Linking Applications to Adabas

Since most systems do not allow a standard call to Adabas, Software AG provides an application programming interface (API) to translate calls issued by an application program into a form that can be handled by Adabas.

Batch applications are supported in both single-user and multiuser mode; online operations are controlled by teleprocessing (TP) monitors. The Adabas API is available across all supported mainframe platforms; versions of the API that are specific to particular TP monitors are provided.

"Adalink" is a generic term that refers to the portion of the API that is specific to a particular TP monitor.

This chapter covers the following topics:

- How the Adabas API Works
- Available Link Routines
- Required Work Area
- Required Application Reentrancy Properties
- Adabas Control Block Options
- Programming Conventions for Issuing Direct Calls
- Using the Adabas API in Batch Mode
- Support for OpenEdition OS/390 Adabas Clients

How the Adabas API Works

- Online Operation
- Batch Operation

Online Operation

As an online operation, a request to Adabas is processed as follows:

1. The TP monitor invokes the application program. The application program must be loaded into the TP monitor region.
2. The application program invokes the Adabas API. *The Adabas API module must be installed in the TP monitor as an application module.*
3. The Adabas API takes the Adabas command passed to it from the application program and
    - builds the required control blocks and structures;

- translates the Adabas parameter list provided by the application program call into a request that can be handled by the Adabas router or SVC;
- includes information that identifies the user (terminal ID, TJID etc.) to Adabas.

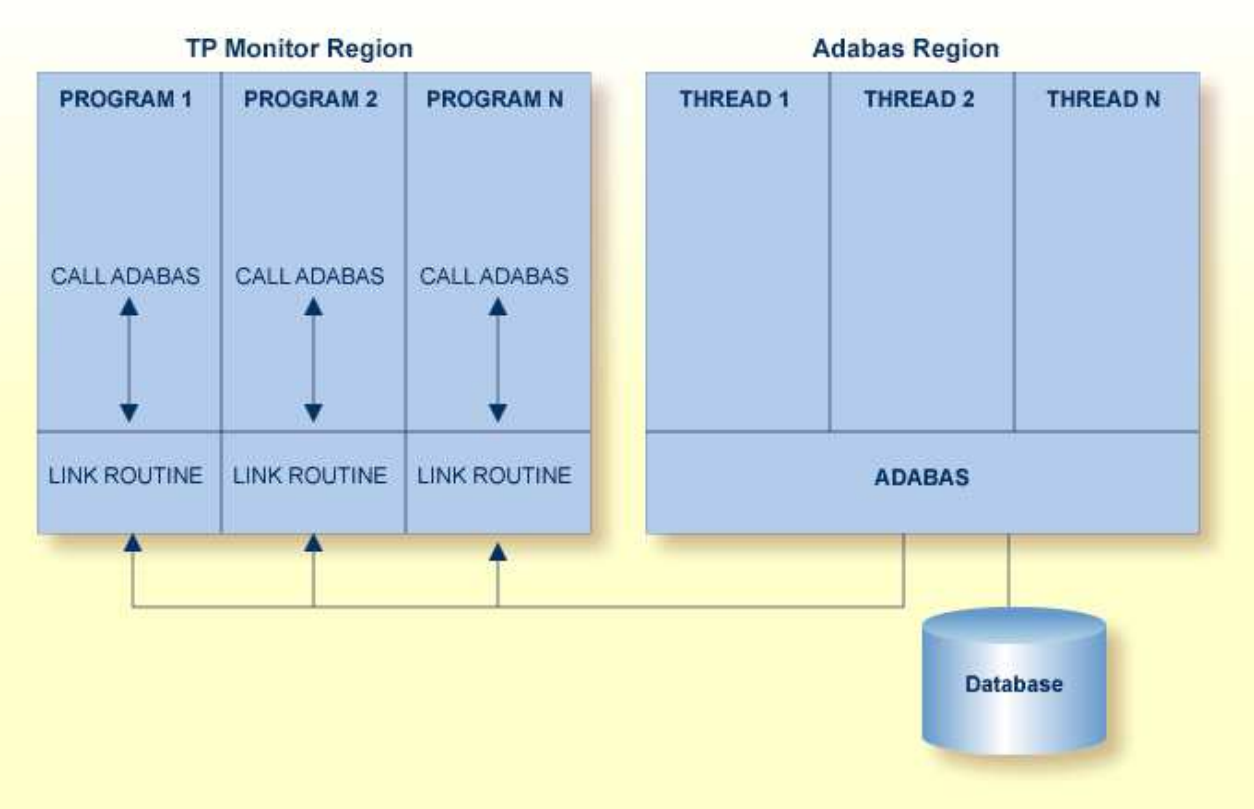
The TP monitor's equivalent of the LINK function is used to pass the user's Adabas control block and buffers to the API.

4. The Adabas API usually uses the Adabas router or SVC (supervisor call) installed on the operating system to send the formatted request to Adabas.
5. The Adabas router or SVC moves the user's control block and buffers from the TP monitor region to the Adabas region (into the Adabas nucleus).
6. The Adabas API waits for a response from the Adabas nucleus.

