```

You call these functions in the same manner as you would call an RFC function with SAP LCO. The data can be copied from a dynamically allocated LCO memory space to a local application space.

Other supported functions are `putenv(NAME=VALUE)` and `getenv(NAME)`. These functions are used to save (`putenv`) and read (`getenv`) string variables in the environment.

3.6 Multiple Connections to SAP Systems

Note the following if you want to set up more than one RFC connection to different SAP systems:

- You need one RCB for each connection. The RCB must be permanently allocated to its respective connection. SAP LCO maintains each connection through its RCB.
- The RCB must not be released before the end of a task.

3.7 Code Generator

The ABAP utility `ZZLCOGEN` is necessary for facilitating rapid LCO program code generation, which enables SAP interface access. This ABAP program supports automatic generation of SAP structures in COBOL or PL/I syntax, as well as program code for calling your RFC/BAPI function.

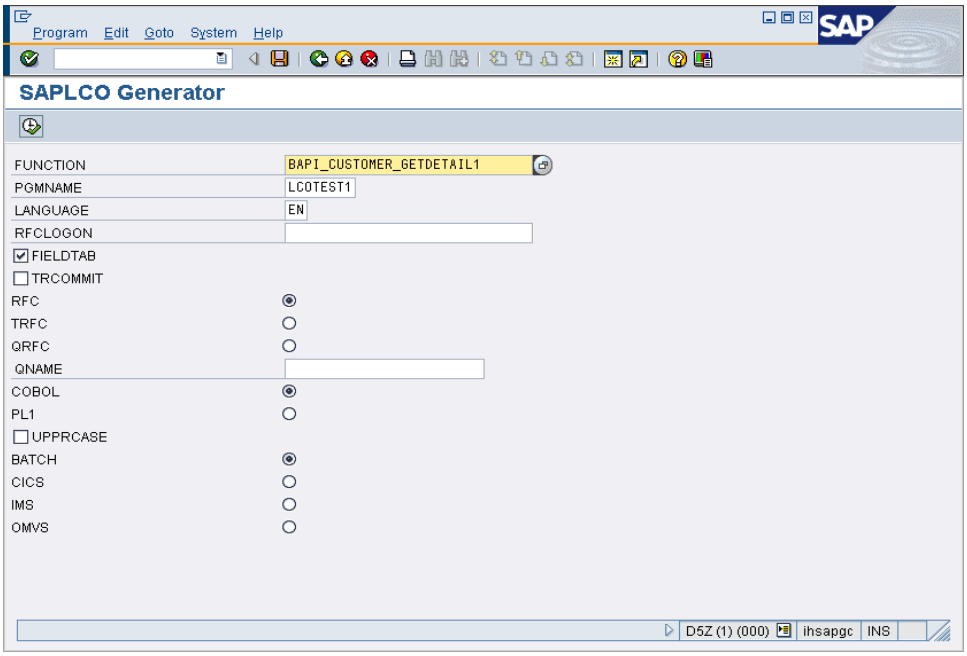
The generator is uploaded as an ABAP program into an existing SAP system. The prerequisites are at least one SAP developer license and the upload/download authorization in the SAP system. Supported languages are COBOL and PL/I.

You can download the generated program code as an ABAP list and save it on a PC. You must complete the specially marked sections of code manually. Some of the data that you must complete is as follows: Login data, as well as the mapping of in- and output data.

You can configure the code generator for your environment as follows:

1. Choose your RFC / BAPI interface:

Figure 2: SAP LCO Generator I



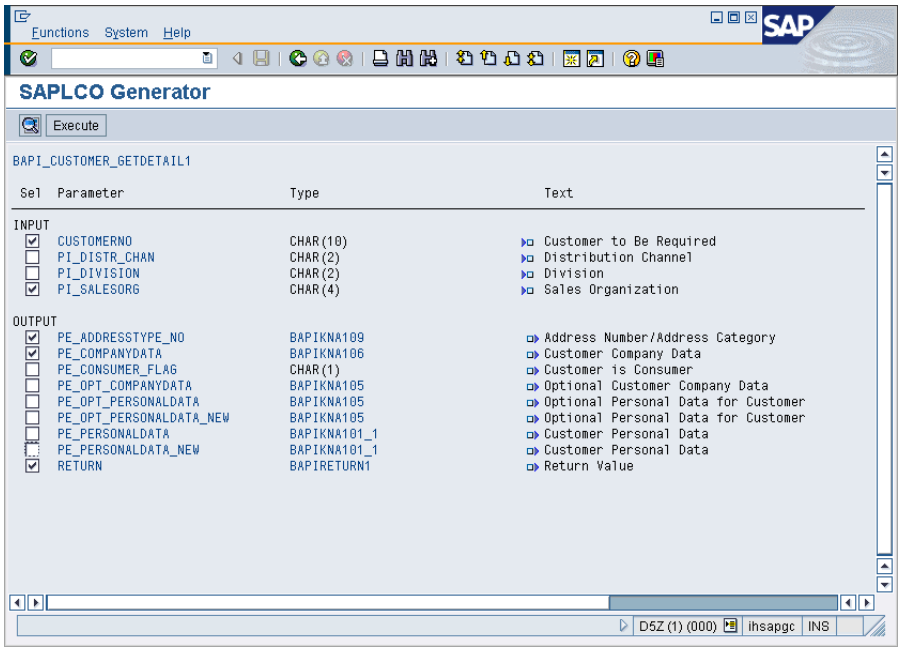
You must complete the following fields:

Field Name	Description
FUNCTION	Name of the RFC/BAPI function module for which code is to be generated. To create a list of options, enter the asterisk (*) and choose `F4`.
PGMNAME	Name of the generated program
LANGUAGE	Language identifier for internal information
RFCLOGON	Identifier for the SAP login data in file `LCOPARM`

FIELDTAB	Generates a description table for SAP structures This table serves as an entry for the RFC function `RfcInstallStructure`
TRCOMMIT	Generates calls for `RFC_TRANSACTION_COMMIT`/`ROLLBACK` Only necessary for RFC/BAPI building blocks that carry out changes in the SAP system
RFC	Synchronous RFC call
TRFC	Transactional RFC call (with transaction ticket TID)
QRFC+QNAME	Queue RFC, like TRFC with serialized processing
COBOL	Generates program code in COBOL
PL1	Generates program code in PL/I
UPPRCASE	Generates program code in uppercase
BATCH	Program type JCL Batch
CICS	Program type CICS
IMS	Program type IMS
OMVS	Program type UNIX System Services

2. Choose the necessary in- and output fields of your SAP function module:

Figure 3: SAP LCO Generator II



Set the indicator next to the fields that you wish to use in the SAP function module. For the purpose of clarity and performance, only select the fields that are essential to your code.

To display the properties of the different fields:

- Click a field name to display the data type in the chosen language.
- Click a type name in a structure to display an overview of the fields in that structure.

A comment line is displayed on the right in the language you have chosen.

3. Generate the program:

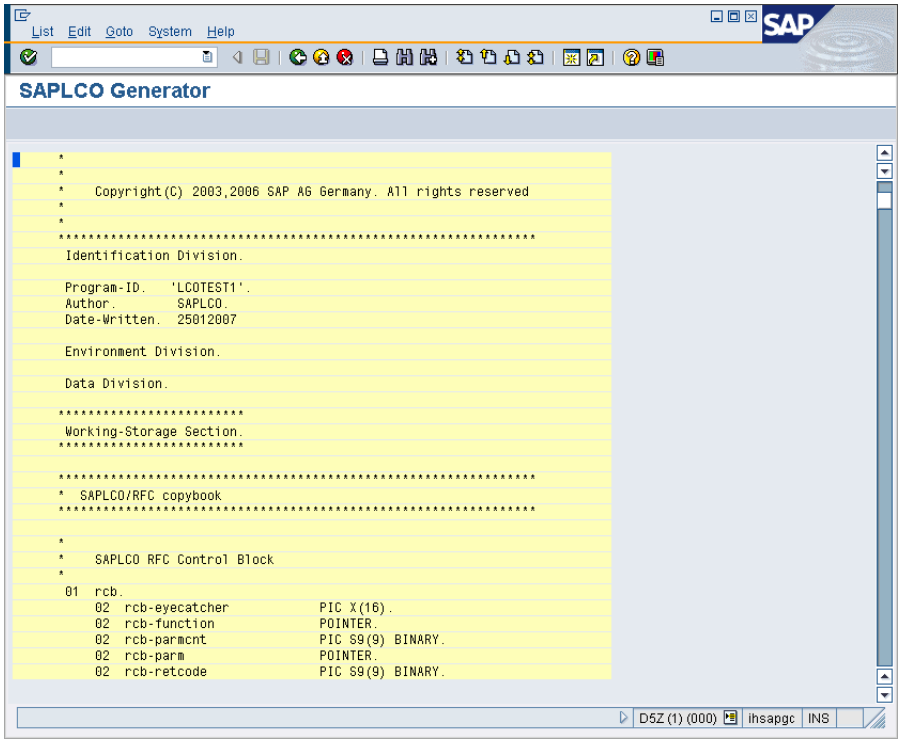
In this step, the code for the function module and the fields chosen in the previous step are created as an ABAP list. You can save the list on your PC as a local file by choosing the first menu item.

Depending on the type of program that you chose, additional calls for in- and output operations are created. This tests the generated program code for the runtime environment.

The following is a list of some of the available program types:

Program Type	Additional Calls
CICS	EXEC CICS SEND / RECEIVE and `COMMAREA` input check
IMS	CEETDLI calls for IOPCB
OMVS	Analogous to Batch with broadened argument checks

Figure 4: SAP LCO Generator – Generated Program Code



4. Completing the Code Manually

   The generated code contains several arrows (→) marking specific areas in the code. You must complete these areas manually. For example:

   - Definition of the in- and output parameters of the program
   - Mapping of the input parameters to the SAP fields/structures
   - Mapping of the received SAP data to the return fields

   If necessary, you must also map SAP error messages to an output structure and adjust the error handling routines to the available infrastructure.

5. Uploading and Converting the Code

   You can now transfer the generated code by uploading it into the IBM mainframe development environment, where you can compile and link it.

3.8 LCO Configuration

3.8.1 Configuration File LCOPARM

You configure the LCO runtime environment with the `LCOPARM` dataset. Depending on your requirements, you use different configuration files for online / Batch mode, or test / production systems.

To define the file in the corresponding environment:

- **OMVS USS Environment**

  You create an ordinary text file for the parameters in a text editor. Set the environment variable `LCOPARM` with the path and the name of the text file:

  ```
  export LCOPARM=/u/saplco/lcoparm
  ```

- **JCL Environment**

  You define the dataset as follows:

- **UNIX file (HFS, zFS)**

```
//LCOPARM   DD PATH='/u/saplco/LCOPARM',PATHOPTS=(ORDONLY)
```

- **In-Stream Dataset**

```
//LCOPARM   DD *
. . .
/*
```

- **Sequential Dataset** (RECFM=FB, LRECL=80, BLKSIZE=80)

```
//LCOPARM   DD DSN=SAP.SAPLCO.LCOPARM
```

- **Partitioned Dataset Member**

```
//LCOPARM   DD DNS=SAP.SAPLCO.PARMS(LCOPARM)
```

3.8.2 LCO Parameter

You must adhere to the following when creating the configuration file:

- You must use the following syntax for the parameters: `<keyword=value>`.
- The keyword must begin in the first column.
- There must not be spaces before or after the equal sign.
- Only one parameter per line is permitted.

Due to the special SAP LCO architecture in CICS (see section *SAP LCO in CICS/TS*) many parameters are only necessary for the CICS environment:

In the following parameter lists, the default values are underlined.

3.8.2.1 CICS-Specific Parameters

NTCB=n Number TCBs in CICS Environment (3-99)

With this parameter you define the number of LCO TCBs that are started in the CICS address space. One TCB is reserved for internal administration, the others are for allotted for processing incoming LCO calls. Each LCO TCB is also a USS UNIX process (`POSIX(ON)`).
The size of the CICS region should be large enough to process RFC calls with dynamic memory requirements.

NTCB=5 1 admin TCB + 4 LCO TCBs

ITC=LCOI Transaction for LCO Initialization (LCOI)

This parameter determines the name of the transaction that carries out the LCO initialization in CICS environment.

Default: **LCOI**

ETC=LCOE Transaction for LCO Shutdown (LCOE)

This parameter determines the name of the transaction with which the LCO TCBs are terminated.

Default: **LCOE**

OPEN=YES|NO First LCO Call of a Transaction

The value of this parameter determines whether the first LCO call of a transaction must be an `RfcOpenEx` – downward compatible to other LOC versions. As of version 2.1, other calls are also permitted.
The first call allocates the transaction to an LCO TCB and serializes this resource with a CICS ENQ call.

ENQL=n Maximal Number CICS ENQs per TCB

With this parameter, you determine the maximum number of CICS transactions that are permitted to wait for processing by an LCO TCB. Once this threshold has been reached, the transaction returns with a negative return code.
This restriction prevents resource problems in CICS that occur if a great number of transactions are waiting for LCO processing. Some reasons for resource problems could be, for example, long-running SAP transactions or other problems with the network, the SAP system or the database.

ENQT=YES|NO Length of CICS ENQ to an LCO TCB

This parameter controls how long a LCO TCB is exclusively reserved for a transaction. Normally, a `RfcClose` call explicitly starts a CICS DEQ, so that the next CICS transaction in the queue can be processed.
A CICS SYNCPOINT call leads to an implicit DEQ. This can cause problems if the Syncpoint takes place before an `RfcClose`.
If the parameter is set to YES, the TCB ENQs are maintained even when the syncpoints have been reached. If no `RfcClose` is used, they can be maintained until the end of the task, the latest.

RESTART=0|n Restart of a TCBs after n Requests

This parameter defines how many RFC calls are necessary before a dynamic restart of LCO-CICS TCB occurs. All LCO calls of the running transaction are completely processed before the restart.
After the restart, unused memory space is deallocated.

IDLELOGOUT=n

This parameter determines – in connection with the parameter `POOL` – how much time should elapse without LCO activity until the SAP system logs off. The session is then removed from the connection pool. You enter the time in seconds.
If the parameter is not explicitly set, the value of the `POOL` parameter is used instead, providing that `POOL > 1`.

IDLELOGOUT=300 Logoff after 300 seconds (5 minutes) idle time

3.8.2.2 General Parameters

TRACE=0|1|2 `Trace Level`

The parameter `TRACE` is used to trace the output for the RFC call and return values.
As of level 2, data in hexadecimal form is returned, as well.
Part of the trace data is information regarding the time necessary for the call. The time is given in microseconds:

1234µs

CICS: A file with the name `RFCLOG.<cics>.<number>` is created in the HFS file system (`HOME` directory of the CICS startup user) for each LCO TCB. This file can also be displayed in an editor in a running CICS system.
Since these files can be quite large, we recommend that you delete them periodically.
If necessary, you can also restrict access to the `HOME` directory!
You can dynamically change the trace level with transaction `LCOM`.

IMS/Batch: The trace output is written to the region protocol or the job log by default. Alternatively, you can use the LCO parameter `RFCLOG` to specify a different file.

RFCLOG=/tmp/SAPLCO.LOG

You can also use the RFC keyword `TRACE` in the login data. However, using `TRACE` creates an additional RFC trace file in the user's `HOME` directory in the HFS file system. The file name has the following syntax: `rfc<number>.trc`.

This `TRACE` should only be activated to analyze RFC problems!

POOL=0|1|n `Activate Connection Pooling`

This parameter defines if an RFC session is closed or is added to a connection pool, resetting the corresponding user context in the SAP system (deallocation of resources in the SAP system).
The next `RfcOpenEx` call with the same parameters receives a free session from the pool, as long as the connection is still valid. If this is not the case, a new session is opened.

This parameter is for frequently recurring connections with the same login data in the online environment. This improves performance, since the time needed for a new user login to the SAP system is saved when this parameter is set.

If you set `n > 1`, this is interpreted as time limit in seconds during which connection pooling is active. The connection is released when the time limit has been reached.

By using RFC load balancing, you can achieve a new cyclic distribution across the available SAP application servers.

POOL=600	Logoff/Logon every 10 minutes

IMS/Batch: Active connections of a connection pool are terminated as soon as the language environment (ENCLAVE) of a program or a transaction ends.

Other strings are read and exported as environment-variables.

