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 (ACB) Options
- Extended Adabas Control Block (ACBX) Options
- Programming Conventions for Issuing Direct Calls
- Using the CICS COMMAREA or TWA with COBOL Programs
- Using the Adabas API in Batch Mode
- Support for OpenEdition z/OS 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.

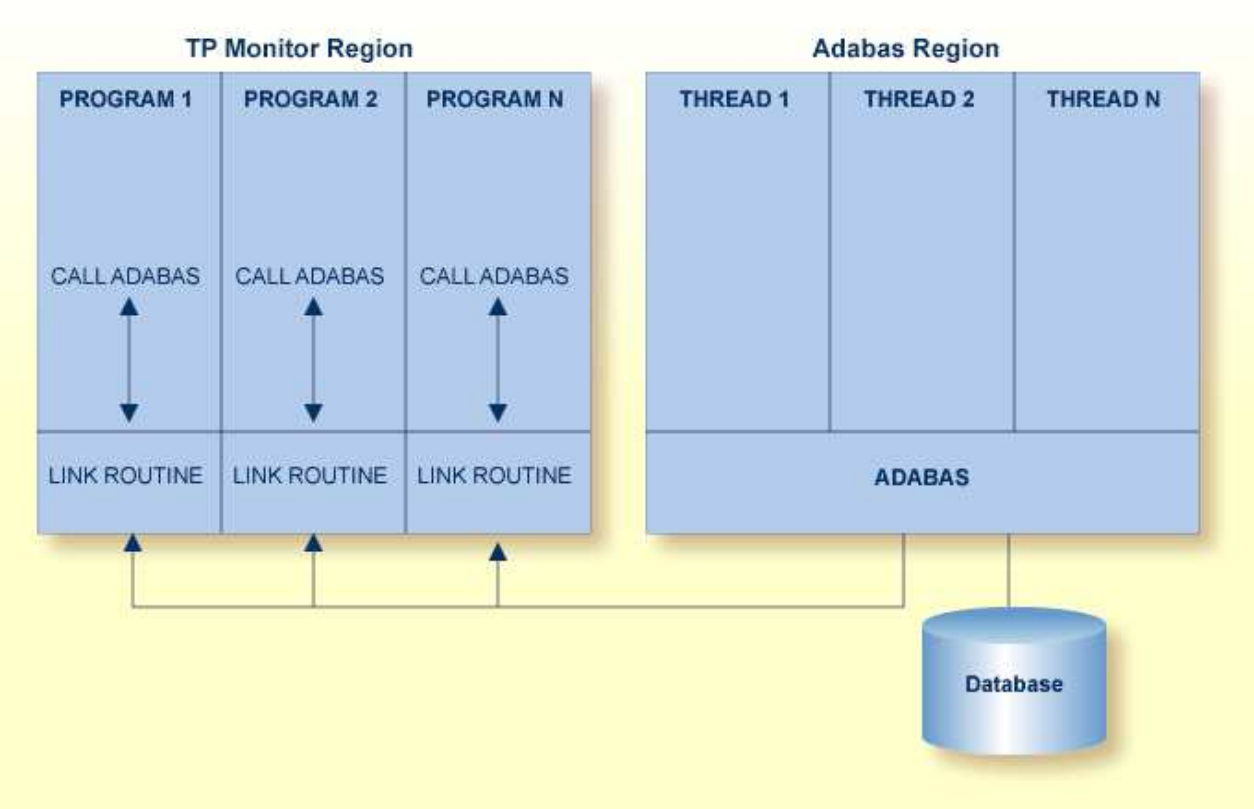
4. The Adabas API usually uses the Adabas router or SVC (supervisor call) installed on the operating system to send the formatted request to Adabas.
5. The Adabas router or SVC moves the user's control block and buffers from the TP monitor region to the Adabas region (into the Adabas nucleus).
6. The 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 batch link 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 Member	TP Monitor
ADACICS	CICS command-level under Adabas 7 and 8
ADACICT	Adabas 7 and 8 CICS task-related user exit (TRUE)
ADALCO	Executable module for Com-plete under Adabas 8.
ADALCO8	Base module for Com-plete under Adabas 8. This member must be linked with a link globals module you prepare and with any link routine exits you require to create the final ADALCO load module that is loaded by Com-plete to service Adabas 8 calls when Com-plete is initialized.
ADALNI	Executable module for IMS TM under Adabas 8 on z/OS systems.
ADALNI8	Base module for IMS TM under Adabas 8 on z/OS systems.
ADALNK	Executable module for batch/TSO under Adabas 7 and 8 on z/OS systems.
ADALNK8	Base module for batch/TSO under Adabas 7 and 8 on z/OS systems.
ADALNKR	Executable reentrant batch/TSO module under Adabas 7 and 8 on z/OS systems
ADALNKR8	Base reentrant batch/TSO module under Adabas 7 and 8 on z/OS systems
LNCSTUB	High-performance CICS stub for Adabas 7 and 8