   The TP monitor's equivalent of a WAIT is posted when the Adabas function is complete.
7. Adabas performs the function requested, then invokes the Adabas router or SVC, which returns the resulting data and response codes to the user application buffer.
8. The Adabas router or SVC then posts the Adabas API in the TP monitor region.
9. The Adabas API sends the response from the Adabas nucleus to the application program and returns control to the application program.
10. The application program returns control to the TP monitor.

The specific way each of the above functions is handled depends on the TP monitor used. In addition to these functions, each link routine can call one or more user exits at various processing points to provide additional capability and flexibility when making Adabas requests.

The following figure illustrates the basic configuration used by the majority of systems:



Adabas/TP Monitor Communication

Batch Operation

As a batch operation, a request to Adabas is processed as follows:

1. The operating system loads the batch application.
2. The batch application makes an Adabas request (CALL ADABAS ...).
3. The Adabas "stub" program ADAUSER loads and invokes ADARUN, which then loads and invokes the batch Adabas link routine ADALNK.
4. The ADALNK routine transforms the Adabas request into a format suitable for use by the Adabas nucleus.
5. The link routine invokes Adabas, usually through a call to the Adabas router or SVC installed in the operating system. It also determines a unique identification for the user.
6. The link routine then waits for Adabas to handle the request.
7. Adabas, which is usually running in a different address space or partition, processes the request and posts the link routine, returning all necessary buffers and response codes.
8. The link routine receives control and returns the Adabas buffers with response codes and data to the application.

Available Link Routines

The Adabas API is available for both batch and online applications. For online applications, the Adabas API is under the control of the TP monitor. When Adabas is installed, the Adabas API that is specific to the TP monitor in use is also installed.

- For IBM Operating Environments
- For BS2000 Operating Environments

For IBM Operating Environments

The following table lists the API versions and the corresponding supported TP monitors available for IBM operating environments:

Link Routine	TP Monitor
ADALCO	Com-plete
ADALNK	Batch (also TSO under OS/390 or z/OS)
ADALNKR	Batch/TSO under OS/390 or z/OS (reentrant)
ADALNA5	Batch for Entire System Server (formerly Natural Process)
ADALNI	IMS/DC
ADALNM	Intercomm
ADALNS	Shadow
ADALTM	Taskmaster
LNKOLSC	CICS (command-level)
LNKVSER	Batch under VSE/ESA (reentrant)

For BS2000 Operating Environments

The BS2000 version 7 ADALNK module contains the combined functionality of all the ADALNx modules of previous Adabas versions and contains the following entry points:

Entry Point	TP Monitor
ADALNK	Batch / TIAM
ADALNR	Batch / TIAM (reentrant)
ADALNN	UTM running Natural
ADALNU	UTM with Assembler or a 3GL language
ADAUTM	UTM with SNI BS2000 system Adabas database operations. For more information, read *ADAUTM (Universal Transaction Monitor Support)* .

Required Work Area

Parameters must be passed to the API. Many of the monitors do not allow standard parameter passing conventions, and the Adabas control block and buffer addresses must be moved into a special work area.

The work area is obtained by the application program from the TP monitor system. It must be specific to the user and addressable by the Adabas API. The application program must initially obtain and clear this area to binary zeros.

Note:
If your application program is written in Natural, the necessary API work area is handled without change to the program code.

The Adabas API versions each have a specific area reserved for use as a work area. For Com-plete, batch/TSO, and IMS/DC, this area is defined by DSECTs provided in the Adabas source library containing the API as follows:

Link Routine	Work Area DSECT	TP Monitor
ADALCO	MODIFLCO	Com-plete
ADALNK / ADALNKR	MODIFIED	Batch/TSO
ADALNI	MODIFLNI	IMS/DC

For these TP monitors, the work area must be at least as long as the DSECT area. The actual required length of this area depends on the TP monitor, and can be determined by examining the assembly listing of the link routine.

For TP monitors with DSECT work areas, a *reentrant* application program must pass to the link routine a pointer to this area as the 7th parameter on the call statement. See *Required Code Reentrancy Properties* for more information.

Once the MODIFxxx area has been initialized, it should not be changed by the application program between Adabas requests.

Work Area for the Batch/TSO Reentrant ADALNKR

Seventh Parameter Required

In addition to the basic six Adabas parameters, the reentrant batch link routine ADALNKR requires a seventh parameter that points to a work area for use by the program on every call. Some Software AG products like XCOM pass eight parameters to ADALNKR.

MODIFIED Work Area

The work area used is mapped by the MODIFIED DSECT, which is provided in the Adabas source library or as an A.book in the VSE sublibrary. The MODIFIED area is at least 192 bytes (decimal) in length. Since the area may be expanded in future Adabas releases, Software AG recommends that you reserve 256 bytes for the area.

Calling Requirements

The work area should be initialized to binary zeros before the first call to ADALNKR, and its contents should not be modified by calling application programs thereafter. Several key fields are stored in this area. If these fields are modified improperly, results are unpredictable and may range from poor performance to abnormal termination of the link routine.

When calling ADALNKR, it is critical to mark the last parameter address in the calling parameter list with an X’80’:

- High-level languages do this automatically when their CALL statements are employed.
- Assembler applications can do this by using the CALL macro to invoke ADALNKR.