```
LOGON_ABC=". . . "
RFC_TRACE=0
```

3.8.3 Example LCOPARM

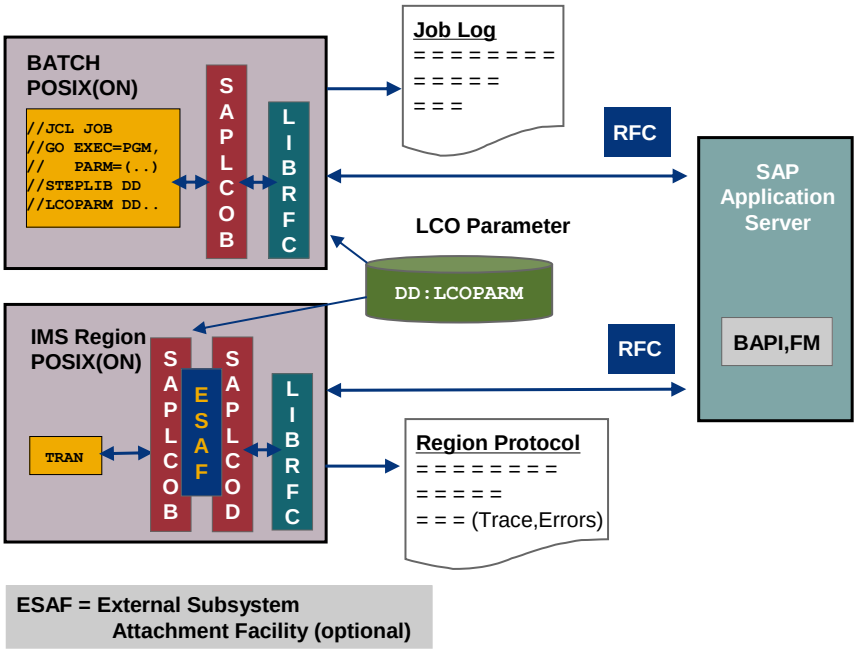
The following is an example of an `LCOPARM` file:

```
EDIT             SAPLCO.LCOPARM
Columns 00001 00080 ****** ***************** Top of Data****
000001 *
000002 *  CICS related LCO parameters
000003 *
000004 NTCB=5
000005 OPEN=NO
000006 ENQL=5
000007 ENQT=NO
000008 ITC=LCOI
000009 ETC=LCOE
000010 *
000011 *  Common LCO parameters
000012 *
000013 LCOOPT=”TRACE=0 POOL=1 IDLELOGOUT=300”
000014 *
000015 *  Logon alias for SAP test system
000016 *
000017 LOGON_T=”ASHOST=<appserver>
000018          SYSNR=01
000019          USER=RFCUSER
000020          PASSWD=xxxxxxxx
000021          LANG=EN
000022          CODEPAGE=0123”
000023 *
000024 *  Logon alias for SAP production system
000025 *
000026 LOGON_P=”R3NAME=PRD
000027          MSHOST=<msgserver>
000028          GROUP=PUBLIC
000029          USER=RFCUSER
000030          PASSWD=xxxxxxxx
000031          LANG=EN”
Command ===>
Scroll ===> CSR
****** ******************************** Bottom of Data ****
```

4 SAP LCO in IMS and Batch

In Batch programs, a dynamic LCO call is initiated by SAPLCOB/A. The RFC runtime call is carried out in the TCB of the Batch job or the IMS region. The LCO interface programs load the RFC runtime DLL first. Subsequently, it loads the RFC API functions contained in the DLL.

Figure 5: LCO Architecture in Batch and IMS



To configure SAP LCO in IMS and Batch:

1. You must configure the LCO load library as a STEPLIB entry. You must set the Language Environment RUNTIME Option `POSIX(ON)` for the runtime environment.
2. You can set runtime options as follows:
   - **JCL Batch**

     You can set `LE` options in the `EXEC` command as part of the parameter `PARM`.

     Note the following syntactical differences between the programming languages:
     - COBOL expects runtime options **after** the slash by default
     - PL/I expects runtime options **before** the slash by default

     The slash serves as a separator from the actual program parameters.
   - **IMS Message Region**

     With the program `CEEROPT`, you can override standard values of runtime options for IMS Regions with **LRR**.

     As of IMS Version 8, it is possible to set runtime options dynamically for programs and transactions (parameter `LEOPT=Y`).

     For more information, see the IBM IMS documentation at

     http://www.ibm.com/software/data/db2imstools/products/ims-tools.html.
   - **In the Program**

     You can set runtime options in PL/I programs as a string with the reserved variable `PLIXOPT`. You must separate all options that you want to override by a space:

```
DCL PLIXOPT CHAR(80) VAR INIT('POSIX(ON)') STATIC EXTERNAL;
```

In COBOL programs, you must use the object module `CEEUOPT` to change program-specific runtime options.

```
CEEUOPT  CSECT
CEEUOPT  AMODE ANY
CEEUOPT  RMODE ANY
         CEEXOPT POSIX=(ON)
         END
```

For more information about runtime options, see the IBM Language Environment documentation at

http://www.ibm.com/servers/eserver/zseries/zos/le/.

- **JCL Example for a Batch Program Call**:

```
//LCOSAMP1 JOB (1,,USER1,,,,),
//             'USER1',CLASS=A,MSGCLASS=X,TIME=NOLIMIT,
//          NOTIFY=USER1
//GO            EXEC  PGM=LCOPGM1,REGION=0M,
//               PARM=('/POSIX(ON)')
//STEPLIB       DD    DSN=<hlq>.LCOV2nn.LOAD,DISP=SHR
//               DD    DSN=SYS1.SCEERUN,DISP=SHR
//               DD    DSN=SYS1.SCEERUN2,DISP=SHR
//SYSOUT        DD    SYSOUT=*
//CEEOUT        DD    SYSOUT=*
//LCOPARM       DD    DSN=<hlq>.LCOPARM
```

5 SAP LCO on CICS/TS

CICS is a transaction monitor with its own multitasking and dispatching within a z/OS address space. It does not use the operating system multitasking, which explains the appearance of tasks in one TCB (QR-TCB). The transactions running on CICS cannot be seen from the «outside».

Access to external resources – for example, a database, sequential files and communication interfaces, etc. – can only take place using special CICS interfaces. As a result, CICS retains the control over the running transactions.

If a CICS transaction does actually access an external resource, this may cause waiting time to occur, which would considerably limit the possible throughput.

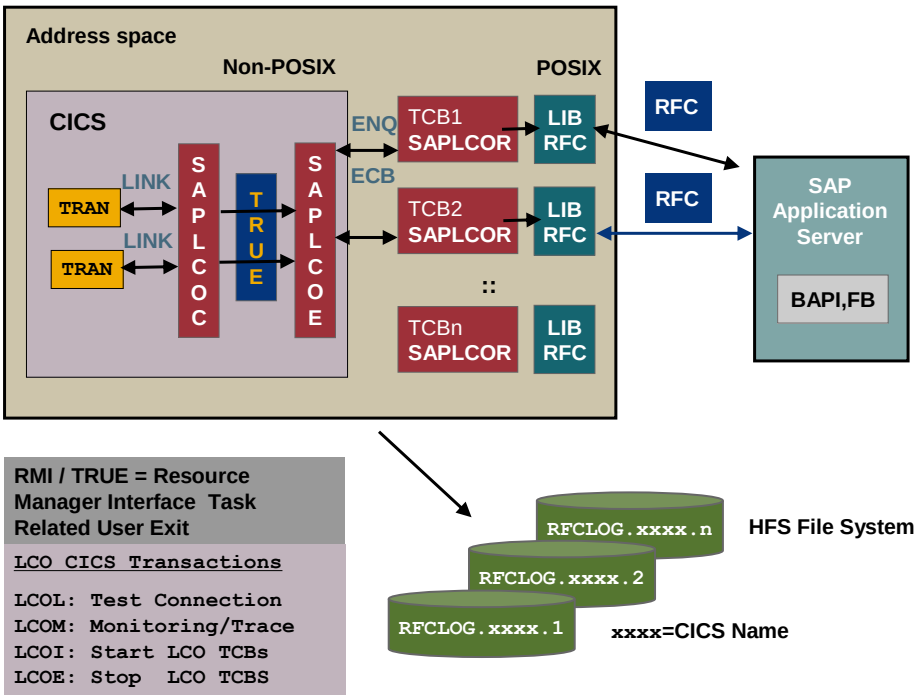
Beginning with CICS/TS V3.1, it is possible to use several TCBs for application transactions – with restrictions – if they were coded according to CICS specifications (OPENAPI support). In many cases, application transactions cannot use this new CICS feature.

5.1 SAP LCO: Subtasks on CICS

The RFC interface of `LIBRFC` is one of the communication interfaces that must not be called directly in `CICS-QR-TCB`. The reason for this is the internal use of POSIX interfaces, which make it necessary to use the runtime option `POSIX(ON)` for the IBM Language Environment. This is not supported in the conventional CICS environment.

Therefore, SAP LCO creates its own z/OS subtasks (TCBs) in the CICS address space with POSIX support. A transaction running in `CICS-QR-TCB` calls `LIBRFC` functions from the LCO interface. This carries out the processing in a subtask TCB.

Figure 6: LCO CICS Architecture



SAP LCO is implemented as a **Task Related User Exit** (TRUE). This ensures the following:

- The control is retained, even when the CICS transactions are aborted
- The ability to have new functions for future use

RFC function calls must take place through the **Resource Manager Interface** (RMI). The RMI is encapsulated in the SAP interface program `SAPLCOE` and is not visible to the CICS application. Therefore, the application logic is strictly separate from the communication module.

SAP cannot use the CICS **Open Transaction Environment** (OTE) for the same reason. SAP LCO cannot run with OPENAPI-TRUE on a L8-TCB, since SAP LCO is not authorized to carry out its own LE initialization on CICS L8-TCBs. L9 TCBs cannot be used for TRUE, either.

RFC calls that are passed to SAP LCO are processed in a UNIX environment in parallel MVS subtasks in the LE TCBs. The LCO dispatcher distributes the work among the available subtasks in such a way as to minimize the number of necessary RFC connections.

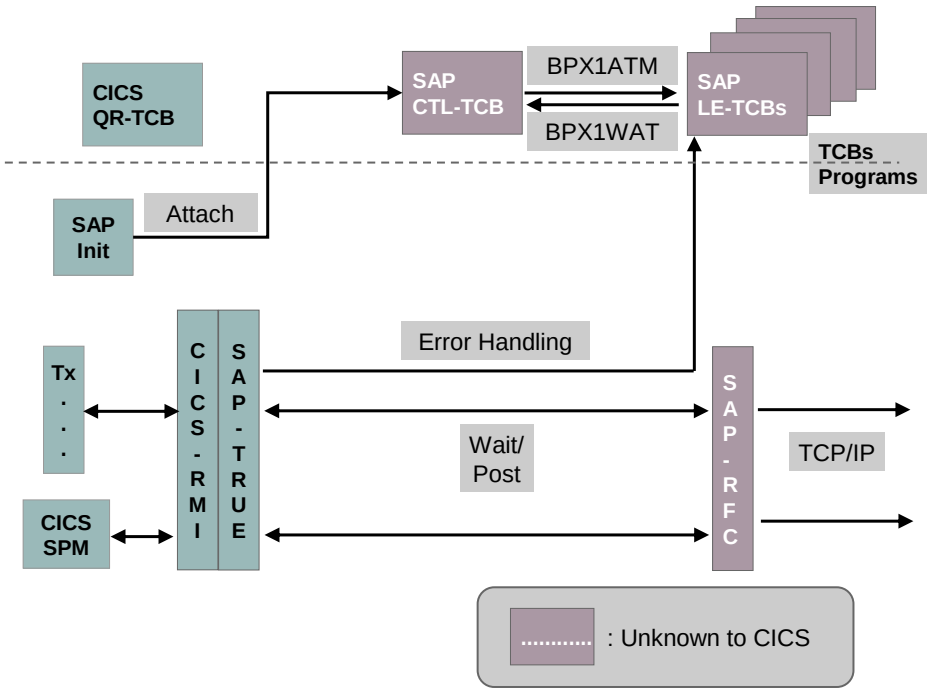
CICS ENQUEUE makes it possible to serialize the processing of the different TCBs. The enqueue argument is a string with the following syntax:

```
TCB000nn
```

where `nn` is the current number of the TCB. By using the enqueue argument, you can track queues of tasks that are waiting for TCBs on an external CICS monitor. If there are too many requests waiting, this means that you have a throughput problem that you need to analyze. In addition to improving the response time on the SAP application servers, you may want to consider increasing the number of available LE-TCBs, as well as the respective RFC connections.

In order to minimize the time exclusively allocated to a LE-TCB for communicating a task, you should keep the communication short. For more information, see the section *Call Sequence*.

Figure 7: CICS: TRUE Implementation of SAP LCO



Configuring the Enqueue Wait

There are two ways to force a timeout as a result of a wait:

1. Set the `DTIMOUT` Parameter in CICS

If you want very long enqueue waits to produce a timeout, set the `DTIMOUT` parameter in the definition of the CICS transaction that is calling SAP LCO. If the time limit set by `DTIMOUT` is exceeded, CICS terminates the transaction with a definition-specific abend code that can be handled by the calling application.

2. Set the `ENQL` Parameter in the SAP LCO File `LCOPARM`

Instead of a definition in CICS, SAP LCO offers its own possiblity to control the number of waiting tasks, and to produce a soft transaction termination in SAP LCO when a limit has been exceed, which is returned to the calling application as a return code. Enqueue Monitoring makes this possible. You can activate this by adding the `ENQL` parameter to the `LCOPARM` file.

`ENQL`'s value is the maximum number of tasks that are permitted to wait for a subtask TCB of SAP LCO. Each additional task is rejected with return code `-16` in the `rc` field of the RCB. The pointer `*errtxt` in the RCB indicates the error message

```
SAPLCOE Max Length of Wait Chain reached,
```

that is also written to the CICS protocol.

Enqueue Monitoring is deactivated by default (`ENQL`=0).

When you set the parameters `NTCB=6` and `ENQL=5`, 5 TCBs are defined for RFC processing that can wait for a maximum of 5 tasks each. Therefore, a maximum of 25 tasks can wait in parallel for RFC requests to be processed.

5.1.1 Initializing LCO

The SAP LCO subsystem is initialized by transaction `LCOI`. If you call this transaction, you can pass the number of TCBs that are to be initialized.

The following is an example call:

```
LCOI 5
```

The number used in the `LCOI` call must be greater than two, but less than 99.

The `LCOI nn` call on a CICS monitor initializes the LCO interface, with `nn` number of TCSs that are independent from CICS, from which `nn-1` can be used for the RFC communication.

A TCB serves administrative purposes and is not taken into account during dispatching. If you do not give `LCOI` a number, the number of TCBs that are started is defined by the `NTCB` parameter in the configuration file `LCOPARM`.
This procedure can be automated by the `CICS PLTPI`. The following is an example for a `PLTSI` (post init) and a `PLTSD` (shut down) definition.