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)
ADALNB	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.

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

Adabas 8 link routines do not require that the application program obtain a work area, except if your applications make ACB interface calls to the Adabas 8 batch/TSO reentrant link routine (ADALNKR). This section describes the calling requirements peculiar to the ADALNKR routine and how applications mixing ACB and ACBX calls to the ADALNKR routine should be programmed:

- Adabas 8 Batch/TSO Reentrant Link Routine (ADALNKR) Calling Requirements
- Mixing ACB and ACBX Interface Direct Calls to ADALNKR

Adabas 8 Batch/TSO Reentrant Link Routine (ADALNKR) Calling Requirements

Your existing Adabas 7 applications can call the Adabas 8 ADALNKR link routine without modification.

If your application makes ACB interface direct calls using the Adabas 8 batch/TSO reentrant link routine (ADALNKR), the seventh parameter must provide a four-byte fullword address that for a work area that must be initially set to binary zeros before the first call to ADALNKR. The contents of this work area should not be modified by calling application programs thereafter. The Adabas 8 ADALNKR routine uses this work area to store the reentrancy token. If this area is modified improperly, results are unpredictable and may range from poor performance to abnormal termination of the link routine.

Likewise, if your application makes ACBX interface direct calls using the Adabas 8 batch/TSO reentrant link routine (ADALNKR), the ACBX call’s reentrancy token (APLXRTOK) field must initially be set to binary zeros and remain unchanged by calling application programs thereafter.

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.

Mixing ACB and ACBX Interface Direct Calls to ADALNKR

The Adabas 8 ADALNKR routine accepts calls with the Adabas 7 parameter list with one change. To relieve the application code of the burden of obtaining the work area required by these reentrant modules, the seventh parameter should address a four-byte area initialized to binary zero, into which the ADALNKR routine will place a reentrancy token (as described earlier). This reentrancy token must remain intact between calls. If Adabas 7 and Adabas 8 calls are intermixed from the same application, this token must also be placed in the reentrancy token (APLXRTOK field) in the ACBX call.

If you provide the seventh parameter on an ACB direct call, be sure to specify the address used as the seventh parameter also in the ACBX call's reentrancy token (APLXRTOK field) to preserve user context between the logic performed by the two types of calls. Likewise, if you provide a reentrancy token on an ACBX direct call, be sure to specify the same address as the seventh parameter of the ACB direct call. If the work area address is not used in this way across ACB and ACBX direct calls, the user context for the two calling types will be different, and this could lead to incorrect results for the application.

For more information about the structure of an ACBX direct call, read *Specifying an ACBX Interface Direct Call*.

Note that the work area used by all Adabas 8 link routines is substantially larger than the equivalent work area used in Adabas 7. This is necessary to support the additional data structures introduced in Adabas 8 and the additional context information Adabas 8 will support.

Note:
If the Adabas 8 non-reentrant ADALNK is invoked using both ACB and ACBX interface direct calls, the context will be preserved because the work area is part of the ADALNK module itself.

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.

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

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

The Adabas 8 CICS link routine components are fully reentrant. This is possible because the Adabas task-related user exit (TRUE) is no longer optional. Because these modules are fully reentrant, they may be loaded from LPA and the task-related user exit (TRUE), ADACICT, may be defined as thread safe to CICS. To support fully reentrant and thread safe operation and to support the larger work areas required for Adabas 8, the storage working set of these link routine components has increased. Where possible, all storage areas used by the CICS Adabas 8 components are obtained above the 16 megabyte line.

Adabas Control Block (ACB) Options

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

The first byte of the ACB 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 ACB are reserved for use by Software AG.