Required Application Reentrancy Properties

Applications running under most TP monitors must use nonstandard calls to perform functions that are transparently handled by the operating system in a batch environment. In these cases, it is the reentrant properties of application code that determine how multiple users execute Adabas API calls online.

Each Adabas API version complies with the reentrancy requirements for its associated TP monitor. Application programs that use the Adabas API must also comply with the requirements for the TP monitor used.

Note:
The reentrancy requirement set by the TP monitor is a minimum. For example, if the TP monitor requires a quasi-reentrant application program, a fully reentrant program will also be accepted (see CICS special requirement below). However, if a reentrant application program is required, a quasi- or non-reentrant program is not acceptable.

Ideally, code for application programs that are shared by a large number of users (commonly used TP transactions) is "reentrant". The code itself never changes. All work areas are either in general registers or in user-specific work areas that are addressed by general registers. A transfer of control from one user to another requires only a change in the program counter (PSW) and the general registers. Many system routines are coded in this manner.

Most COBOL compilers do not produce reentrant code. The PL/ I compiler produces reentrant code, but by using operating system functions that are not allowed by most TP monitors. These limitations have led to the concept of quasi-reentrancy.

A "quasi-reentrant" program may alter its code between calls to TP monitor functions. When a monitor function is invoked, all user data must be saved in a special work area obtained from the TP monitor system. The TP monitor will then schedule another user task as the active task in the system, and this task may reuse the same code. When the original user’s task becomes active again, his work area is reestablished and control is passed back to the point at which the user requested a TP monitor function.

The following subsections give more detailed information about the reentrancy requirements of several TP monitors.

- Com-plete : Code Reentrancy Requirements

- CICS : Code Reentrancy Requirements
- TSO and IMS/DC (Standalone) : Code Reentrancy Requirements

Com-plete : Code Reentrancy Requirements

Com-plete does not require nonstandard calling sequences: users may use standard non-reentrant code. Adabas linkage is provided by a Com-plete service routine, which is automatically included in the user's load module if Adabas calls are contained in the user program. The service routine simply passes the user parameters to Com-plete and returns control when the Adabas command has been executed.

CICS : Code Reentrancy Requirements

LNKOLSC, the command-level CICS API, can use either quasi-reentrant or fully reentrant application programs, depending on how the API is installed. Optional installation procedures must be executed in order to use fully reentrant application programs under CICS; however, this allows the use of CICS program isolation under CICS/ESA 4.1 and above.

TSO and IMS/DC (Standalone) : Code Reentrancy Requirements

Only TSO and IMS/DC allow the use of "nonreentrant" application programs. However, this is the least efficient means of coding an application program to use with the Adabas API. If a nonreentrant application program is used, it will have to obtain the Adabas communication ID (LTERM, TJID or sign-on ID) on every Adabas request. For TSO and IMS/DC, nonreentrant is the minimum requirement; using a fully reentrant application program will result in significant performance gains (for IMS/ESA, performance gains will result for version 3.1 and above).

Note:
Reentrant and quasi-reentrant application programs must obtain a special work area for the Adabas API. See *Required Adabas API Work Area*.

Adabas Control Block Options

The first parameter passed to the Adabas API by the application program is a pointer to the Adabas control block (ADACB). The ADACB contains information needed to process an Adabas request.

The first byte of the ADACB is used by the Adabas API to determine the processing to be performed. The values for logical requests are:

Hex	Indicates ...
X'00'	a 1-byte file number (file numbers between 1 and 255)
X'30'	a 2-byte file number (file numbers between 1 and 65535)
X'40'	values greater than or equal to a blank. These are accepted as "logical application calls" to maintain compatibility with earlier releases of Adabas. The following calls, however, are reserved for use in special Software AG functions or products and are therefore not accepted: X'44', X'48', and X'4C'.

All other values in the first byte of the ADACB are reserved for use by Software AG.

Using One-Byte File Numbers

For an application program issuing Adabas commands for file numbers between 1 and 255 (single byte), build the control block as follows:

Position	Action
1	Place X’00’ in the first byte of the ADACB.
9	Place the file number in the second (rightmost) byte of the ACBFNR field of the ACB. The first (leftmost) byte of the ACBFNR field is used to store the logical (database) ID or number.

If the first byte in ACBFNR is zero, the API will use either the database ID value provided in the DDCARD input data (ADARUN cards) or the default database ID value assembled into the link routine at offset X’80’. Applications written in Software AG’s Natural language need not include the first byte of the ADACB because Natural supplies appropriate values.

Using Two-Byte File Numbers

Adabas permits the use of file numbers greater than 255 on logical requests. For an application program issuing Adabas commands for file numbers between 256 and 5000 (two-byte), build the control block as follows:

Position	Action
1	Place X’30’ in the first byte of the ADACB.
9	Use both bytes in ACBFNR for the file number, and use the two bytes in ACBRESP for the database (logical) ID.

If the ACBRESP field is zero, the API will use either the database ID from the ADARUN cards provided in DDCARD input data, or the default database ID value assembled into the link routine at offset X’80’.