```
//ASM.SYSIN  DD *
         PRINT NOGEN
PLT      DFHPLT TYPE=INITIAL,SUFFIX=01
         DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
         DFHPLT TYPE=ENTRY,PROGRAM=SAPLCOI
         DFHPLT TYPE=FINAL
         END    DFHPLTBA
/*
//LNK.SYSIN DD *
 MODE AMODE(31),RMODE(ANY)
 NAME  DFHPLT01(R)
/*
//ASM.SYSIN  DD *
         PRINT NOGEN
PLT      DFHPLT TYPE=INITIAL,SUFFIX=02
         DFHPLT TYPE=ENTRY,PROGRAM=SAPLCOI
         DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
         DFHPLT TYPE=FINAL
         END    DFHPLTBA
/*
//LNK.SYSIN DD *
 MODE AMODE(31),RMODE(ANY)
 NAME  DFHPLT02(R)
/*
```

```
SIT Entry:
  :
PLTSI=01
PLTSD=02
   :
```

If you initialize in this way, the number of TCBs is also taken from LCOPARM. The SAPLCOI call initializes LCO if there is no available system, or terminates LCO if the system is initialized.

To begin with, the reference value for SAP TCBs can be the number of expected parallel RFC/BAPI calls. Subsequently, you can check the running system with the monitor transaction LCOM to see if CICS ENQ waiting times often develop. The would indicate that too few TCBs are active.

You must adjust the maximum number of possible parallel transactions in CICS (SIT parameter MXT). It does not make any sense to initialize more SAP TCBs than noted in the MXT value.

5.1.2 Terminating LCO

You can terminate SAP LCO with transaction LCOE. This can be done automatically in the PLTSD of CICS.

If you do not terminate SAP LCO correctly, you receive CICS System Abends S33E and/or SA03 during shutdown.

To shut down SAP LCO automatically with PLTSD, you must enter the program SAPLCOI in the PLTSD.

5.1.3 Testing an RFC Connection

You can test an RFC connection to an SAP system with transaction LCOL. This transaction establishes a connection to an SAP system and calls the function module RFC_SYSTEM_INFO, that returns general information like SAP release, database type, among other things, as a string.

You must enter the RFC login data as parameters on the CICS monitor.

```
LCOL ASHOST=srvr SYSNR=21 CLIENT=000 USER=RFCUSER PASSWD=..
```

For more information about possible login parameters, see the RFC SDK documentation.

Alternatively, you can pass a LOGON alias name to the transaction LCOL from the LCOPARM file:

```
LCOL LOGON_T
```

For an example of the LCOPARM file, see the section *LCO Configuration*.

Transaction LCOL saves the last set of login data. If you want to test a certain connection several times, you only have to call LCOL – no additional data is necessary.

If you want to call a connection to a different system, you must enter a complete login string or alias name.

5.1.4 Transactional Integrity

What about transactional integrity?

When a CICS transaction successfully calls a SAP RFC-function or Business API (BAPI) to update data via SAPLCO, the SAP part of the transaction is committed in SAP. If the CICS part of the transaction now abends or the CICS application decides to rollback its work, there is a mismatch between the SAP database and the legacy database.

SAP does not support Two-Phase-Commit (2pc). Therefore SAPLCO cannot ensure transactional integrity in CICS. SAPLCO is a connector only.

There are three phases of a CICS transaction calling SAPLCO:

- Processing before SAPLCO
- SAPLCO processing (call SAP BAPI + call SAP Commit)
- Processing after SAPLCO.

If an error occurs in one of the above phases, the following actions should be taken by the SAPLCO calling CICS transaction to ensure transactional integrity:

Error occurs	Before SAPLCO	During SAPLCO	After SAPLCO
Action	Rollback CICS Transaction, do not call SAPLCO	Rollback SAP, set RC for CICS Rollback	Rollback in CICS and initiate compensating transaction in SAP

This action should be implemented in the legacy application calling SAPLCO.

If the part of the legacy application responsible for the phase ‘after SAPLCO’ for some reason cannot be adapted in this way (i.e. cannot initiate a compensation transaction), the CICS features ‘Temporary Storage’, ‘Transient Data’ or ‘CICS Event Processing’ can be used to check for abends/rollback after calling SAPLCO to initiate compensating work in SAP.

For example, we describe the use of Temporary Storage to check for an abend/rollback during the ‘after SAPLCO’ phase of the transaction. To learn how to use this CICS features, check the CICS Library at: http://www-01.ibm.com/software/htp/cics/cics-library/ .

Select your CICS release and click into the CICS information Center.

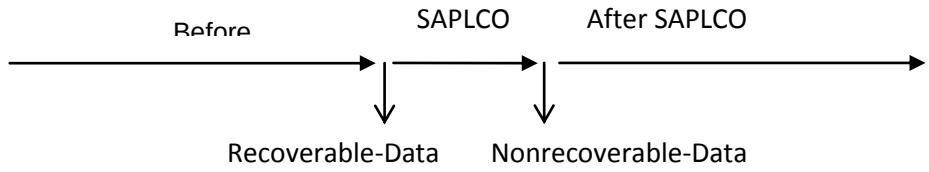
Ensure Transactional Integrity with the Help of Temporary Storage

To detect possible inconsistencies resulting from the ‘after SAPLCO’ processing, two different Temporary Storage Queues can be used:

A **Recoverable TS-Queue**: Data written to this queue can be read, only if the unit of work from which it is written completes successfully.

A **Nonrecoverable TS-Queue**: Data written to this queue can be read regardless of whether the unit of work completes successfully or not. The data in this queue should enable other tasks to schedule compensating transactions in SAP.

First, a log-record is written to the recoverable TS-Queue just before calling the SAP update function via SAPLCO. Second, a log-record is written to the nonrecoverable queue just after the SAP update processing has completed successfully (SAP Commit). For a better understanding see the following graphic:



A periodically running monitoring task may monitor these 2 Temporary Storage Queues. If the SAPLCO calling task completes successfully, there is one record in each queue. If it was not successful, there is only a record in the nonrecoverable TS-Queue, because the entry written to the recoverable queue has been backed out. In this case the monitoring task should initiate a compensating transaction within SAP.

The following two situations require additional attention:

- When the SAPLCO calling task is still running when the monitoring task is started, the nonrecoverable queue may be written but the recoverable queue not, because the entry in this queue can be read only after the SAPLCO calling task reaches a syncpoint or ends successfully. In this case EXEC CICS INQUIRE TASK(data) can be used by the monitoring task to check whether the task is still running.
- The whole construct fails, if for some error it is not possible to write into the nonrecoverable queue when all processing beforehand was successful. So we lose our means to ensure transactional integrity. We suggest to abend the transaction and write necessary information to correct the error to another place (console). Checking for integrity of data and starting a compensating transaction within SAP is to be done by hand.

You can adapt this example to meet your special requirements. It is also possible to use Transient Data or CICS Event Processing. CICS Event Processing is a noninvasing methodology for enhancing business applications. You do not need to modify your application. CICS Event processing in conjunction with WebSphere offers more functionality to check and react on these inconsistency situations.

5.2 Monitoring SAP LCO on CICS

This section deals with questions regarding performance and error situations.

5.2.1 Memory Requirements

LCO administration does not have high memory requirements. 4128 bytes are allocated for each TCB. In addition, 1356 bytes are allocated to each CICS task with LCO calls that are released at the end of the CICS task.

The memory requirement of RFC runtime for the different LE TCBs depends on which functions are called by the application. If large tables are transported, several MB of heap memory may be necessary.

Heap memory, that is used for LE TCBs, is not visible from CICS. In particular, this memory is not included in corresponding displays of CICS monitors that show the memory reserved by a CICS task. The memory requirements of a CICS task displayed on the monitor only contain the 1356 bytes mentioned above that CICS allocates for TRUE. The rest belongs to the calling application.

5.2.2 Monitoring Data

The pointer `*statistic` of RCB points within CICS to an area that contains timestamps which are collected while a SAP LCO call is being processed. The unit of these timestamps is milliseconds since midnight. In SAP systems with SAP BASIS release 6.40 or higher, the processing time of the SAP system in milliseconds is returned, as well.

The fields for SAPLCOE and CICS ENQ are only filled in the CICS environment.

COBOL:

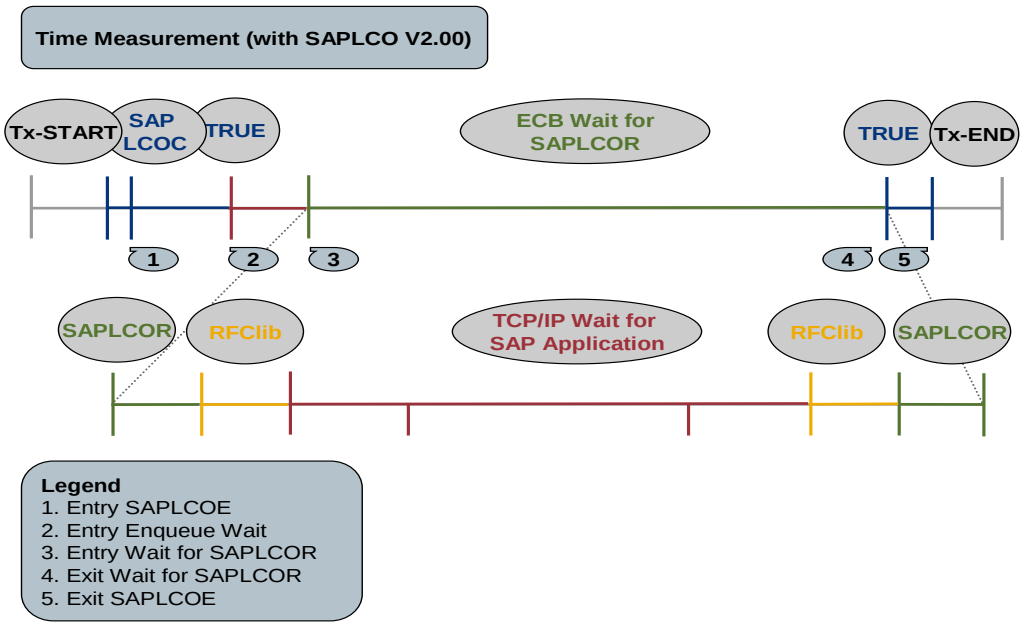
```
01  LCO-STAT-INFO.
    05  lco-stat-len        PIC 9(9) BINARY.
    05  lco-lnk-entry       PIC 9(9) BINARY.
    05  lco-enq-entry       PIC 9(9) BINARY.
    05  lco-ecb-entry       PIC 9(9) BINARY.
    05  lco-ecb-exit        PIC 9(9) BINARY.
    05  lco-lnk-exit        PIC 9(9) BINARY.
    05  lco-sap-runtime     PIC 9(9) BINARY.
```

PL/1:

```
 1  LCO_STAT_INFO,
    2  len                 FIXED BIN(31),  /* length of area     */
    2  time_lnk_entry      FIXED BIN(31),  /* time call SAPLCOE  */
    2  time_enq_entry      FIXED BIN(31),  /* time call CICS ENQ */
    2  time_ecb_entry      FIXED BIN(31),  /* time call SAPLCOR  */
    2  time_ecb_exit       FIXED BIN(31),  /* time exit SAPLCOR  */
    2  time_lnk_exit       FIXED BIN(31),  /* time exit SAPLCOE  */
    2  sap_runtime         FIXED BIN(31);  /* runtime in SAP Sys */
```

The following figure illustrates when the timestamp values are taken:

Figure 8: Request Processing



5.2.3 LCOM Monitoring Transaction

There are two ways to gather statistical data for the purpose of monitoring SAP LCO:

- You can use the SAP transaction `LCOM` to comfortably monitor SAP LCO.
- You can call the program `SAPLCOM` from your own program to receive the same data that is displayed in transaction `LCOM`.

This section describes the transaction `LCOM`. For more information about the program `SAPLCOM`, see the section *The SAPLCOM Program*.

We describe the `LCOM` functions with the corresponding screens.

The CICS system ID is displayed in the lower right corner of each `LCOM` screen. The ID in the following screens is `CI2K`.

The transaction `LCOM` offers the following basic functions:

- Show Parameters from SAP LCO
- Show LE-TCB Activity
- Display Main Storage
- Other useful functions

After calling transaction `LCOM`, you can select these functions on the following screen:

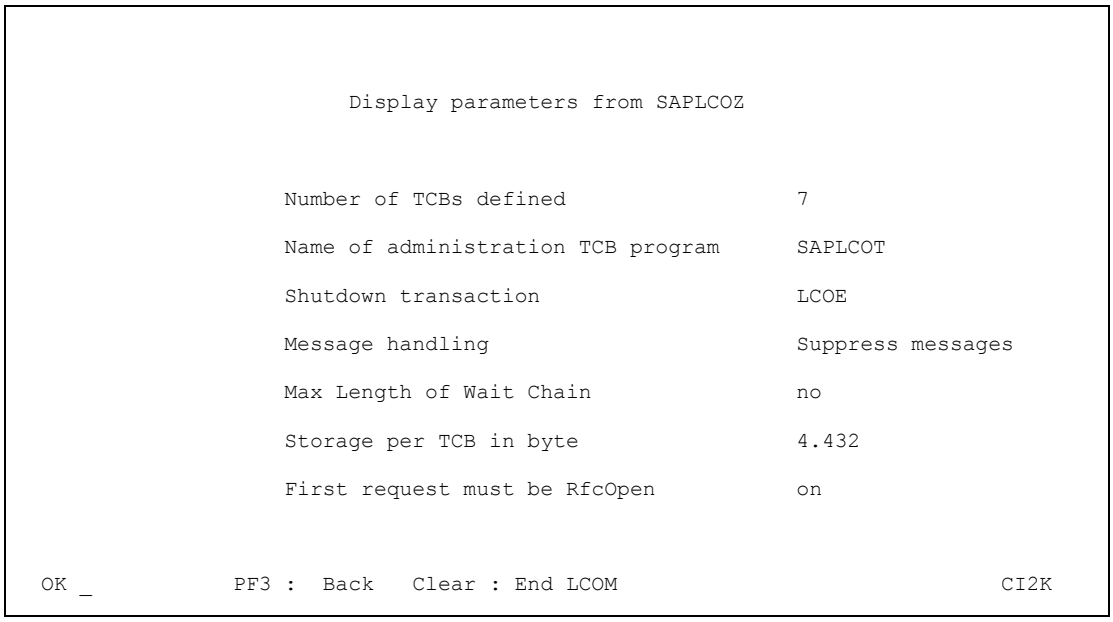
Figure 9: Entry Screen of the Monitoring Transaction LCOM