This section covers the following topics:

- Using One-Byte File Numbers
- Using Two-Byte File Numbers
- Using Both One- and Two-Byte File Numbers in a Single Application
- Using COBOL to Set the Control Byte

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 ACB.
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 ACB.
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 ACB 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 ACB 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.

Extended Adabas Control Block (ACBX) Options

The first parameter passed to the Adabas API by the application program in an ACBX interface direct call is a pointer to the extended Adabas control block (ACBX). The ACBX contains information needed to process an Adabas request.

The first byte of the ACBX is used by the Adabas API to determine the processing to be performed. When issuing an Adabas command, set this field to binary zeros. This indicates that a logical user call is being made (ACBXTUSR equate)

The following values in ACBXTYPE are reserved for use by Software AG and are therefore not accepted by application programs: X’04’, X’08’, X’0c’, X’10’, X’14’, X’18’, X’1c’, X’20’, X’24’, X’28’, X’2c’, X’34’, X’38’, X’3c’, X’44’, X’48’, and X’4c’.

Applications written in Software AG’s Natural language need not include this first byte of the Adabas ACBX because Natural supplies appropriate values.

This section covers the following topics:

- Specifying File Numbers
- Specifying Database IDs

Specifying File Numbers

For an application program issuing Adabas commands using the ACBX interface, specify the file number in bytes 21-24 (ACBXFNR field) of the ACBX.

- For a one-byte file number, enter the file number in the rightmost byte (24); the leftmost bytes (21-23), should be set to binary zeros (B’0000 0000’).
- For a two-byte file number, use the rightmost bytes (23-24) of the field and set the leftmost bytes (21-22) to binary zeros.

To specify the database ID of a file, use bytes 17-20 (ACBXDBID field) of the ACBX. For complete information about the layout of the ACBX, read *Extended Adabas Control Block (ACBX)*.

Specifying Database IDs

For an application program issuing Adabas commands using the ACBX interface, specify the database ID in bytes 17-20 (ACBXDBID field) of the ACBX.

Specify the database ID for a call in the rightmost two bytes of this field, setting the leftmost bytes to binary zeros. At this time, only two-byte database IDs are support by Entire Net-Work.

If this field is set to binary zeros, the Adabas API uses either the database ID from the ADARUN cards provided in DDCARD input data, or the default database ID value provided in the LNKGBLS module linked with or loaded by the link routine.

If a file number is also needed, specify it in bytes 21-24 (ACBXFNR field) of the ACBX. For complete information about the layout of the ACBX, read *Extended Adabas Control Block (ACBX)*.

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 TM

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 ACB interface direct 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 ADACICS and the CICS Transaction Work Area (TWA) or the CICS COMMAREA to communicate parameters.

The Adabas 8 CICS link routine components accept application calls using either the TWA or the CICS COMMAREA. The COMMAREA is preferred. Software AG strongly recommends that any new CICS applications that will make ACBX interface direct calls use the COMMAREA instead of the TWA. The new ACBX direct call interface, with its variable number of Adabas buffer descriptions (ABDs) means that the Adabas parameter list length is of indeterminate size. Therefore, sizing the TWA for each transaction is more difficult using ACBX direct calls, whereas the size of the COMMAREA used is dynamically controlled by the application and is therefore better suited for ACBX calls. For information about the format of the COMMAREA, read *COMMAREA Formats*, later in this section.

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.

- Using the TWA With an ACB Call
- COMMAREA Formats

Using the TWA With an ACB Call

The addresses of the Adabas control block and buffers are passed in the same way for all releases of CICS. In an ACB call, these addresses must be placed in the first six words of the TWA.

To place the ACB 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 of an ACB call 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.

This section covers the following topics:

- Addressing the CICS TWA : Assembler
- Addressing the CICS TWA : PL/I
- Addressing the CICS TWA: VS COBOL
- Addressing the CICS TWA: 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);
```

Addressing the CICS TWA: 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.
 .
 .
 .
```

Addressing the CICS TWA: 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.
 .
 .
 .
```

COMMAREA Formats

Different COMMAREA formats are provided, one for Adabas 7 releases and one for Adabas 8.

The format of the Adabas 7 CICS COMMAREA, which must be at least 32 bytes long, is:

```
COMMID   DC   CL8'ADABAS52'       Adabas COMMAREA ID
CACBA    DC   A(ADACB)            Address of Adabas 7 CB
CAFBA    DC   A(ADAFB)            Address of Adabas 7 FB
CARBA    DC   A(ADARB)            Address of Adabas 7 RB
CASBA    DC   A(ADASB)            Address of Adabas 7 SB
CAVBA    DC   A(ADAVB)            Address of Adabas 7 VB
CAIBA    DC   A(ADAIB)            Address of Adabas 7 IB
```

The layout of the Adabas 8 CICS COMMAREA is:

```
V8COMID  DC   CL8'ADABAS8X'       Adabas V8 COMMAREA ID
V8APLX   DS   0A                  Beginning of Adabas V8 APLX
V8ACBX   DC   A(ACBX)             Adabas ADACBX
V8RSV1   DC   A(0)                ACBX direct call reserved field
V8APLXR  DC   A(0)                ACBX direct call reentrancy token
V8ABD1   DC   A(ABD1)             First ABD
         ...
V8ABD#n  DC   A(ABD#N+X'800000')  Last ABD address
```

IMS TM

IMS message processing programs that use the Adabas API require no special link. No reentrant option is supported.

The IMS link routine for Adabas 8 is ADALNI.

This section covers the following topics:

- Adabas ACB Call Using IMS TM (Nonreentrant)

Adabas ACB Call Using IMS TM (Nonreentrant)

A nonreentrant Adabas API call under IMS TM 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.
 .
 .
```

Using the CICS COMMAREA or TWA with COBOL Programs

Modern COBOL programs may use either the CICS Transaction Work Area (TWA) or the CICS Communications Area (COMMAREA) to pass data to the Adabas CICS link routines. For calls using the Adabas Version 8 API, the CICS COMMAREA is preferred because it is dynamically sized on a per call basis. The Adabas CICS high-performance stub routine (LNCSTUB) uses the CICS COMMAREA if the ADAGSET keyword PARMTYP is set to ALL or COMM.

- Using the TWA
- Using the CICS COMMAREA

Using the TWA

In previous Adabas releases and with older COBOL compilers it was necessary to assemble and link-edit the Software AG-provided subprogram ADASTWA with COBOL programs to properly set the Adabas parameter addresses in the CICS Transaction Work Area.

A skeleton of such a program might have the following kinds of statements:

```
WORKING-STORAGE SECTION.
        ...
        ...
      01  ADABAS-CB       PIC X(80).
      01  ADABAS-FB       PIC X ...
      01  ADABAS-RB       PIC X ...
      01  ADABAS-SB       PIC X ...
      01  ADABAS-VB       PIC X ...
      01  ADABAS-IB       PIC X ...
        ...
        ...
     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) C

    PROCEDURE DIVISION
        ...
        ...
      CALL 'ADASTWA' USING TWA,
                     ADABAS-CB,
                     ADABAS-FB,
                     ADABAS-RB,
                     ADABAS-SB,
                     ADABAS-VB,
                     ADABAS-IB.

      EXEC CICS LINK PROGRAM (LINK-NAME) END-EXEC.
        ...
        ...
```

The use of the ADASTWA subprogram is perfectly acceptable with modern compilers but because these compilers offer a limited means for address manipulation, the above skeleton may be changed as follows to avoid the need to link the ADASTWA program with CICS COBOL applications.

```
    WORKING-STORAGE SECTION.
        ...
      01 TWA-LEN              PIC S9(04) COMP VALUE +0.
      01 ABEND-CODE           PIC X(4) VALUE SPACES.
        ...
      01  ADABAS-CB       PIC X(80).
      01  ADABAS-FB       PIC X ...
      01  ADABAS-RB       PIC X ...
      01  ADABAS-SB       PIC X ...
      01  ADABAS-VB       PIC X ...
      01  ADABAS-IB       PIC X ...
        ...
        ...
    LINKAGE SECTION.
        ...
      01 TWA.
         05 CB-PTR           USAGE POINTER.
         05 FB-PTR           USAGE POINTER.
         05 RB-PTR           USAGE POINTER.
         05 SB-PTR           USAGE POINTER.
         05 VB-PTR           USAGE POINTER.
         05 IB-PTR           USAGE POINTER.
```

```
     ...

PROCEDURE DIVISION USING TWA.
     ...
INIT-TWA.
    EXEC CICS ASSIGN TWALENG (TWA-LEN) END-EXEC.
    IF TWA-LEN = 0
       MOVE 'U649' TO ABEND-CODE
       GO TO ...
    IF TWA-LEN < 28
       MOVE 'U650' to ABEND-CODE
       GO TO ...
    EXEC CICS ADDRESS TWA (ADDRESS OF TWA) END-EXEC.
     ...
    SET  CB-PTR TO ADDRESS OF ADABAS-CB.
    SET  FB-PTR TO ADDRESS OF ADABAS-FB.
    SET  RB-PTR TO ADDRESS OF ADABAS-RB.
    SET  SB-PTR TO ADDRESS OF ADABAS-SB.
    SET  VB-PTR TO ADDRESS OF ADABAS-VB.
    SET  IB-PTR TO ADDRESS OF ADABAS-IB.
     ...
    EXEC CICS LINK PROGRAM (LINK-NAME) END-EXEC.
     ...
```