Using Both One- and Two-Byte File Numbers in a Single Application

Because the application can reset the value in the first byte of the ADACB on each call, it is possible to mix both one- and two-byte file number requests in a single application.

If this method is used, you must ensure the proper construction of the ACBFNR and ACBRESP fields in the ADACB for each call type.

Software AG recommends that an application written to use two-byte file numbers always place X’30’ in the first byte of the ADACB, the logical ID in the ACBRESP field, and the file number in the ACBFNR field. The application can then treat both the database ID and file number as 2-byte binary integers, regardless of the value for the file number in use.

Using COBOL to Set the Control Byte

A programming language such as COBOL is not designed to easily manipulate single-byte values as required to establish two-byte file number support for the Adabas API. The following COBOL example illustrates one way to set these values:

```
WORKING-STORAGE SECTION
01  ACB-CONTROL
    05  ACB-TYPE           PIC 9(4) COMP.
    05  ACB-DATA REDEFINES ACB-TYPE.
          07  FILLER       PIC X.
          07  ACB-TYPE-X   PIC X.
01  ADABAS-CB.
    05  ACBTYPE            PIC X.
 .
PROCEDURE DIVISION
 .
 *  FOR SINGLE-BYTE FILE NUMBERS . . .
        MOVE 0 TO ACB-TYPE.
 .
 *  FOR TWO-BYTE FILE NUMBERS . . .
        MOVE 48 TO ACB-TYPE.
 .
    MOVE ACB-TYPE-X TO ACBTYPE.
 .
    CALL 'ADABAS' USING ADABAS-CB, . . .
 .
 .
```

The key to this code segment is the use of the REDEFINES clause to remap the PIC 9(4) COMP field to its constituent two bytes. Then the second byte containing the hexadecimal value for the Adabas control byte can be moved as "character" data to the Adabas control block.

Programming Conventions for Issuing Direct Calls

This section describes the procedures used to issue Adabas calls in direct mode from a program that is to be run under the control of one of the following teleprocessing (TP) monitors:

- Com-plete
- CICS
- IMS/DC
- Shadow II

Com-plete

Application programs that are to be run under control of Com-plete may be coded in exactly the same manner as batch programs. Since each application program is assigned a processing thread by Com-plete, the program need not be written using reentrant or quasi-reentrant code.

The following example shows an Adabas call from a COBOL program that is to be run under Com-plete:

```
WORKING-STORAGE SECTION
 .
 .
01  CONTROL-BLOCK COPY ADACBCOB.
01  FORMAT-BUFFER COPY FORDEF.
01  RECORD-BUFFER COPY RECDEF.
01  SEARCH-BUFFER COPY SEADEF.
01  VALUE-BUFFER  COPY VALDEF.
01  ISN-BUFFER    COPY ISNBUF.

PROCEDURE DIVISION
 .
 .
 .
CALL 'ADABAS' USING
              CONTROL-BLOCK, FORMAT-BUFFER, RECORD-BUFFER,
              SEARCH-BUFFER, VALUE-BUFFER, ISN-BUFFER.
 .
```

CICS

Applications running under CICS use the command-level API LNKOLSC and the CICS Transaction Work Area (TWA) to communicate parameters.

The high-level language interface guarantees the quasi-reentrancy of COBOL, PL/ I , and Assembler (release 1.4 and above).

Language	Control blocks and buffers may be defined ...
COBOL	in working storage. All of working storage is copied to a user work area when a transaction is initiated.
PL/ I	as automatic storage (default storage class) variables.

The addresses of the Adabas control block and buffers are passed in the same way for all releases of CICS. These addresses must be placed in the first six words of the TWA.

To place the parameter addresses in the TWA, Software AG provides an Assembler subroutine that can be called from a COBOL or Assembler application program. The subroutine uses entry point ADASTWA and accepts the TWA as its first parameter.

The 'EXEC CICS ADDRESS TWA' function is used to make the TWA addressable. The second to seventh parameters are the usual Adabas calling parameters. The Assembler subroutine places the parameter addresses into the TWA, and the CICS/Adabas link routine retrieves them from the TWA.

- Addressing the CICS TWA : Assembler
- Addressing the CICS TWA : PL/I
- Adabas Call Using CICS : VS COBOL
- Adabas Call Using CICS : COBOL II or COBOL/LE

Addressing the CICS TWA : Assembler

A CICS Assembler programmer can address the TWA directly by using an installation macro to place the addresses in the TWA and call Adabas.