```
                    LCO Monitoring Transaction LCOM



                PF13      Show Parameters from SAPLCOZ

                PF14      Show TCB Activity

                PF15      Display Main Storage

                PF16      Utilities

                PF17      Show CICS/region parms
--------------------------------------------------------------------------------
OK _                PF1 : Help   Clear : End LCOM                          CI2K
```

Choose **PF13** when you are on the entry screen to obtain the following information (Figure 10):

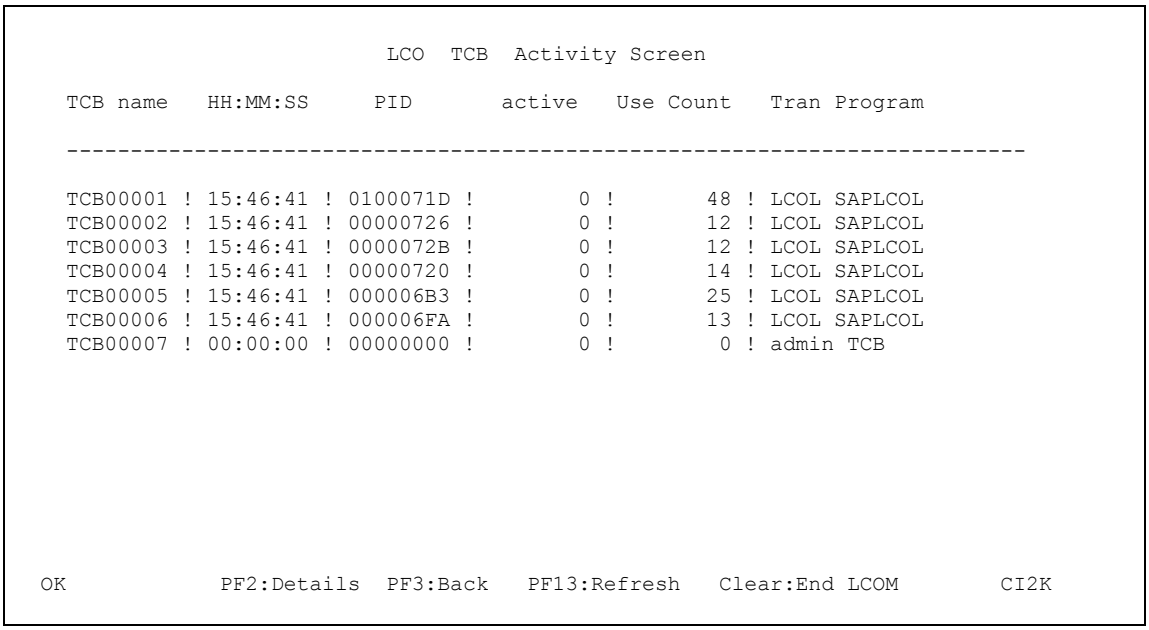
- How many subtask TCBs are available for RFC processing
- Whether the first communication request must be an `RfcOpenEx()`
- The maximum length of the wait chain, if it has been defined

Figure 10: PF13: Show Parameters from SAPLCOZ



Choose **PF14** on the entry screen to display the current LE-TCB activity (Figure 11):

Figure 11: PF14: Show TCB Activity



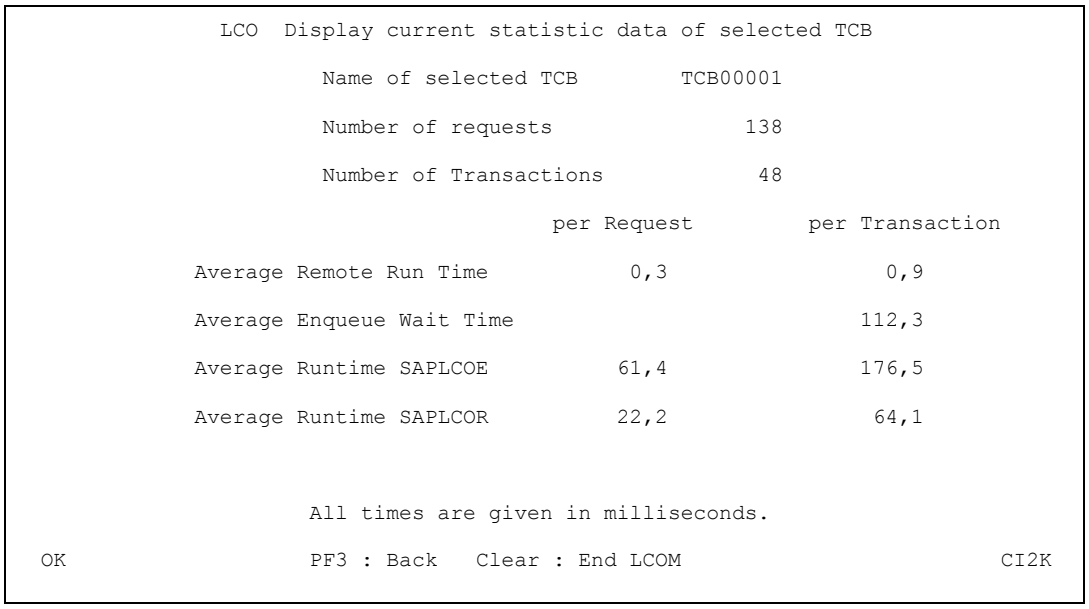
Choose **PF13** to refresh this display. Choose **Enter** to scroll through the list of TCBs: This displays how many tasks are currently allocated to a TCB and how many requests each TCB has processed since SAP LCO was initialized.

One TCB is displayed as an admin TCB. The number of requests that this TCB processed is identical to the frequency with which the LE-TCBs are restarted. In our example, none of the 6 LE-TCBs were restarted. Therefore, the admin TCB has a *Use Count* of 0. In addition, the time of the last processed request is displayed. That is the request that is currently active in a LE-TCB, in case the number of active tasks for this TCB is greater than zero.

The *PID* is the UNIX process identifier.

By placing the cursor in the line of a TCB and choosing **PF2**, you display the current statistical data of that particular TCB on a new screen (Figure 2). The *Average Remote Run Time* is the runtime of the processing in the SAP system. This value is only available if you are using an RFC library version 6.40 or higher.

Figure 12: PF14, PF2: Display Current Statistic Data of Selected TCB



Choose **PF15** to display the *LCOregion Main Storage* screen (Figure 13). This screen is helpful when you want to analyze a problem:

Figure 13: PF15: Display LCOregion Main Storage

```
Display LCOregion Main Storage
 Program  *RCB00002
 OFFset   00000000
 ------------------------------------------------------------------------------
 00000000___000002D0_D3C3D67A_D3C3D6D3_00000000___*....LCO:LCOL....*_251421C8___
 00000010___F2F0F0F7_F1F0F2F3_7AF1F5F4_F6F4F24B___*20071023:154642.*_251421D8___
 00000020   F1F2F040 D9C3C27A 24F00518 00000000   *120 RCB:.0......* 251421E8
 00000030___00000000_00000000_00000000_00000000___*................*_251421F8___
 00000040   C4F5E9F9 F7F0F0F0 33323530 38363035   *D5Z97000........* 25142208
 00000050___D98683C3_9396A285_00000000_00000000___*RfcClose........*_25142218___
 00000060___00000000_00000000_00000000_00000000___*................*_25142228___
 00000070___00000000_00000000_00000000_00000000___*................*_25142238___
 00000080   00000000 00000000 00000000 00000000   *................* 25142248
 00000090   00000000 00000000 00000000 00000000   *................* 25142258
 000000A0___C1E3E37A_8988A281_97928300_00000000___*ATT:ihsapkc.....*_25142268___
 000000B0___00000000_00000000_00000000_00000000___*................*_25142278___
 000000C0___00000000_00000000_00000000_00000000___*................*_25142288___
 000000D0   00000000 00000000 00000000 00000000   *................* 25142298
 000000E0___00000000_008988A2_81979283_00000000___*.....ihsapkc....*_251422A8___
 000000F0   00000000 00000000 00000000 00000000   *................* 251422B8
 ------------------------------------------------------------------------------


 OK _       PF: 1=Help 2=Pickup 3=Back 14=Sel Offset 23=P-                  CI2K
```

You can enter the following in the *Program* field (upper left corner):

- Program Name (for example, `SAPLCOC`): Displays the program in the main memory
- `*tcbnn` (for example, `*tcb03`): Displays the work area of TCB number 3
- `*rcbnn` (for example, `*rcb04`): Displays details about the last processed request in TCB4, including the following:
  - Calling program
  - Target system
  - RFC call

Choose **PF16** on the LCOM entry screen to call other useful functions (Figure 14). These utilities have been preconceived especially for development systems.

Figure 14: PF16: LCO Utility Screen

```
         LCO Utility Screen


               PF13      Switch message handling on

               PF14      Switch "messages to console" on

               PF15      Change number of TCBs to _

               PF16      Switch dumphandling on

               PF17      Switch trace: 1 - 2 - OFF

               PF18      Reset all Connections

               PF19      Change length of wait chain to

               PF20      Switch RfcOpen required  off


--------------------------------------------------------------------------------

OK _               Return:  PF3     Terminate LCOM:  Clear Key             CI2K
```

The following is a summary of the function keys in Figure 14:

Function Key	Function
PF13	Triggers SAP LCO to write more messages about request processing in the CICS protocol
PF14	Writes the above mentioned messages to the console
PF15	If you want SAP LCO to be restarted with a different number of TCB, you can enter this number here for the next SAP LCO startup.
PF16	Triggers a SAP LCO dump in certain error situations (for example: in the event of an invalid `COMMAREA`), to facilitate error analysis.
PF17	Turns on the trace described in the section *LCO Configuration → General Parameters*. By choosing **PF17** more than once, you can switch between all the trace possibilities (trace level 1-2-off).
PF18	Restarts all TCBs. All open connections to SAP systems are closed. This function also releases memory that was implicitly allocated in the RFC layer for the application, but was not released as a result of a missing RFC call in the application (for example, `ItCreate` with a corresponding `ItDelete`).
PF19	Sets enqueue monitoring. The value set here overrides the `ENQL` parameter of the `LCOPARM` file. Changes in this value go into effect immediately. Enqueue monitoring is described in the section *SAP LCO: Subtasks in CICS*.
PF20	Overrides the `OPEN=YES` parameter located in the `LCOPARM` file. For more information about this parameter, see the section *LCO Configuration*.

5.2.4 The Program SAPLCOM

With the program SAPLCOM, you can call up statistical data to monitor SAP LCO.

SAPLCOM, which is located behind transaction LCOM, is DPL-enabled (Distributed Program Link). SAPLCOM can be called from another program with EXEC CICS LINK to make monitoring data available or to set parameters.

The call is activated using a CICS link with a COMMAREA. The following sections display the COMMAREA structure.

5.2.4.1 SAPLCOM COMMAREA Structure

The following is the structure of the communication with the SAPLCOM program using COMMAREA.

COMMAREA to call SAPLCOM	
length	Length of COMMAREA
Request	Function code **Value** **Definition** 1 read TCB data 2 set trace level 3 set length of enqueue wait chain 4 set number of TCBs 5 set Open required 6 reset all threads
Data	data for request with function code **Value** **Definition** 1 number of TCB that is to be read 2 trace level (for example: 0, 1, or 2) 3 new length of enqueue wait chain 4 new number of TCBs value (2-99) 5 value 0, 1
ReturnCode	return code **Value** **Definition** 0 OK 1 invalid TCB number 2 invalid function code 3 invalid trace level 4 program SAPLCOZ cannot be loaded

General data	
Sysid	CICS systemid (char 4)
TcbNumber	Total number of TCBs
WaitChain	Max length of CICS enqueue wait chain for LCO
TraceLevel	Current tracelevel (char 1)
OpenReq	First request must be `RfcOpenEx` (char 1) '0' = NO '1' = YES
Unused	Reserved
Data for returned TCB	
TcbPid	UNIX Process Identifier (PID) of this TCB
Tasks	Current number of CICS tasks this TCB
UseCount	Number of transactions processed
TaskNo	CICS task number currently active
RequestNo	Number of RFC requests processed
EnqWait	Enqueue wait time of the TCB
RntLCOE	Total runtime `SAPLCOE` (CICS+UNIX) (ms)
RntLCOR	Total runtime `SAPLCOR` (UNIX) (ms)
RntSAP	Total runtime in SAP system (ms)
CpuLCOR	Total CPU time `SAPLCOR` (UNIX) (ms)
RcbData	diagnostic data for rcb (1024)

5.2.4.2 COBOL and PL/1 COMMAREA

The following is the structure of `COMMAREA` for COBOL and PL/1.