Using the CICS COMMAREA

The CICS TWA size is part of the CICS transaction definition for a given set of application programs and its size may not be dynamically altered during execution of application programs running under that CICS transaction. For this reason the CICS Communications Area (COMMAREA) is recommended for Adabas Version 8 programs. The COMMAREA size may be altered as needed by the requirements of the data to be passed between programs using it. The Adabas Version 8 parameter list and data structures are dynamic in size, so the COMMAREA is the best method for passing data between CICS applications and the Adabas Version 8 CICS link routines.

Using the address manipulation features of a modern COBOL compiler, a sample skeleton of the statements needed to use the CICS COMMAREA might be:

```
WORKING-STORAGE SECTION.
     ...
     ...

  01 ADA-COMM-AREA.
    05 COMMID           PIC X(8) VALUE 'ADABAS52'.
    05 CB-PTR           USAGE POINTER.
    05 FB-PTR           USAGE POINTER.
    05 RB-PTR           USAGE POINTER.
    05 SB-PTR           USAGE POINTER.
    05 VB-PTR           USAGE POINTER.
    05 IB-PTR           USAGE POINTER.
     ...
  01  ADABAS-CB       PIC X(80).
  01  ADABAS-FB       PIC X ...
  01  ADABAS-RB       PIC X ...
  01  ADABAS-SB       PIC X ...
  01  ADABAS-VB       PIC X ...
  01  ADABAS-IB       PIC X ...
     ...
  01 ADA-COMM-AREA-LENGTH PIC S9(4) COMP.
     ...
```

```
PROCEDURE DIVISION.
     ...
     ...
 INIT-ADA-COMM-AREA.
    SET  CB-PTR TO ADDRESS OF ADABAS-CB.
    SET  FB-PTR TO ADDRESS OF ADABAS-FB.
    SET  RB-PTR TO ADDRESS OF ADABAS-RB.
    SET  SB-PTR TO ADDRESS OF ADABAS-SB.
    SET  VB-PTR TO ADDRESS OF ADABAS-VB.
    SET  IB-PTR TO ADDRESS OF ADABAS-IB.
    MOVE 32 TO ADA-COMM-AREA-LENGTH.
     ...
    EXEC CICS LINK PROGRAM ('ADABAS')
         COMMAREA(ADA-COMM-AREA)
         LENGTH(ADA-COMM-AREA-LENGTH)
    END-EXEC.
```

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 or ADALNKR) 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 ADALNK or ADALNKR.

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

ADAUSER and ADARUN with the Adabas API

The ADAUSER module can 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 (z/OS)
- Link Example (z/VSE)

Link Example (BS2000)

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

Link Example (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 (z/VSE)

```
*  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 (z/OS)
- Multiuser Mode Example (z/VSE)
- 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 (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 (z/VSE)

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.

Note:
The Adabas load library must be available during execution so that required modules can be dynamically loaded. Alternatively, only extracts from this library can be used; in this case, the following modules must be included: ADAIOR, ADAIOS, ADALNK (which can be adjusted to your needs), and ADARUN. If the ADARUN parameters for multifetch or prefetch are used, the modules ADAMLF or ADAPRF are also necessary.

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 data sets for the Associator, Data Storage, the Work data set, and any data sets for protection or command logging. For more information about Adabas runtime job control requirements, see *Adabas Session Execution*.

Support for OpenEdition z/OS Adabas Clients

A client running under OpenEdition 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)