Addressing the CICS TWA : PL/I

PL/ I offers a facility for addressing the TWA and obtaining the addresses of data areas. The programmer himself can place parameter addresses in the TWA. Your site may wish to establish a PL/ I preprocessor procedure to generate the calling code.

```
DCL 1        TWA  BASED  (TWAPTR),
    2    CBPTR    POINTER,
    2    FBPTR    POINTER,
    2    RBPTR    POINTER,
     .
    EXEC  CICS  ADDRESS  TWA  (TWAPTR)  END-EXEC;
     .
     .
    CBPTR=ADDR(ADA-CONTROL-BLOCK);
    FBPTR=ADDR(FORMAT-BUFFER);
    RBPTR=ADDR(RECORD-BUFFER);
```

Adabas Call Using CICS : VS COBOL

Under VS COBOL, Adabas is called using the statement:

```
EXEC CICS LINK PROGRAM ('ADABAS')
  END-EXEC.
```

```
CBL XOPTS (APOST)
IDENTIFICATION DIVISION.
 .
 .
WORKING-STORAGE SECTION.
 .
 .
01  ADABAS-CB   COPY  ADACBCOB.
01  ADABAS-FB   COPY  ADAFBCOB.
01  ADABAS-RB   COPY  ADARBCOB.
01  ADABAS-SB   COPY  ADASBCOB.
01  ADABAS-VB   COPY  ADAVBCOB.
01  ADABAS-IB   COPY  ADAIBCOB.
 .
 .

LINKAGE SECTION.
 .
 .
01  PARMLIST.
    05  FILLER    PIC S9(08) COMP.
    05  TWAPTR    PIC S9(08) COMP.

01  TWA.
    05  PARM-ADDRESSES OCCURS 7 TIMES   PIC S9(08) COMP.
 .
 .
PROCEDURE DIVISION.
 .
 .
    SERVICE RELOAD PARMLIST.
 .
    EXEC CICS ADDRESS TWA (TWAPTR) END-EXEC
```

```
    SERVICE RELOAD TWA.
 .
    CALL 'ADASTWA' USING TWA, ADABAS-CB, ADABAS-FB,
                         ADABAS-RB, ADABAS-SB, ADABAS-VB
                         ADABAS-IB.
    EXEC CICS LINK PROGRAM ('ADABAS') END-EXEC.
 .
 .
 .
```

Adabas Call Using CICS : COBOL II or COBOL/LE

Under COBOL II or COBOL/LE, Adabas is called using the statement:

```
EXEC CICS LINK PROGRAM ('ADABAS')  END-EXEC.
```

```
CBL XOPTS (APOST,ANSI85)
IDENTIFICATION DIVISION.
 .
 .
WORKING-STORAGE SECTION.
 .
 .
01  ADABAS-CB   COPY  ADACBCOB.
01  ADABAS-FB   COPY  ADAFBCOB.
01  ADABAS-RB   COPY  ADARBCOB.
01  ADABAS-SB   COPY  ADASBCOB.
01  ADABAS-VB   COPY  ADAVBCOB.
01  ADABAS-IB   COPY  ADAIBCOB.
 .
 .
LINKAGE SECTION.
 .
01  TWA.
    05  PARM-ADDRESSES OCCURS 7 TIMES   PIC S9(08) COMP.
 .
 .
PROCEDURE DIVISION.
 .
 .
 .  EXEC CICS ADDRESS TWA (ADDRESS OF TWA) END-EXEC.
 .
    CALL 'ADASTWA' USING TWA, ADABAS-CB, ADABAS-FB,
                         ADABAS-RB, ADABAS-SB, ADABAS-VB,
                         ADABAS-IB.
    EXEC CICS LINK PROGRAM ('ADABAS') END-EXEC.
 .
 .
 .
```

IMS/DC

IMS message processing programs that use the Adabas API require no special link and need not be reentrant. However, a reentrant option is supported: the application code and the Adabas IMS API module ADALNI can function with full reentrancy if the application program provides a work area as the 7th parameter when calling the API (see example below).

Under IMS/ESA 3.1 and above, the API ADALNI should be linked with an AMODE of 31 because the IMS control blocks referenced by the routine may be above the 16-megabyte line.

Adabas Call Using IMS/DC (Nonreentrant)

A nonreentrant Adabas API call under IMS/DC is made like a conventional Adabas API call under batch as follows:

```
WORKING-STORAGE-SECTION.
 .
 .
01  ADA-CONTROL BLOCK   COPY  ADACBCOB.
01  FORMAT-BUFFER       COPY  FORDEF.
01  RECORD-BUFFER       COPY  RECDEF.
01  SEARCH-BUFFER       COPY  SEADEF.
01  VALUE-BUFFER        COPY  VALDEF.
01  ISN-BUFFER          COPY  ISNDEF.
 .
PROCEDURE DIVISION.
 .
 .
    CALL 'ADABAS' USING ADA-CONTROL-BLOCK, FORMAT-BUFFER,
                        RECORD-BUFFER, SEARCH-BUFFER,
                        VALUE-BUFFER, ISN-BUFFER.
 .
 .
```

Adabas Call Using IMS/DC (Reentrant)

The Adabas ADALNI module can be assembled to be functionally reentrant. Refer to the Adabas Installation documentation for information about setting up ADALNI as a reentrant module.

To facilitate reentrant operation, an additional parameter pointing to a work area obtained by the caller must be passed to the ADALNI routine. This area must be initialized to binary zeros before the first call to the Adabas IMS API, and must not be modified between calls. The length of the ADALNI reentrant work area can be determined by examining the current assembly listing. Under Adabas, the length is 128 bytes.