```
      *
      *  COBOL: COMMAREA to call SAPLCOM
      *
       01  LCOM-CB.
           05 lc-length                      PIC 9(9)  BINARY.
           05 lc-request                     PIC 9(9)  BINARY.
           05 lc-data                        PIC 9(9)  BINARY.
           05 lc-retcode                     PIC 9(9)  BINARY.
           05 lc-sysid                       PIC X(4)  BINARY.
           05 lc-tcbnumber                   PIC 9(9)  BINARY.
           05 lc-waitchain                   PIC 9(9)  BINARY.
           05 lc-tracelevel                  PIC X.
           05 lc-openreq                     PIC X.
           05 lc-unused                      PIC X(2).
           05 lc-tcbpid                      PIC 9(9)  BINARY.
           05 lc-tasks                       PIC 9(9)  BINARY.
           05 lc-usecount                    PIC 9(9)  BINARY.
           05 lc-taskno                      PIC 9(9)  BINARY.
           05 lc-requestno                   PIC 9(9)  BINARY.
           05 lc-enqwait                     PIC 9(9)  BINARY.
           05 lc-rntlcoe                     PIC 9(9)  BINARY.
           05 lc-rntlcor                     PIC 9(9)  BINARY.
           05 lc-rntsap                      PIC 9(9)  BINARY.
           05 lc-cpulcor                     PIC 9(9)  BINARY.
           05 lc-rcbdata                     PIC X(1024).
      /*
       *  PL/1:  COMMAREA to call SAPLCOM
       */
        1 LCOM_CB,
          2 length                          FIXED BIN(31),
          2 request                         FIXED BIN(31),
          2 data                            FIXED BIN(31),
          2 retcode                         FIXED BIN(31),
          2 sysid                           CHAR(4),
          2 tcbnumber                       FIXED BIN(31),
          2 waitchain                       FIXED BIN(31),
          2 tracelevel                      CHAR(1),
          2 openreq                         CHAR(1),
          2 unused                          CHAR(2),
          2 tcbpid                          FIXED BIN(31),
          2 tasks                           FIXED BIN(31),
          2 usecount                        FIXED BIN(31),
          2 taskno                          FIXED BIN(31),
          2 requestno                       FIXED BIN(31),
          2 enqwait                         FIXED BIN(31),
          2 rntlcoe                         FIXED BIN(31),
          2 rntlcor                         FIXED BIN(31),
          2 rntsap                          FIXED BIN(31),
          2 cpulcor                         FIXED BIN(31),
          2 rcbdata                         CHAR(1024);
```

All times are displayed in milliseconds.

6 The SAPLCO RFC Server

As previously mentioned, SAP LCO is an abstraction layer in the RFC runtime environment that makes it possible to use COBOL und PL/I as programming languages for RFC API client calls.

It is also possible to create RFC server programs that contain one or more RFC functions that can be called from an SAP system or an external RFC client program. This type of RFC server function corresponds to an ABAP function module within an SAP system. By using this encapsulation, you can access the mainframe in many different ways with the RFC logic.

However, due to the differences between programming languages, there are many restrictions in the use of the RFC server programming interfaces. These restrictions are discussed in the following sections.

SAP LCO source library contains an example COBOL source of an SAP LCO RFC server (hereinafter referred to as LCO server). You can use the LCO server to call CICS and IMS programs from within SAP.

You cannot use SAP LCO to write an RFC server application as a CICS or IMS transaction.

The following sections contain a general description of the RFC server and the special characteristics of the LCO server.

6.1 RFC Server Program Logic

An RFC server program is a client of the SAP gateway that is part of each SAP application server.

You can activate the server program in the following ways:

- *Started RFC Server*
  After having received an RFC function call, the SAP gateway starts the RFC server – locally or on a remote computer.
  The program ends as soon as the RFC connection is terminated.

- *Registered Server*
  The RFC server program is started and registers itself under a service name with the SAP Gateway. This kind of service can be addressed by several clients. It does not end when the RFC connection is terminated.

The recommended type of an LCO server is the **Registered Server**, since the LCO programs run on an IBM mainframe. In contrast, the SAP gateway is installed on workstations. A registered server performs better, since the time needed to start the server program is no longer necessary.

An LCO server is only supported in Batch and USS (OMVS) environments – not in IMS or CICS dialog environments.

The following is a detailed description of the LCO server example delivered with SAP LCO.

6.2 LCO Server Connection

After the LCO server program has been started, the parameters in the command line are read to establish a **Server Connection** to the SAP gateway of your choice.

In COBOL and PL/I programs, these parameters are passed depending on the environment. There are differences between the JCL batch environment and OMVS/USS shells.

The LCO server program automatically recognizes the runtime environment, reads the parameters and creates a parameter string (C string ending in x'00').

The parameters needed to register the LCO server with the SAP gateway are as follows:

- **`-a <PROGID>`**

  Service name with which the LCO server is registered with the SAP Gateway

- **`-g <HOSTNAME> | <IP-ADDR>`**

  The TCP hostname or the IP address of the server, upon which the SAP gateway is running

- **`-x <PORTNUMBER>`**

  The TCP service name `sapgw<xx>` or the port number `33<xx>`, under which the SAP gateway can be reached, where `<xx>` is the SAP system number (SID).

To connect a server to the SAP gateway, these parameters are passed with the API call `RfcAcceptExt()` to the RFC interface. If successful, this call returns a `connection handle`.

If the connection is not successful, or if there is a technical interruption during operations, the LCO server attempts to carry out a new `RfcAccept` several times.

You can define the number of `RETRY` attempts and their intervals in the program by using the variables `retry-count` and `retry-sleep-sec`.

It is possible to register a server program with more than one gateway. Alternatively, you can also start several instances of the program.

6.2.1 Definition of an RFC Destination in an SAP System

To establish a connection between an SAP ABAP program and the LCO server, you must setup an RFC Destination. This is done with SAP transaction `SM59`. You need the definition of an *TCP/IP Connection* to a server.

1. On the next screen, enter a logical name under which you can address the LCO server as `DESTINATION`. You must enter **`T`** as connection type with a short description. Choose `ENTER` to confirm.
2. On the tab *Technical Setting*s, area *Activation type,* choose *Registered server program* and enter a *Program ID*.

   The LCO Server parameter `-a PROGID` must correspond to the program ID that you enter here.
3. If several SAP servers are available, you can explicitly allocate one of the SAP gateways across which you want this connection to run. To do this, you need the *Host name* and *Port number*, under which the Gateway is available. If you do not enter a value, the system uses the gateway of the SAP server that you are currently using.

   The LCO server parameters `-g Hostname` and `-x PortNr` must correspond to the values that you enter here.

6.2.2 Starting the LCO Server

The following are examples of how an LCO server program is started. It is absolutely necessary in the JCL batch environment to set the LE Runtime Option `POSIX(ON)`. When using the `BPXBATCH` Utility in the USS environment, this option is set by default.

JCL Batch:

```
//LCOSRV  JOB …,…REGION=0M,TIME=NOLIMIT,
          :
//GO      EXEC PGM=LCOSRV,
//             PARM=('-a LCOSRV -g MYHOST -x 3300/POSIX(ON)')
//STEPLIB      DD    DSN=TST.LCOV210.LOAD,DISP=SHR,DCB=BLKSIZE=32760
//             DD    DSN=SYS1.SCEERUN,DISP=SHR
//SYSOUT       DD    SYSOUT=*
//SYSPRINT     DD    SYSOUT=*
//STDOUT       DD    SYSOUT=*
//STDERR       DD    SYSOUT=*
```

JCL with `BPXBATCH`:

```
//LCOSRV   JOB '…',…REGION=0M,TIME=NOLIMIT
             :
//GO       EXEC PGM=BPXBATCH,
//             PARM='-a LCOSRV -g MYHOST -x 3300'
//STEPLIB      DD    DSN=TST.LCOV210.LOAD,DISP=SHR,DCB=BLKSIZE=32760
//             DD    DSN=SYS1.SCEERUN,DISP=SHR
//STDOUT   DD  PATH='/u/myhome/lcosrv.out',
//             PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//             PATHMODE=SIRWXU
//STDERR   DD  PATH='/u/myhome/lcosrv.err',
//             PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//             PATHMODE=SIRWXU
//STDENV   DD  *
… set Environment Variables here when no login shell
//*
```

UNIX System Services Shell (z/OS):

```
sh :  export STEPLIB=TST.LCOV210.LOAD:..
csh:  setenv STEPLIB TST.LCOV210.LOAD:SYS1.CICS.SDFHLOAD:..)
lcosrv -a LCOSRV -g MYHOST -x 3300 &
```

6.2.3 Special Characteristics of the LCO Server

Due to the differences in how parameters are passed to subroutines between different programming languages, the COBOL LCO server does not use the standard APIs `RfcInstallFunction()` and `RfcDispatch()` to register a function and to access it when an RFC request comes in. These functions are only available in the programming language C/C++, since function pointers from C routines are passed as addresses.

Alternatively, the `RfcWaitForRequest()` and `RfcGetNameEx()` are used. This sequence waits for an incoming request and returns the function name as a string to the application.

Then, depending on the name, the function must either be statically implemented in the LCO server or be called dynamically.

An example of a local function is implemented under the names `LOCAL_FUNCTION` and `Local-Function Section` in the LCO server program.

The **dynamic function call** is a special characteristic of the LCO server. If a function name begins with // and is followed by a maximum of seven characters, this name is interpreted as a program name and is loaded and addressed with COBOL CALL.

As a result, you can broaden the available functions of an LCO server, without stopping the server program. Each called program corresponds to exactly to one callable function.

```
CALL FUNCTION '//LCOEXCI'         (or //LCOIMSC)
     DESTINATION 'LCOSERVER'
     SYSTEM = …
```

Another special characteristic is function '`//STOP`'. This function can be called without parameters and terminates the LCO server program.

The LCO modules are only examples. Before you implement them in your production system, you must test them thoroughly with the application.

6.2.4 Communication with CICS: LCOEXCI

An example of a way to implement an SAP RFC server function is `//LCOEXCI`. This is a server module that can be used to call CICS programs dynamically that communicate with a data area (`COMMAREA`) that is limjted to an maximum of 32 KB.

This function corresponds to the **Distributed Program Call** (DPL).The module uses the **CICS External Call Interface** (EXCI) that is addressed by the `DFHXCIS` interface on the mainframe. The EXCI sample resource definition group `DFH$EXCI` must be installed in CICS for a generic connection (CONNECTION and SESSIONS definition).

The LCO server module must either run in the CICS LPAR or in a z/OS environment that allows access to CICS beyond the boundaries of the LPAR. You could access CICS with XCF, for example.

For more information, see the IBM documentation *CICS External Interfaces Guide* (SC34-6449) at http://www.ibm.com/software/htp/cics/tserver.

The RFC import parameters necessary for this function are described as RFC metadata and read as `RfcGetData()`:

```
SYSTEM     C(08): Name of the CICS system
PROGRAM    C(08): Name of the CICS program
TRANSID    C(04): Name of the CICS transaction
USERID     C(08): User ID
DATALENGTH I    : Data length
COMMLENGTH I    : Size of the COMMAREA space
COMMAREA   X(nn): COMMAREA, data transfer area
TRACE      C(01): Trace level
```

The application data is passed with the parameter `COMMAREA` as a binary. Necessary ASCII-EBCDIC conversions before and after the call must be carried out by the caller (for example, in the SAP program).

In addition to a relatively small maximum size of the `COMMAREA`, there is also another restriction. The field *DATALENGTH* only describes the length of the input data. There is no special field for the output data.

A simple solution for this is to use the first two bytes of the `COMMAREA` as a data length field.

After calling the CICS program, the return parameters are described as metadata and returned to the RFC client – for example, an ABAP program – with `RfcSendData()`. The return parameters are as follows:

```
COMMAREA   X(nn): COMMAREA, data transfer area
RETCODE    I    : Return code
RSNCODE    X(04): Reason code
RESP       I    : DPL Return code
ABEND      C(04): CICS Abend code
ERRMSG     C(80): EXCI error message
```

If the fields *RETCODE*, *RSNCODE* and *RESP* are not equal to 0, then an error has occurred. A complete error message is produced in the field *ERRMSG*.

For more information about the error codes, see the IBM CICS documentation (for example, IBM CICS Application Programming Reference) at

http://www.ibm.com/software/htp/cics/

There are two ABAPs delivered with the SAPLCO source that use the LCO server function `LCOEXCI`. You can use them as examples for developing your own ABAPs. For more information, see the section *Examples: From ABAP to CICS or IMS with the SAP LCO RFC Server*.

6.2.5 Communication with IMS: LCOIMSC

An example of a way to implement an SAP RFC server function is `//LCOIMSC`. This is a dynamic server module that can be used to communicate with IMS Connect.

This IBM product is a server that receives specially formatted messages using TCP/IP and passes them to the IMS Open Transaction Manager Access Interface (OTMA). This makes it possible to call the IMS transactions and exchange data.

The LCO server module can run in any LPAR. For the connection to IMS Connect, you need the host name (IP address) and the service port number.

For more information about IMS Connect, see
http://www.ibm.com/software/data/db2imstools/imstools-library.html

The necessary RFC import parameters for this function are described as RFC metadata and are read with `RfcGetData()`:

```
HOSTNAME   C(22): IMS Connect host name
PORT       X(02): IMS Conect port number
SYSTEM     C(08): IMS datastore ID
LTERM      C(08): IMS LTerm
TRANSID    C(08): Name of the IMS transaction *)
USERID     C(08): User ID
PASSWORD   C(08): Password
GROUP      C(08): RACF Group (optional)
TRACE      C(01): Trace level
IRMHDR     X(80): IMS Request Message Header (optional)
INPUT      TABLE: Table with input messages
```

The field *IRMHDR* is optional and contains an 80 byte long IMS request header, with which you can set different options for message processing. If the *IRMHDR* parameter is not passed, the `LCOIMSC` module uses an internal *IRMHDR*.

The input data is passed as segmented messages in table format. IMS Connect expects to receive the incoming message segments in the following format:

```
LLZZinputdata …
```

The last segment of the input data of a transaction must be the identifier EOM (X'00040000').

The output messages of this transaction are then collected in an output table. The last segment is labeled with one of the following special strings: `*CSMOKY*` or `*REQSTS*`.

In this way, the output data of several transactions in the INPUT table can be passed – with an EOM identifier as a separator.

Necessary ASCII/EBCDIC conversions of the input and output segments are to be carried out by the calling program.

The RFC export parameter are sent back to the caller with `RfcSendData()`:

```
RETCODE     I     : Return code
RSNCODE     X(04): Reason code
ERRNO       I     : TCP/IP error number
ERRMSG      C(80): Error message
OUTUT       TABLE: Table with output messages
```

6.3 Examples: From ABAP to CICS or IMS with SAP LCO RFC Server

The source delivered with SAP LCO contains two example ABAPs to communicate with CICS: the ABAPs `ZZEXCI` and `ZZLCOM`. The ABAP `ZZIMSC` communicates with IMS using IMS Connect.

6.3.1 ZZEXCI

In this ABAP example program, the data is transferred in text format. Therefore, the `COMMAREA` is converted from ASCII to EBCDIC. After the call, the `COMMAREA` is converted from EBCDIC to ASCII. The size of the `COMMAREA` is 2048. This can be modified as needed. However, the maximum size is 32767 bytes.

6.3.2 ZZLCOM

This is a more developed variation of `ZZEXCI`, in which the monitoring program of SAP LCO SAPLCOM is called to display statistical data of a chosen LE-TCBs, or to set SAP LCO parameters. This SAPLCOM function is described in the section *The Program SAPLCOM*. Data conversion from EBCDIC to ASCII is implicitly carried out in the Gateway using the definition of the data types of the fields in the `COMMAREA`.

6.3.3 ZZIMSC

In this ABAP example program, the data are transferred in text format. Therefore, the message is converted from ASCII to EBCDIC, and after the call from EBCDIC to ASCII.

7 Appendix

7.1 Error Messages

7.1.1 RFC Error Messages

RFC-specific errors are returned in the RCB field `RETCODE` as positive values.

For more information about RFC error messages, see the `RFC SDK` documentation.

The following values are possible (in COBOL):

```
77  RFC-OK                    PIC S9(9) BINARY VALUE  0.
77  RFC-FAILURE               PIC S9(9) BINARY VALUE  1.
77  RFC-EXCEPTION             PIC S9(9) BINARY VALUE  2.
77  RFC-SYS-EXCEPTION         PIC S9(9) BINARY VALUE  3.
77  RFC-CALL                  PIC S9(9) BINARY VALUE  4.
77  RFC-INTERNAL-COM          PIC S9(9) BINARY VALUE  5.
77  RFC-CLOSED                PIC S9(9) BINARY VALUE  6.
77  RFC-RETRY                 PIC S9(9) BINARY VALUE  7.
77  RFC-NO-TID                PIC S9(9) BINARY VALUE  8.
77  RFC-EXECUTED              PIC S9(9) BINARY VALUE  9.
77  RFC-SYNCHRONIZE           PIC S9(9) BINARY VALUE 10.
77  RFC-MEMORY-INSUFFICIENT   PIC S9(9) BINARY VALUE 11.
77  RFC-VERSION-MISMATCH      PIC S9(9) BINARY VALUE 12.
77  RFC-NOT-FOUND             PIC S9(9) BINARY VALUE 13.
77  RFC-CALL-NOT-SUPPORTED    PIC S9(9) BINARY VALUE 14.
77  RFC-NOT-OWNER             PIC S9(9) BINARY VALUE 15.
77  RFC-NOT-INITIALIZED       PIC S9(9) BINARY VALUE 16.
77  RFC-SYSTEM-CALLED         PIC S9(9) BINARY VALUE 17.
77  RFC-INVALID-HANDLE        PIC S9(9) BINARY VALUE 18.
77  RFC-INVALID-PARAMETER     PIC S9(9) BINARY VALUE 19.
77  RFC-CANCELED              PIC S9(9) BINARY VALUE 20.
```

Error messages from the SAP LCO component are returned to the calling application as negative return codes in RCB. In addition, for CICS-specific LCO errors, a message is written to the CICS protocol.

7.1.2 General LCO Error Messages

Return code	Error	Dump code (if activated with LCOM)
-1	POSIX(ON) Runtime option is missing or LIBRFC not found	-
-16	Max ENQL reached	-
-17	Max. number of open connections reached	SAPLCO15
-27	RfcOpenEx missing	SAPLCO14
-35	COMMAREA exceeded	SAPLCO05
-37	See CICS protocol	-

7.1.3 CICS Abend Codes

If errors occur in the LCO environment, SAP LCO terminates the current transaction with a CICS abend code. Following abend codes are possible:

Abend STCB

The current transaction detects that the control task, which administers the LE-TCBs, has aborted. The task terminates with the abend STCB.

For more information about the cause of the abort, see the CICS protocol.

Abend SAPS

SAP LCO detects that the RCB is damaged and contains invalid data. Processing is aborted and a dump is taken. The validity of the RCBs is checked after the program SAPLCOE has regained control of the RFC layer.

Abend SAPR

SAP LCO internal error. For more information about the errors, see the error messages in the CICS protocol. In addition, the RCB contains a negative return code with the following definition:

-2	Number of parameters of the RFC function too large (max. 10)
-4	Signal SIGILL received (illegal Instruction) (S0C1)
-6	Signal SIGABORT received (abort processing)
-8	Signal SIGFPE received (Floating Point Exception)
-9	Signal SIGKILL received, process terminates immediately
-11	Signal SIGSEGV received (Segmentation Violation) (S0C4)
-205	LIBRFC DLL could not be found
-214	RFC function name unknown

Abend AEY9

If SAP LCO is not initialized, all tasks, which attempt to call SAP LCO, are terminated with a CICS abend AEY9. This abend is generated by CICS itself. SAP LCO is not called by CICS and therefore does not have the possibility to write a more explicative error message.

7.1.4 Error Messages in CICS Protocol

The following errors do not lead to a termination of the transaction, but rather are written to the CICS protocol and to the RCB (error code -16), so that the calling program can continue processing the error message.

SAPLCOE No COMMAREA passed, no processing possible
Without Commarea, no processing is possible.
COMMAREA passed is too short, processing ends
Without Commarea, no processing is possible.
COMMAREA destroyed
The fields in RCB that do not belong to the interface, in which der SAP LCO administers the connection, have been exceeded. Further processing is not possible.
Address of TCB-WA invalid
SAP LCO can no longer access its control blocks. You can try to terminate SAP LCO with `LCOE`, and reinitialize it with `LCOI`. If you fail, you must restart CICS.
Program SAP$$$$ could not be loaded
One of the SAP LCO programs could not be loaded. Check the load libraries and library concatenation.
First Call must be RfcOpen(Ex) instead of $$$$$$$$
Protocol error. The first call of SAP LCO from a transaction must be `RfcOpenEx`.
No more RfcOpen allowed
The current transaction has already made 16 `RfcOpenEx` calls to different SAP systems. The seventeenth `RfcOpenEx` causes the transaction to be terminated with error code -16. If it was activated with LCOM, a dump is taken with code `SAPLCO15`.
General error occurred: $$$$ $$$$$$$$$$$$
During processing, an error occurred (CICS condition code) that was not able to be taken care of by `SAPLCOE`. The error message holds the content of the fields `EIBFN` and `EIBRCODE` of the CICS-EIB (Exec Interface Block). If you obtain these errors, you must inform SAP.
Error while processing EXEC CICS ENQ
An error occurred during SAP LCO enqueue handling. The most probable cause: Due to a CICS syncpoint in the calling application, the CICS enqueue is released to the LE-TCB, even though RFC processing has not been completed.

7.2 Traces and Logs

7.2.1 LCO Log Files

In the event of problems or for the purpose of trace analysis, we recommend that you open the files `RFCLOG.<cics>.<number>` in the `HOME` directory of the CICS user with an editor and check for error messages. For every TCB initialized in CICS, a unique `RFCLOG` file is created. We recommend that you delete these files periodically, since they can become very large.

7.3 LCO Example Program (COBOL)

```
 CBL RENT,DYNAM,NODLL
*************************************
*
*    SAP LCO Sample Program (COBOL)
*
*    (C) SAP AG, 2003, 2007
*
*************************************
 IDENTIFICATION DIVISION.

 PROGRAM-ID. SAPINFO.
 AUTHOR.     SAP.

 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.

 DATA DIVISION.
 FILE SECTION.

 WORKING-STORAGE SECTION.
*
* RFC Request Control Block
*
 01  RCB.
     05 rcb-eyecatcher                    PIC X(16).
     05 rcb-function                      POINTER.
     05 rcb-parmcnt                       PIC S9(9) BINARY.
     05 rcb-parm                          POINTER.
     05 rcb-retcode                       PIC S9(9) BINARY.
     05 rcb-retaddr REDEFINES rcb-retcode POINTER.
     05 rcb-errtext                       POINTER.
     05 rcb-status                        PIC S9(9) BINARY.
     05 rcb-pgmname                       PIC X(8).
     05 rcb-attributes                    POINTER.
     05 rcb-statistic                     POINTER.
     05 rcb-reserved                      PIC X(240).
*
*    LCO Parameter Array
*
 01  RCB-PARM-ARRAY.
     02  rcb-parm-fields OCCURS 8 TIMES.
         03  rcb-parm-val          PIC S9(9) BINARY.
         03  rcb-parm-ptr REDEFINES rcb-parm-val POINTER.
*
* RFC Error Info Structure (extended)
*
 01 RFC-ERROR-INFO-EX.
    05 einfo-group                        PIC X.
    05 einfo-key                          PIC X(33).
    05 einfo-message                      PIC X(513).

 01 RFC-EXCEPTION-PTR                     POINTER.
*
* RFC Parameters for Export
*
 01 RFC-EXPORT-TABLE.
    05 RFC-EXPORT   OCCURS 10 TIMES.
       10 exp-name                        POINTER.
       10 exp-nlen                        PIC 9(9) BINARY.
       10 exp-type                        PIC 9(9) BINARY.
       10 exp-leng                        PIC 9(9) BINARY.
       10 exp-addr                        POINTER.
*
* RFC Parameters for Import
*
 01 RFC-IMPORT-TABLE.
    05 RFC-IMPORT   OCCURS 10 TIMES.
       10 imp-name                        POINTER.
       10 imp-nlen                        PIC 9(9) BINARY.
```

```
       10 imp-type                        PIC 9(9) BINARY.
       10 imp-leng                        PIC 9(9) BINARY.
       10 imp-addr                        POINTER.
*
* RFC internal tables
*
 01 RFC-TABLES-TABLE.
    05 RFC-TABLES   OCCURS 10 TIMES.
       10 tab-name                        POINTER.
       10 tab-nlen                        PIC 9(9) BINARY.
       10 tab-type                        PIC 9(9) BINARY.
       10 tab-leng                        PIC 9(9) BINARY.
       10 tab-ithandle                    PIC 9(9) BINARY.
       10 tab-itmode                      PIC 9(9) BINARY.
       10 tab-newitab                     PIC 9(9) BINARY.

 77  RFCTYPE-CHAR                 PIC S9(9) BINARY  VALUE 0.

 01 rfc-handle                            PIC S9(9) binary.
 01 connect-param                         PIC X(255).

 01 RFCSI.
    05 rfcsi-buffer                       PIC X(200).
*
*  Strings
*
 01 RfcOpenEx               PIC X(10) value Z'RfcOpenEx'.
 01 RfcClose                PIC X(9)  value Z'RfcClose'.
 01 RfcCallReceive          PIC X(15) value Z'RfcCallReceive'.
 01 RFCSI-EXPORT            PIC X(13) VALUE Z'RFCSI_EXPORT'.
 01 RFC-SYSTEM-INFO         PIC X(16) VALUE Z'RFC_SYSTEM_INFO'.

 LINKAGE SECTION.

 01 RFC-EXCEPTION.
    05 exception-c                        PIC X(40).

 PROCEDURE DIVISION.

**************
 MAIN SECTION.
**************
     PERFORM RFC-INIT
     PERFORM RFC-OPEN
     IF rfc-handle > 0 THEN
        PERFORM RFC-CALL-RECEIVE
        PERFORM RFC-CLOSE
     end-if
     GOBACK.

******************
 RFC-INIT SECTION.
******************
*
*    init control block, LogonData-String
*
     initialize RFC-HANDLE
     move all LOW-VALUE to RCB
     move all LOW-VALUE to RFC-ERROR-INFO-EX
     move all LOW-VALUE to connect-param

     string "ASHOST=hostname "
            "SYSNR=00 "
            "CLIENT=000 "
            "USER=RFCUSER "
            "PASSWD=XXXX "
            "LANG=EN"
            X'00' delimited by size
            INTO connect-param
     end-string
*
*    parameter array: val/ptr for each RFCAPI Call
*
     set rcb-parm to address of RCB-PARM-ARRAY
```

```
     .
     exit.

******************
 RFC-OPEN SECTION.
******************
*
*    * Open Rfc Connection to R/3
*
*    C: rfchandle = RfcOpenEx( connectparam, &errinfo );
*
     set rcb-function    to ADDRESS of RfcOpenEx
     move 2              to rcb-parmcnt
     set rcb-parm-ptr(1) to ADDRESS of connect-param
     set rcb-parm-ptr(2) to ADDRESS of RFC-ERROR-INFO-EX
     perform RFC-CALL
*
*    * save Rfc handle, if no handle -> error msg
*
     move rcb-retcode to rfc-handle
     .
     EXIT.

******************
 RFC-CLOSE SECTION.
******************
*
*    * Close RFC Connection
*
*    C: RfcClose( rfchandle );
*
     set rcb-function  to ADDRESS of RfcClose.
     move 1            to rcb-parmcnt.
     move rfc-handle   to rcb-parm-val(1).
     perform RFC-CALL.

     EXIT.

*************************
 RFC-CALL-RECEIVE SECTION.
*************************
*
*    * Call Function 'RFC_SYSTEM_INFO'
*
*    C: RfcCallReceive( rfchandle,
*                       "RFC_SYSTEM_INFO",
*                        &export,
*                        &import,
*                        &table,
*                        &exception );
*
     initialize RFC-EXPORT-TABLE
     initialize RFC-IMPORT-TABLE
     initialize RFC-TABLES-TABLE
*
*    describe import parameter RFCSI
*
     set     imp-name(1) to ADDRESS of Z-RFCSI-EXPORT
     compute imp-nlen(1) =   LENGTH of Z-RFCSI-EXPORT - 1
     set     imp-addr(1) to ADDRESS of rfcsi-buffer
     compute imp-leng(1) =   LENGTH of rfcsi-buffer
     move RFCTYPE-CHAR to imp-type(1)
*
*    set next imp/exp/tab entries to NULL
*
     set  imp-name( 2 ) to NULL
     set  exp-name( 1 ) to NULL
     set  tab-name( 1 ) to NULL

     SET  rcb-function  to ADDRESS of RfcCallReceive
     move 6             to rcb-parmcnt
     move rfc-handle    to rcb-parm-val(1)
     set  rcb-parm-ptr(2) to ADDRESS of RFC-SYSTEM-INFO
     set  rcb-parm-ptr(3) to ADDRESS of RFC-EXPORT-TABLE
```

```
           set  rcb-parm-ptr(4) to ADDRESS of RFC-IMPORT-TABLE
           set  rcb-parm-ptr(5) to ADDRESS of RFC-TABLES-TABLE
           set  rcb-parm-ptr(6) to ADDRESS of RFC-EXCEPTION-PTR

           PERFORM RFC-CALL.