The reentrant Adabas API call under IMS/DC is as follows:

```
WORKING-STORAGE-SECTION.
 .
 .
01  ADA-CONTROL BLOCK   COPY  ADACBCOB.
01  FORMAT-BUFFER       COPY  FORDEF.
01  RECORD-BUFFER       COPY  RECDEF.
01  SEARCH-BUFFER       COPY  SEADEF.
01  VALUE-BUFFER        COPY  VALDEF.
01  ISN-BUFFER          COPY  ISNDEF.
01  ADALNI-WORK-AREA       PIC X(128).
 .
PROCEDURE DIVISION.
    MOVE LOW-VALUES TO ADALNI-WORK-AREA.
 .
 .
    CALL 'ADABAS' USING ADA-CONTROL-BLOCK, FORMAT-BUFFER,
                        RECORD-BUFFER, SEARCH-BUFFER,
```

```
                        VALUE-BUFFER, ISN-BUFFER.
                        ADALNI-WORK-AREA.
 .
 .
```

Shadow II

The following procedure is used for calling Adabas from quasi-reentrant COBOL application programs that are to be run under Shadow II:

1. Define storage for variable Adabas control block and buffers in the linkage section of the COBOL program. Obtain storage using the Shadow get-variable function, ISHDHLGV.
2. Pass parameters to the Adabas interface routine directly using the Shadow calling procedure. Include in the call the count of the number of parameters as a binary halfword number.

   The Adabas interface routine is linked as part of the Shadow monitor and is defined in the Shadow program table.
3. Use the Shadow routine ISHDHLCP to access the Adabas interface routine. The first parameter is the character name of the interface routine (ADABAS); the second, the number of parameters being passed to the routine; followed by the parameters to be passed to Adabas.

Adabas Call Using Shadow II

```
WORKING-STORAGE SECTION.
 .
 .
01  ADABAS     PIC X(8) VALUE 'ADABAS'.
 .
 .
01  PARM-NUMBER        PIC S9(4) COMP VALUE +6.
01  ISHD-H80           PIC S9(4) COMP VALUE +80.
 .
 .
LINKAGE-SECTION.
 .
 .
01  ADABAS-CONTROL-BLOCK    COPY ADACBCOB.
 .
 .
PROCEDURE DIVISION USING TCBD RELOCATE.
 .
 .
GET-STORAGE.
CALL    'ISHDHLGV' USING ISHD-FULLWORD0, ISHD-XF0,
          ISHD-H80, BLL-NUMBER.
 .
 .
 .
CALL-ADABAS.
          CALL 'ISHDHLCP' USING     ADABAS, PARM-NUMBER,
                                    ADABAS-CONTROL-BLOCK,
                                    FORMAT-BUFFER, RECORD-BUFFER,
                                    SEARCH-BUFFER, VALUE-BUFFER,
                                    ISN-BUFFER.
 .
 .
```

Using the Adabas API in Batch Mode

The Adabas API in batch mode uses a standard call with a parameter list in register 1 and register 13 pointing to a register save area. This convention is supported by all major programming languages through their CALL mechanisms.

Under most mainframe operating systems, the batch API (ADALNK) can either be linked directly with the batch application module or it can be loaded by ADAUSER. Software AG *strongly* recommends that batch applications be linked with ADAUSER and not the batch API (ADALNK).

- ADAUSER and ADARUN with the Adabas API
- Batch Execution Modes

ADAUSER and ADARUN with the Adabas API

The ADAUSER module can optionally be linked with the Adabas API. ADAUSER provides upward compatibility with Adabas releases and a degree of isolation from changes that might be made in the API or the Adabas SVC in the future.

Each user program to be executed should be linked with the Adabas version-independent module ADAUSER, which dynamically loads the Adabas control module ADARUN. For batch mode execution, the user program should be linked with ADAUSER to achieve maximum environment independence, as shown below:

User Program linked with ...	**Advantage**
ADAUSER, ADARUN, and ADALNK	independent of mode and Adabas version
ADARUN and ADALNK	independent of mode only
ADALNK	none; no version- or mode-independence

The following sections illustrate the JCL/JCS required to link the batch application module with ADAUSER.

- Link Example (BS2000)
- Link Example (OS/390 or z/OS)
- Link Example (VM/ESA or z/VM)
- Link Example (VSE/ESA)

Link Example (BS2000)

```
/ EXEC $TSOSLNK
PROGRAM USERPROG
INCLUDE USERPGM,             ... User Library
INCLUDE ADAUSER,             ... Adabas Library
END
```

Link Example (OS/390 or z/OS)

```
// EXEC LKED,PARM='NCAL'
//LKED.SYSLMOD   DD           ... User Library
//LKED.ADALIB    DD           ... Adabas Library
//LKED.SYSIN     DD *
    INCLUDE SYSLMOD(USERPGM)
    INCLUDE ADALIB(ADAUSER)
    ENTRY USEREP                (see note)
    NAME USERPROG(R)
/*
```

Note:
The entry point, if specified, must be the entry point of the user program.

Link Example (VM/ESA or z/VM)

```
FILEDEF ADALIB DISK ADAVnnn LOADLIB fm
FILEDEF SYSLIN DISK LINKEDIT CARDS A
LKED userprog ( NCAL LET XREF MAP LIBE USERLIB LIST
```

The file "LINKEDIT CARDS A" contains the following linkage editor control statements. The user program exists as a TEXT file.

```
INCLUDE ADALIB(ADABAS)
ENTRY userprog
NAME userprog(R)
```

Note:
Link with ADAUSER is not applicable when using CMS since this system dynamically loads all necessary modules.

Link Example (VSE/ESA)

```
*  Appropriate assignments must be made for private
libraries, where necessary.
*
// OPTION CATAL
   PHASE USERPROG,*
   INCLUDE USERPGM
   INCLUDE ADAUSER
   ENTRY USEREP              (see note)
// EXEC LNKEDT
```

Note:
The entry point, if specified, must be the entry point of the user program.

Batch Execution Modes

When executing under batch, the program can be run in either single-user or multiuser mode:

- Single-user mode runs the application program, the batch API, ADARUN, and the Adabas nucleus in the same address space or partition.
- Multiuser mode executes the application program and the Adabas API in an address space separate from the Adabas nucleus.

The recommended mode of operation is multiuser mode. The user must provide only those job control statements required by ADARUN and the user program.

- Multiuser Mode Example (BS2000)
- Multiuser Mode Example (OS/390 or z/OS)
- Multiuser Mode Example (VM/ESA or z/VM)
- Multiuser Mode Example (VSE/ESA)
- Execution in Single-User Mode

Multiuser Mode Example (BS2000)

In SDF Format:

```
/ASS-SYSDTA *SYSCMD      (ADARUN PARAMETERS)
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK BLSLIB00,user modlib
/START-PROGRAM USERPROG,PR-MO=ANY,RUN-MODE=ADV(ALT-LIB=YES)
ADARUN MODE=MULTI...
```

In ISP Format:

```
/FILE ADABAS MODLIB,LINK=DDLIB
/SYSFILE TASKLIB=user modlib
/SYSFILE SYSDTA=(SYSCMD) (ADARUN PARAMETERS)
/EXEC USERPROG
ADARUN MODE=MULTI...
```

Notes:

1. As an alternative to using SYSDTA as the input stream, the user program can assign a sequential file containing the ADARUN parameters to the link name DDCARD using /SET-FILE-LINK (in ISP format, /FILE).
2. Software AG recommends that you link the ADAUSER module to user programs in a TP environment; for example, COBOL.

Multiuser Mode Example (OS/390 or z/OS)

The following example assumes that the user program USERPROG has been linked with the module ADAUSER and is to be executed in multiuser mode.

```
// EXEC PGM=USERPROG
//STEPLIB   DD        ... User Library
//          DD        ... Adabas Library
//DDCARD    DD *
ADARUN MODE=MULTI
//DDPRINT   DD SYSOUT=*
//... user  DD statements ...
```

Multiuser Mode Example (VM/ESA or z/VM)

User programs running under VM/ESA or z/VM in multiuser mode cannot use ADARUN to control other programs.

Multiuser Mode Example (VSE/ESA)

The following example assumes that the user program USERPROG has been linked with the module ADAUSER and is to be executed in multiuser mode.

```
//....user program assignments....
//  LIBDEF PHASE,SEARCH=(user-library, ADABAS-library)
//  EXEC USERPROG
ADARUN MODE=MULTI
/*
```

If the user program reads statement input, one of the following applies:

- If all user statements are read before the first Adabas call, they must immediately follow the EXEC statement and be followed by /*. The user file must be opened, read, and closed before the first Adabas call.
- If the first Adabas call is made before the first user statement is read, the user statements must follow the ADARUN parameter statements and start with a /* statement.
- Otherwise, the ADARUN parameter statements must be read from file CARD on tape or disk.

Notes:

1. ADARUN and/or the Adalink, ADAIOR, ADAMOD, and other Adabas modules must be available for dynamic loading during execution.
2. The Adabas load library must be available during execution so that required modules can be dynamically loaded.

Execution in Single-User Mode

In single-user mode, the appropriate Adabas nucleus JCL must be included with the JCL of the user program. This includes job control statements to define the Adabas datasets for the Associator, Data Storage, the Work dataset, and any datasets for protection or command logging. For more information about Adabas runtime job control requirements, see *Adabas Session Execution*.

Note:
User programs that use VM/ESA or z/VM facilities cannot access an Adabas nucleus in single-user mode.

Support for OpenEdition OS/390 Adabas Clients

A client running under OpenEdition OS/390 or z/OS can access Adabas. An OpenEdition application containing calls to Adabas can be linked with ADALNK (option 1) or ADAUSER (option 2).

Software AG recommends that you link your OpenEdition application with ADAUSER (option 2) for the following reasons:

- the application is not tied to a specific database ID and SVC number, or Adabas release;
- the DDPRINT output provides information about the database ID and SVC number used, as well as diagnostic information in case of error (DDPRINT output is lost when using option 1); and

- the program occupies less space in the hierarchical file system (HFS).
- Option 1 : Link OpenEdition Application with ADALNK
- Option 2 : Link OpenEdition Application with ADAUSER
- Setting the OpenEdition Shell Variable STEPLIB
- Limitations for OpenEdition Support

Option 1 : Link OpenEdition Application with ADALNK

An OpenEdition application that contains calls to Adabas can be linked with the module ADALNK. The database ID and SVC number must be zapped into the Adabas CSECT of the linked module at the offsets described in section *Writing User Exits for an Adalink* of the Adabas Installation documentation.