      *    IF rcb-retcode = 0 then
      *       ....... Display RFCSI-BUFFER
      *    ELSE
      *       ....... Error MSG
      *    END-IF
           .
           EXIT.

      ********************
       RFC-CALL SECTION.
      ********************
      *
      *  IMS/Batch: Dynamic Call
      *
      *    CALL 'SAPLCOB' USING BY REFERENCE RCB.
      *
      *  CICS:  Dynamic Program Link
      *
           EXEC CICS
                LINK PROGRAM ('SAPLCOC') COMMAREA(RCB)
           END-EXEC
           .
           EXIT.

       END PROGRAM SAPINFO.
```

7.4 PL/1 Example Program

```
*PROCESS SYSTEM(OS) DISPLAY(STD);
*PROCESS RENT STDSYS DFT(IEEE) LIMITS(FIXEDDEC(31));

 /*********************************************************************
 **
 **
 **  Read User details
 **
 **
 **  Function: BAPI_USER_GET_DETAIL
 **            EXPORTING
 **              USERNAME
 **            IMPORTING
 **              ADDRESS
 **            TABLES
 **              RETURN
 **
 **
 **  Date:     27102007
 **
 **  System:   ABC (4.6D)
 **
 **
 **  Copyright(C) SAP AG Germany 2007. All rights reserved
 **
 **
 *********************************************************************/

 PL1USRE: Proc(jclparm) options(Main,NoExecOps);

 DCL jclparm                          CHAR(100) VAR;


 /*
 *   Input Message
 */
 DCL 1 Inp,
       2 LL                           FIXED BIN(15),
       2 Msg                          CHAR(100);


 /*
 *   Output Message
 */
 DCL 1 Out,
       2 LL                           FIXED BIN(15),
       2 Msg                          CHAR(2048) VARZ;

 DCL str                              CHAR(200) VARZ  based;


 /*
 *   set Language Envrionment Runtime Options for this program
 */

 DCL PLIXOPT CHAR(80) VAR INIT('POSIX(ON)') STATIC EXTERNAL;




 /*********************************************************************/
 /*  SAPLCO includes                                                  */
 /*********************************************************************/
 /*
 *  SAPLCO RFC Control Block
 */
 Define Structure
     1 RFC_CONTROL_BLOCK,
```

```
      2 eyecatcher                  CHAR(16),
      2 function                    PTR,
      2 parmcnt                     FIXED BIN(31),
      2 parm                        PTR,
      2 retcode                     FIXED BIN(31),
      2 errtxt                      PTR,
      2 status                      FIXED BIN(31),
      2 program                     CHAR(8),
      2 attributes                  PTR,
      2 statistic                   PTR,
      2 reserved                    CHAR(240);



/*****************************
 *  SAP RFC Type Definitions
 *****************************/

 /*
 *  RFC Parameter Description
 */
 Define Structure
     1 RFC_PARAMETER,
       2 name                       PTR,
       2 nlen                       FIXED BIN(31),
       2 type                       FIXED BIN(31),
       2 leng                       FIXED BIN(31),
       2 addr                       PTR;


 /*
 *  RFC Table Description
 */
 Define Structure
     1 RFC_TABLE,
       2 name                       PTR,
       2 nlen                       FIXED BIN(31),
       2 type                       FIXED BIN(31),
       2 leng                       FIXED BIN(31),
       2 ithandle                   FIXED BIN(31),
       2 itmode                     FIXED BIN(31),
       2 newitab                    FIXED BIN(31);


 /*
 *  RFC Error Info (Extended)
 */
 Define Structure
     1 RFC_ERROR_INFO_EX,
       2 group                      CHAR(1),
       2 key                        CHAR(32) VARZ,
       2 message                    CHAR(512) VARZ;


 /*
 *  RFC Type Element2
 */
 Define Structure
     1 RFC_TYPE_ELEMENT2,
       2 name                       PTR,
       2 type                       FIXED BIN(31),
       2 length                     FIXED BIN(31),
       2 decimals                   FIXED BIN(31),
       2 offset                     FIXED BIN(31);







 /*
 *  SAPLCO RFC Function Names
 */
 DCL ItCreate                       CHAR(8)  VARZ
```

```
                                  INIT('ItCreate');
DCL ItCpyLine                     CHAR(9)  VARZ
                                  INIT('ItCpyLine');
DCL ItDelete                      CHAR(8)  VARZ
                                  INIT('ItDelete');
DCL ItFree                        CHAR(6)  VARZ
                                  INIT('ItFree');
DCL ItFill                        CHAR(6)  VARZ
                                  INIT('ItFill');
DCL RfcOpenEx                     CHAR(9)  VARZ
                                  INIT('RfcOpenEx');
DCL RfcClose                      CHAR(8)  VARZ
                                  INIT('RfcClose');
DCL RfcLastErrorEx                CHAR(14) VARZ
                                  INIT('RfcLastErrorEx');
DCL RfcInstallStructure2          CHAR(20) VARZ
                                  INIT('RfcInstallStructure2');
DCL RfcCallReceive                CHAR(14) VARZ
                                  INIT('RfcCallReceive');

/*
*  RFC Return Codes
*/
DCL RFC_OK                        FIXED BIN(31)  VALUE(0);
DCL RFC_FAILURE                   FIXED BIN(31)  VALUE(1);
DCL RFC_EXCEPTION                 FIXED BIN(31)  VALUE(2);
DCL RFC_SYS_EXCEPTION             FIXED BIN(31)  VALUE(3);
DCL RFC_CALL                      FIXED BIN(31)  VALUE(4);
DCL RFC_INTERNAL_COM              FIXED BIN(31)  VALUE(5);
DCL RFC_CLOSED                    FIXED BIN(31)  VALUE(6);
DCL RFC_RETRY                     FIXED BIN(31)  VALUE(7);
DCL RFC_NO_TID                    FIXED BIN(31)  VALUE(8);
DCL RFC_EXECUTED                  FIXED BIN(31)  VALUE(9);
DCL RFC_SYNCHRONIZE               FIXED BIN(31)  VALUE(10);
DCL RFC_MEMORY_INSUFFICIENT       FIXED BIN(31)  VALUE(11);
DCL RFC_VERSION_MISMATCH          FIXED BIN(31)  VALUE(12);
DCL RFC_NOT_FOUND                 FIXED BIN(31)  VALUE(13);
DCL RFC_CALL_NOT_SUPPORTED        FIXED BIN(31)  VALUE(14);
DCL RFC_NOT_OWNER                 FIXED BIN(31)  VALUE(15);
DCL RFC_NOT_INITIALIZED           FIXED BIN(31)  VALUE(16);
DCL RFC_SYSTEM_CALLED             FIXED BIN(31)  VALUE(17);
DCL RFC_INVALID_HANDLE            FIXED BIN(31)  VALUE(18);
DCL RFC_INVALID_PARAMETER         FIXED BIN(31)  VALUE(19);
DCL RFC_CANCELED                  FIXED BIN(31)  VALUE(20);


/*
*  RFC Data Types
*/
DCL RFCTYPE_CHAR                  FIXED BIN(31)  VALUE(0);
DCL RFCTYPE_DATE                  FIXED BIN(31)  VALUE(1);
DCL RFCTYPE_BCD                   FIXED BIN(31)  VALUE(2);
DCL RFCTYPE_TIME                  FIXED BIN(31)  VALUE(3);
DCL RFCTYPE_BYTE                  FIXED BIN(31)  VALUE(4);
DCL RFCTYPE_ITAB                  FIXED BIN(31)  VALUE(5);
DCL RFCTYPE_NUM                   FIXED BIN(31)  VALUE(6);
DCL RFCTYPE_FLOAT                 FIXED BIN(31)  VALUE(7);
DCL RFCTYPE_INT                   FIXED BIN(31)  VALUE(8);
DCL RFCTYPE_INT2                  FIXED BIN(31)  VALUE(9);
DCL RFCTYPE_INT1                  FIXED BIN(31)  VALUE(10);
DCL RFCTYPE_DAT1                  FIXED BIN(31)  VALUE(11);
DCL RFCTYPE_DAT2                  FIXED BIN(31)  VALUE(12);
DCL RFCTYPE_XMLDATA               FIXED BIN(31)  VALUE(28);
DCL RFCTYPE_STRING                FIXED BIN(31)  VALUE(29);
DCL RFCTYPE_XSTRING               FIXED BIN(31)  VALUE(30);



/*********************************************************************/
/*  SAP Structures                                                   */
/*********************************************************************/

/*********************************************************************
```

```
*
*   Structure: BAPIADDR3  (1857 BYTES)
*
*********************************************************************/
Define Structure
    1 BAPIADDR3,
      2 pers_no                     CHAR(10), /*Person number     */
      2 addr_no                     CHAR(10), /*Address number    */
      2 title_p                     CHAR(30), /*Title             */
      2 firstname                   CHAR(40), /*First name        */
      2 lastname                    CHAR(40), /*Last name         */
      2 birth_name                  CHAR(40), /*Name at birth     */
      2 middlename                  CHAR(40), /*2nd forename      */
      2 secondname                  CHAR(40), /*2nd family name   */
      2 fullname                    CHAR(80), /*Complete name     */
      2 fullname_x                  CHAR(1),  /*converted         */
      2 title_aca1                  CHAR(20), /*Acad. title       */
      2 title_aca2                  CHAR(20), /*Acad. title       */
      2 prefix1                     CHAR(20), /*Name prefix       */
      2 prefix2                     CHAR(20), /*Name prefix       */
      2 title_sppl                  CHAR(20), /*Name supplement   */
      2 nickname                    CHAR(40), /*Nickname          */
      2 initials                    CHAR(10), /*Initials          */
      2 nameformat                  CHAR(2),  /*Format name       */
      2 namcountry                  CHAR(3),  /*Format country    */
      2 langu_p                     CHAR(1),  /*Language          */
      2 langup_iso                  CHAR(2),  /*Lang. (ISO)       */
      2 sort1_p                     CHAR(20), /*Search term A     */
      2 sort2_p                     CHAR(20), /*Search term B     */
      2 department                  CHAR(40), /*Department        */
      2 function                    CHAR(40), /*Function          */
      2 building_p                  CHAR(10), /*Building code     */
      2 floor_p                     CHAR(10), /*Floor             */
      2 room_no_p                   CHAR(10), /*Room no.          */
      2 inits_sig                   CHAR(10), /*Short name        */
      2 inhouse_ml                  CHAR(10), /*Internal mail     */
      2 comm_type                   CHAR(3),  /*Communication typ */
      2 title                       CHAR(30), /*Title             */
      2 name                        CHAR(40), /*Name              */
      2 name_2                      CHAR(40), /*Name 2            */
      2 name_3                      CHAR(40), /*Name 3            */
      2 name_4                      CHAR(40), /*Name 4            */
      2 c_o_name                    CHAR(40), /*c/o               */
      2 city                        CHAR(40), /*City              */
      2 district                    CHAR(40), /*District          */
      2 city_no                     CHAR(12), /*City no.          */
      2 distrct_no                  CHAR(8),  /*District          */
      2 chckstatus                  CHAR(1),  /*Test status       */
      2 postl_cod1                  CHAR(10), /*Postal code       */
      2 postl_cod2                  CHAR(10), /*PO Box post cde   */
      2 postl_cod3                  CHAR(10), /*Company post cd   */
      2 po_box                      CHAR(10), /*PO Box            */
      2 po_box_cit                  CHAR(40), /*PO Box city       */
      2 pboxcit_no                  CHAR(12), /*City no.          */
      2 deliv_dis                   CHAR(15), /*Delivery dist.    */
      2 transpzone                  CHAR(10), /*Transport.zone    */
      2 street                      CHAR(60), /*Street            */
      2 street_no                   CHAR(12), /*Street no.        */
      2 str_abbr                    CHAR(2),  /*Street abbrev.    */
      2 house_no                    CHAR(10), /*House number      */
      2 house_no2                   CHAR(10), /*House no. suppl   */
      2 str_suppl1                  CHAR(40), /*Street 2          */
      2 str_suppl2                  CHAR(40), /*Street 3          */
      2 str_suppl3                  CHAR(40), /*Street 4          */
      2 location                    CHAR(40), /*Street 5          */
      2 building                    CHAR(10), /*Building code     */
      2 floor                       CHAR(10), /*Floor             */
      2 room_no                     CHAR(10), /*Room no.          */
      2 country                     CHAR(3),  /*Country           */
      2 countryiso                  CHAR(2),  /*ISO code          */
      2 langu                       CHAR(1),  /*Language          */
      2 langu_iso                   CHAR(2),  /*Lang. (ISO)       */
      2 region                      CHAR(3),  /*Region            */
      2 sort1                       CHAR(20), /*Search term A     */
```

```
      2 sort2                       CHAR(20), /*Search term B     */
      2 time_zone                   CHAR(6),  /*Time zone         */
      2 taxjurcode                  CHAR(15), /*Jurisdict. code   */
      2 adr_notes                   CHAR(50), /*Notes             */
      2 tel1_numbr                  CHAR(30), /*Telephone         */
      2 tel1_ext                    CHAR(10), /*Extension         */
      2 fax_number                  CHAR(30), /*Fax               */
      2 fax_extens                  CHAR(10), /*Extension         */
      2 e_mail                      CHAR(241), /*E-mail address   */
      2 build_long                  CHAR(20); /*Building code     */

/*
*  Structure length and number of type info entires
*/

DCL BAPIADDR3_L                     FIXED BIN(31)  VALUE(1857);
DCL BAPIADDR3_F                     FIXED BIN(31)  VALUE(   1);

/*
*  Type Info Table (RFC_TYPE_ELEMENT2)
*/
DCL BAPIADDR3_D( 5)                 FIXED BIN(31)
    init( 0,   0,1857,   0,   0);    /* Fields 01-78 type char     */


/*****************************************************************
*
*   Structure: BAPIBNAME  (12 BYTES)
*
*****************************************************************/
Define Structure
    1 BAPIBNAME,
      2 bapibname                   CHAR(12); /* User             */

/*
*  Structure length and number of type info entires
*/
DCL BAPIBNAME_L                     FIXED BIN(31)  VALUE(  12);
DCL BAPIBNAME_F                     FIXED BIN(31)  VALUE(   1);

/*
*  Type Info Table (RFC_TYPE_ELEMENT2)
*/
DCL BAPIBNAME_D( 5)                 FIXED BIN(31)
    init( 0,   0,  12,   0,   0);    /*C:bapibname               */


/*****************************************************************
*
*   Structure: BAPIRET2  (548 BYTES)
*
*****************************************************************/
Define Structure
    1 BAPIRET2,
      2 type                        CHAR(1),  /*Message type      */
      2 id                          CHAR(20), /*Message ID        */
      2 number                      CHAR(3),  /*Message number    */
      2 message                     CHAR(220), /*Message text     */
      2 log_no                      CHAR(20), /*Log number        */
      2 log_msg_no                  CHAR(6),  /*Message no.       */
      2 message_v1                  CHAR(50), /*Message variable  */
      2 message_v2                  CHAR(50), /*Message variable  */
      2 message_v3                  CHAR(50), /*Message variable  */
      2 message_v4                  CHAR(50), /*Message variable  */
      2 parameter                   CHAR(32), /*Parameter name    */
      2 *                           CHAR(2),
      2 row                         FIXED BIN(31), /*Parameter lin*/
      2 field                       CHAR(30), /*Field name        */
      2 system                      CHAR(10); /*Logical system    */

/*
*  Structure length and number of type info entries
*/
DCL BAPIRET2_L                      FIXED BIN(31)  VALUE( 548);
```

```
DCL BAPIRET2_F                      FIXED BIN(31)  VALUE(   3);

/*
*  Type Info Table (RFC_TYPE_ELEMENT2)
*/
DCL BAPIRET2_D(15)                  FIXED BIN(31)
    init( 0,   0, 502,   0,   0,    /*C:Fields 01 - 11 type char  */
          0,   8,   4,   0, 504,    /*I:row            type int   */
          0,   0,  40,   0, 508);   /*C:Fields 13 - 14 type char  */

/*********************************************************************/
/*  Function Name                                                    */
/*********************************************************************/

DCL BAPI_USER_GET_DETAIL_N          CHAR(20) VARZ
                                    INIT('BAPI_USER_GET_DETAIL');

/*********************************************************************/
/*  Structure Names                                                  */
/*********************************************************************/

DCL BAPIADDR3_N                     CHAR(9)  VARZ
                                    INIT('BAPIADDR3');
DCL BAPIBNAME_N                     CHAR(9)  VARZ
                                    INIT('BAPIBNAME');
DCL BAPIRET2_N                      CHAR(8)  VARZ
                                    INIT('BAPIRET2');

/*********************************************************************/
/*  Input-field Names                                                */
/*********************************************************************/

DCL USERNAME_N                      CHAR(8)  VARZ
                                    INIT('USERNAME');

/*********************************************************************/
/*  Output-field Names                                               */
/*********************************************************************/

DCL ADDRESS_N                       CHAR(7)  VARZ
                                    INIT('ADDRESS');

/*********************************************************************/
/*  Table Names                                                      */
/*********************************************************************/

DCL RETURN_N                        CHAR(6)  VARZ
                                    INIT('RETURN');

/*********************************************************************/
/*  Structure Type-handles                                           */
/*********************************************************************/

DCL BAPIADDR3_T                     FIXED BIN(31);
DCL BAPIBNAME_T                     FIXED BIN(31);
DCL BAPIRET2_T                      FIXED BIN(31);

/*********************************************************************/
/*  Table handles                                                    */
/*********************************************************************/

DCL RETURN_H                        FIXED BIN(31);


/*********************************************************************/
/*  Data Fields + Structures                                         */
/*********************************************************************/

DCL Address                         type BAPIADDR3;
DCL Username                        type BAPIBNAME;
DCL Return                          type BAPIRET2;

/*********************************************************************/
/*  local Variables                                                  */
```

```
 /*********************************************************************/
 DCL rcb                              type RFC_CONTROL_BLOCK;
 DCL errinfox                         type RFC_ERROR_INFO_EX;
 DCL exp(10)                          type RFC_PARAMETER;
 DCL imp(10)                          type RFC_PARAMETER;
 DCL tab(10)                          type RFC_TABLE;
 DCL i                                FIXED BIN(31) INIT(0);
 DCL x                                CHAR(1) INIT(' ');
 DCL rc                               FIXED BIN(31) INIT(0);
 DCL more_data                        FIXED BIN(31) INIT(0);
 DCL inp_len                          FIXED BIN(15) INIT(0);
 DCL tab_entries                      FIXED BIN(31) INIT(0);
 DCL rfc_handle                       FIXED BIN(31) INIT(0);
 DCL rfc_exception_ptr                PTR;
 DCL rfc_exception_str                CHAR(30) VARZ
                                      BASED(rfc_exception_ptr);
 DCL rfc_logon_data                   CHAR(255) VARZ;
 DCL rcb_parm(10)                     PTR;

 /*
 *  entry point for dynamic SAPLCO call
 */

 DCL SAPLCO EXT ENTRY(POINTER       BYVALUE)
              RETURNS(FIXED BIN(31) BYVALUE)
              OPTIONS(FETCHABLE);



/*********************************************************************/
 /*                                                                   */
 /* Main                                                              */
 /*                                                                   */
 /*********************************************************************/

    call Rfc_Init();
    call Rfc_Get_Arguments();
    call Rfc_Open_Session();
    if rfc_handle > 0 then do;
       call Rfc_Install_Structures();
       call Rfc_Create_Tables();
       call Rfc_Data_Input();
       call Rfc_Describe_Params();
       call Rfc_Call_Function();
       call Rfc_Out_Message();
       call Rfc_Delete_Tables();
       call Rfc_Close_Session();
    end;

    return;


 /*********************************************************************/
  Rfc_Init: procedure;
 /*********************************************************************/

      call plifill( Addr(rcb), '00'x, SIZE(rcb) );
      call plifill( Addr(Out), '00'x, SIZE(Out) );
      call plifill( Addr(Address), ' ', 1857 );
      call plifill( Addr(Username),' ',   12 );
      call plifill( Addr(Return),  ' ',  548 );

      rfc_logon_data = 'LOGON_MYSAP' || '00'x; /* DD:LCOPARM alias */

      rcb.parm = Addr(rcb_parm);

      return;

  end Rfc_Init;


 /*********************************************************************/
  Rfc_Get_Arguments: procedure;
 /*********************************************************************/
```

```
      /* get Args from JCL-EXEC-PARMS */
      Inp.Msg = jclparm;            /* copy param string (max 100) */
      Inp.LL  = Length(jclparm);    /* get length                  */
      inp_len = Inp.LL;

      return;
  end Rfc_Get_Arguments;


 /*****************************************************************/
  Rfc_Out_Message: procedure;
 /*****************************************************************/

      if Out.LL = 0 then
         Out.LL = Length( Out.Msg );

      put edit( Out.Msg )(A(Out.LL)) skip;

      return;

  end Rfc_Out_Message;


/*****************************************************************/
 Rfc_Open_Session: procedure;
 /*****************************************************************/

      rcb.function = Addr(RfcOpenEx);
      rcb.parmcnt  = 2;
      rcb_parm(1)  = Addr(rfc_logon_data);
      rcb_parm(2)  = Addr(errinfox);
      rc = SAPLCO_CALL();

      rfc_handle   = rcb.retcode;

      if rfc_handle <= 0 then do;         /* open error       */
         if rfc_handle = 0 then           /* RFC error        */
            Out.Msg = errinfox.message;
         else
            Out.Msg = rcb.errtxt->str;    /* LCO error        */

         call Rfc_Out_Message();
      end;

      return;

  end Rfc_Open_Session;


 /*****************************************************************/
  Rfc_Close_Session: procedure;
 /*****************************************************************/

      rcb.function = Addr(RfcClose);
      rcb.parmcnt  = 1;
      rcb_parm(1)  = PointerValue(rfc_handle);
      rc = SAPLCO_CALL();

      return;

   end Rfc_Close_Session;


 /*****************************************************************/
  Rfc_Install_Structures: procedure;
 /*****************************************************************/

 /*
 *    set structure info for RFC data conversion
 */
      rcb.function  = Addr(RfcInstallStructure2);
      rcb.parmcnt   = 4;
      rcb_parm(1)   = Addr(BAPIADDR3_N);
```

```
      rcb_parm(2)   = Addr(BAPIADDR3_D);
      rcb_parm(3)   = PointerValue(BAPIADDR3_F);
      rcb_parm(4)   = Addr(BAPIADDR3_T);
      rc = SAPLCO_CALL();

      rcb_parm(1)   = Addr(BAPIBNAME_N);
      rcb_parm(2)   = Addr(BAPIBNAME_D);
      rcb_parm(3)   = PointerValue(BAPIBNAME_F);
      rcb_parm(4)   = Addr(BAPIBNAME_T);
      rc = SAPLCO_CALL();

      rcb_parm(1)   = Addr(BAPIRET2_N);
      rcb_parm(2)   = Addr(BAPIRET2_D);
      rcb_parm(3)   = PointerValue(BAPIRET2_F);
      rcb_parm(4)   = Addr(BAPIRET2_T);
      rc = SAPLCO_CALL();

      return;

  end Rfc_Install_Structures;



/*****************************************************************/
  Rfc_Create_Tables: procedure;
/*****************************************************************/

      rcb.function  = Addr(ItCreate);
      rcb.parmcnt   = 4;
      rcb_parm(1)   = Addr(RETURN_N);
      rcb_parm(2)   = PointerValue(BAPIRET2_L);
      rcb_parm(3)   = PointerValue(0);
      rcb_parm(4)   = PointerValue(0);
      rc = SAPLCO_CALL();

      RETURN_H = rcb.retcode;

      return;

  end Rfc_Create_Tables;


/*****************************************************************/
  Rfc_Delete_Tables: procedure;
 /*****************************************************************/

      rcb.function  = Addr(ItDelete);
      rcb.parmcnt   = 1;
      rcb_parm(1)   = PointerValue(RETURN_H);
      rc = SAPLCO_CALL();

      return;

  end Rfc_Delete_Tables;



/*****************************************************************/
 Rfc_Describe_Params: procedure;
/*****************************************************************/
 /*
 *    Description tables of all Export,Import,Table Parameters
 *    used by BAPI_USER_GET_DETAIL
 *    For each Parameter:
 *        Name, NameLen, DataLen, DataType, DataAddr/TabHandle
 */

      i = 1;
 /*
 *    User name
 */
      exp(i).name = Addr(USERNAME_N);
      exp(i).nlen = Length(USERNAME_N);
      exp(i).leng = BAPIBNAME_L;
```

```
      exp(i).type = BAPIBNAME_T;
      exp(i).addr = Addr(Username);
      i = i + 1;

      exp(i).name = SYSNULL();

      i = 1;
 /*
 *    Address data
 */
      imp(i).name = Addr(ADDRESS_N);
      imp(i).nlen = Length(ADDRESS_N);
      imp(i).leng = BAPIADDR3_L;
      imp(i).type = BAPIADDR3_T;
      imp(i).addr = Addr(Address);
      i = i + 1;

      imp(i).name = SYSNULL();

      i = 1;
 /*
 *    Return structure
 */
      tab(i).name     = Addr(RETURN_N);
      tab(i).nlen     = Length(RETURN_N);
      tab(i).leng     = BAPIRET2_L;
      tab(i).type     = BAPIRET2_T;
      tab(i).ithandle = RETURN_H;
      tab(i).itmode   = 0;
      i = i + 1;

      tab(i).name     = SYSNULL();

      return;

  end Rfc_Describe_Params;



/*****************************************************************/
 Rfc_Call_Function: procedure;
 /*****************************************************************/
      rcb.function = Addr(RfcCallReceive);
      rcb.parmcnt  = 6;
      rcb_parm(1)  = PointerValue(rfc_handle);
      rcb_parm(2)  = Addr(BAPI_USER_GET_DETAIL_N);
      rcb_parm(3)  = Addr(exp);
      rcb_parm(4)  = Addr(imp);
      rcb_parm(5)  = Addr(tab);
      rcb_parm(6)  = Addr(rfc_exception_ptr);
      rc = SAPLCO_CALL();

      /* check return code, returned data */

      if rcb.retcode = RFC_OK | rcb.retcode = RFC_EXCEPTION then
         call Rfc_Data_Output();
      else do;

        /* get error information */

        rcb.function = Addr(RfcLastErrorEx);
        rcb.parmcnt  = 1;
        rcb_parm(1)  = Addr(errinfox);
        rc = SAPLCO_CALL();

        Out.Msg = errinfox.message;

      end;

      return;
  end Rfc_Call_Function;
```

```
 /*****************************************************************/
  SAPLCO_CALL: procedure returns (FIXED BIN(31));
 /*****************************************************************/

      DCL rc1    FIXED BIN(31) init(0);
      DCL fetched FIXED BIN(31) static init(0);

      if fetched = 0 then do;
         fetch SAPLCO title('SAPLCOB');
         fetched = 1;
      end;
      rc1 = SAPLCO( Addr(rcb) );

      return( rc1 );

  end SAPLCO_CALL;



/*****************************************************************/
  Rfc_Data_Input: procedure;
/*****************************************************************/

      /*
      *  move data to input fields
      */

      rfc_exception_ptr = SYSNULL();
      if inp_len > 0 then do;            /* set Name from Inp.Msg */
         Username.bapibname = SubStr(Inp.Msg,1,12);
      end;
      else do;
         Username.bapibname = '            ';
      end;

      return;

  end Rfc_Data_Input;



/*****************************************************************/
  Rfc_Data_Output: procedure;
/*****************************************************************/

      /* check if table Return is filled ? */

      rcb.function  = Addr(ItFill);
      rcb.parmcnt   = 1;
      rcb_parm(1)   = PointerValue(RETURN_H);
      rc = SAPLCO_CALL();

      tab_entries   = rcb.retcode;               /* no of entries */

      if tab_entries > 0 then do;
         rcb.function  = Addr(ItCpyLine);        /* read line     */
         rcb.parmcnt   = 3;
         rcb_parm(1)   = PointerValue(RETURN_H); /* table handle  */
         rcb_parm(2)   = PointerValue(1);        /* get 1.line    */
         rcb_parm(3)   = Addr(Return);
         rc = SAPLCO_CALL();
      end;

      /* move SAP message to Output Display Buffer */

      if Return.number > '000' then do;
         Out.Msg = Return.type   ||
                   Return.number || ':' ||
                   Return.message;
         if Return.type = 'E' | Return.type = 'A' then do;
            return;
         end;
      end;
```

```
      /* process any other output fields here */

      call plimove( Addr(Out.Msg), Addr(Address), BAPIADDR3_L);

      return;

  end Rfc_Data_Output;


  end PL1USER;
```

7.5 Alternatives to SAP LCO

Alternatively, you can call SAP BAPIs also across Java interfaces (Java Connector), as a web service. Another method would be to use middleware – such as SAP PI. We are only mentioning these techniques for the sake of completeness – they are not described in this documentation.

Related Content

Components of SAP Communication Technology (in SAP Help Portal)

RFC (in SAP Help Portal)

RFCSDK Guide 6.20

Master the Five Remote Function Calls (RFC) Types in ABAP, Part 1: A Comprehensive Guide for SAP Programmers and Administrators

Acknowledgement

Thanks to Jennifer Johnson. This documentation would not be possible without their contribution.

Copyright