The following sample ZAP and link job has the following steps:

- COPYLNK: copy module ADALNK to another library
- ZAPLNK: zap the copied ADALNK module
- BINDAPP1: link (bind) the application with the zapped ADALNK into OpenEdition

```
//*
//* COPY AND RENAME ADALNK
//*
//
COPYLNK

  EXEC  PGM=IEBCOPY
//INLIB    DD    DSN=ADABAS.load.library,DISP=SHR
//OTLIB    DD    DSN=ADABAS.lnk.library,DISP=SHR
//SYSPRINT DD    SYSOUT=*
//SYSIN    DD    *
COPY INDD=INLIB,OUTDD=OTLIB
SELECT MEMBER=((ADALNK,ADALNKOE,R))
/*
//*
//* ZAP DBID AND SVC INTO COPIED ADALNK
//*
//
ZAPLNK

 EXEC PGM=IMASPZAP
//SYSPRINT DD SYSOUT=*
//SYSLIB DD DSN=ADABAS.lnk.library,DISP=SHR
//SYSIN DD *
NAME ADALNKOE ADABAS
VER 0080 0001                    DEFAULT DBID 1
VER 0084 0AF9                    DEFAULT SVC 249
REP 0080 00D3       <======   CHANGE TO USER DBID (HERE DBID 211)
REP 0084 0AE8       <======   CHANGE TO USER SVC  (HERE SVC 232)
/*
//*
//* BIND APPLICATION
//*
//
BINDAPP1
```

```
 EXEC PGM=IEWBLINK,
//             PARM='LIST,LET,XREF,MAP,CASE=MIXED'
//SYSPRINT DD  SYSOUT=*
//SYSLMOD  DD  PATH='/u/group/user',
//         PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//         PATHMODE=(SIRWXU,SIRWXG,SIRWXO)
//APPLIB   DD  DSN=your.appl.library,DISP=SHR
//LNKLOAD  DD  DSN=ADABAS.lnk.library,DISP=SHR
//SYSLIN  DD  *
  INCLUDE APPLIB(applname)
  INCLUDE LNKLOAD(ADALNKOE)
  ENTRY applent
  NAME oeappl(R)
/*
```

Option 2 : Link OpenEdition Application with ADAUSER

An OpenEdition application that contains calls to Adabas can be linked with the module ADAUSER.

Additionally, a member "ddcard" must be set up in the OpenEdition hierarchical file system (HFS) to contain the ADARUN parameters required by the client; for example:

```
ADARUN PROG=USER,DBID=211,SVC=232,MODE=MULTI
```

Prior to the first call to Adabas, the application must set the current working directory (using the chdir() function, for example) to the directory where file "ddcard" is located. As the application runs, Adabas searches the current working directory for member "ddcard", and extracts the parameters. Additionally, Adabas directs the DDPRINT output to member "ddprint" of the current working directory.

Note:
Member names "ddcard" and "ddprint" are case-sensitive. Member name "DDCARD" is not valid and will be ignored.

The following sample link job has one step:

- BINDAPP2: link (bind) the application with ADAUSER into OpenEdition

```
//*
//* BIND APPLICATION
//*
//
BINDAPP2

 EXEC PGM=IEWBLINK,
//             PARM='LIST,LET,XREF,MAP,CASE=MIXED'
//SYSPRINT DD  SYSOUT=*
//SYSLMOD  DD  PATH='/u/group/user',
//         PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//         PATHMODE=(SIRWXU,SIRWXG,SIRWXO)
//APPLIB   DD  DSN=your.appl.library,DISP=SHR
//ADALOAD  DD  DSN=ADABAS.load.library,DISP=SHR
//SYSLIN  DD  *
  INCLUDE APPLIB(applname)
```

```
  INCLUDE ADALOAD(ADAUSER)
  ENTRY applent
  NAME oeappl(R)
/*
```

Setting the OpenEdition Shell Variable STEPLIB

For both options, the OpenEdition shell variable STEPLIB must be set to ensure access to the Adabas load library. The following sample job sets the variable from OpenEdition running in batch mode:

```
//*
//OEBATCH EXEC PGM=BPXBATCH,
//             PARM='PGM /u/group/user/oeappl'
//STDIN   DD
PATH='/u/group/user/oeappl.in',PATHOPTS=(ORDONLY)
//STDOUT  DD   PATH='/u/group/user/oeappl.out',
//             PATHOPTS=(OWRONLY,OCREAT),PATHMODE=SIRWXU
//STERR   DD   PATH='/u/group/user/oeappl.err',
//             PATHOPTS=(OWRONLY,OCREAT),PATHMODE=SIRWXU
//STDENV  DD   *
STEPLIB=ADABAS.load.library
/*
//
```

Limitations for OpenEdition Support

Support is *not* available for running the following under OpenEdition:

- the Adabas nucleus or utilities
- clients running in single-user mode (MODE=SINGLE)
- clients running in 24-bit addressing mode (AMODE 24)