CA ADS™ Batch for CA IDMS™

ADS Batch User Guide

Release 18.5.00

CA Technologies Product References

This document references the following CA products:

- CA IDMS/DB
- CA IDMS/DC (CA IDMS/DC Transaction Server Option)
- DC/UCF (CA IDMS Universal Communications Facility)
- CA ADS
- CA ADS Batch

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: Introduction

This section contains the following topics:

Overview

CA Application Development System/Batch (CA ADS Batch) is a facility that allows you to develop fourth-generation batch applications that execute in the CA IDMS environment. CA ADS Batch applications can read data from input files, perform CA IDMS/DB database update and retrieval, and write data to output files.

Note: Do not confuse batch application *execution* with batch application *definition*. CA ADS Batch refers to Application Development System applications that *execute* in the batch environment. CA ADS Batch application components can be *defined* in online or batch mode.

Common Facilities

CA ADS Batch uses many of the same facilities as CA ADS, including:

- The **CA IDMS mapping facility**, used to define **maps** through which all transaction input and output is performed. In the CA ADS environment, map definitions control the transfer of data between a terminal operator's screen and dialog variable storage. In the CA ADS Batch environment, map definitions control the transfer of data between input and output files and variable storage.

  Maps provide extensive automatic editing and error handling features that simplify application development. These features, formerly available only in the online environment, are now also available in the batch environment.
- The **dialog compiler**, used to define **dialogs** that read, process, and write transactions at runtime.
- The **application compiler**, used to create a global structure of an application.
- The **runtime system**, used to execute applications.
- The **Integrated Data Dictionary** (IDD), used to define certain application components, such as process modules and file definitions, and to provide centralized documentation of all application components. The data dictionary also provides automatic cross-referencing of components, reporting facilities, and security features.

Differences

CA ADS Batch differs from CA ADS mainly in its handling of transaction input and output. Online transactions are typically mapped in and mapped out through a terminal. Batch transactions are typically read from and written to files, such as sequential data sets.

Batch Features

CA ADS Batch includes the following features in support of the batch runtime environment:

- **Input and output file support**—CA ADS Batch provides access to input and output sequential files, such as disk, tape, card, and printer, and to VSAM entry-sequenced data sets (ESDS) that are accessed sequentially. File records can be any type except variable spanned.
- **Suspense file support**—CA ADS Batch can write erroneous input file records to a suspense file. After an application has been executed, the user can correct the records in the suspense file, then resubmit them directly in another run of the application.
- **Log file support**—CA ADS Batch accumulates in a system log runtime information, such as error messages and dialog statistics. CA ADS Batch provides a log file archiving utility (z/OS), which writes log information to tape when the log file is full. CA ADS Batch also provides a print log utility that prints formatted reports of selected log file information.
- **Process commands**—CA ADS Batch uses commands to read from and write to input, output, and log files, and to send messages to the operator's console. Environment test conditions are also included (for example, IF $ONLINE THEN DISPLAY. ..., or, IF $BATCH THEN DO. ...); thus, a process module can be used in both the online and batch environments.

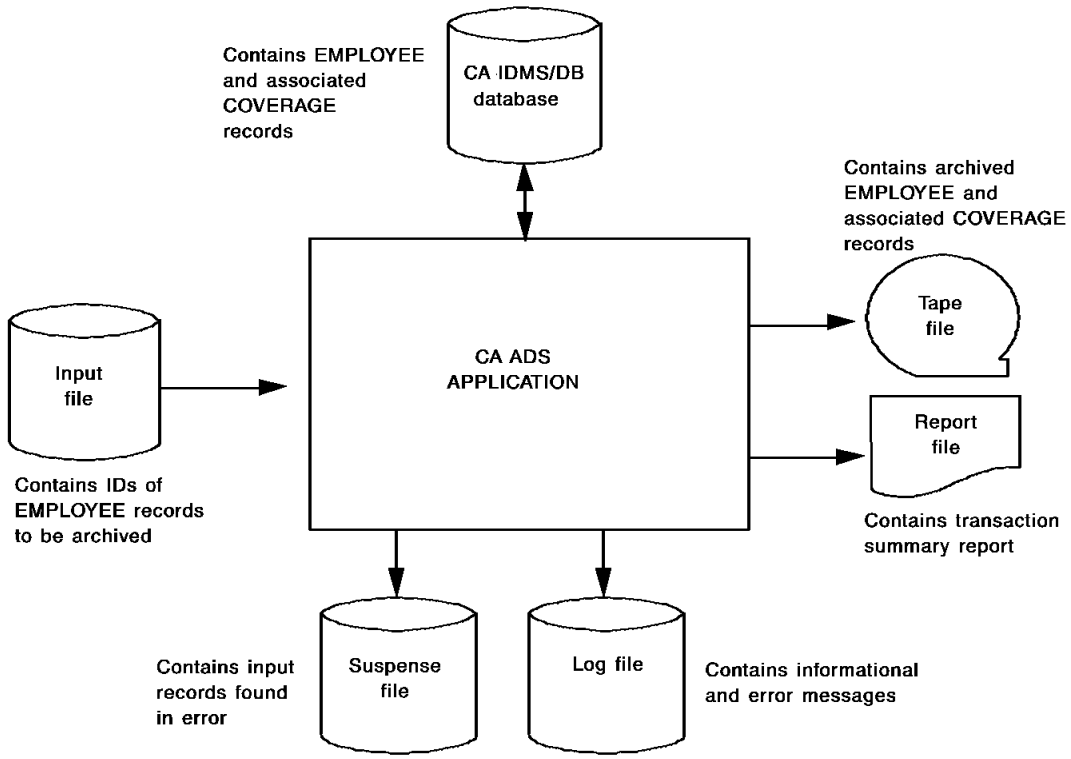
Sample Applications

The remainder of this chapter introduces two examples of CA ADS Batch applications. These examples are illustrated in detail in CA ADS Batch Sample Applications (see page 143).

Example 1: Employee-record Archiving Application

The employee-record archiving application writes selected employee records and their associated insurance coverage records from an CA IDMS/DB database to tape.

The figure below shows the I/O performed by the application.



Application Notes

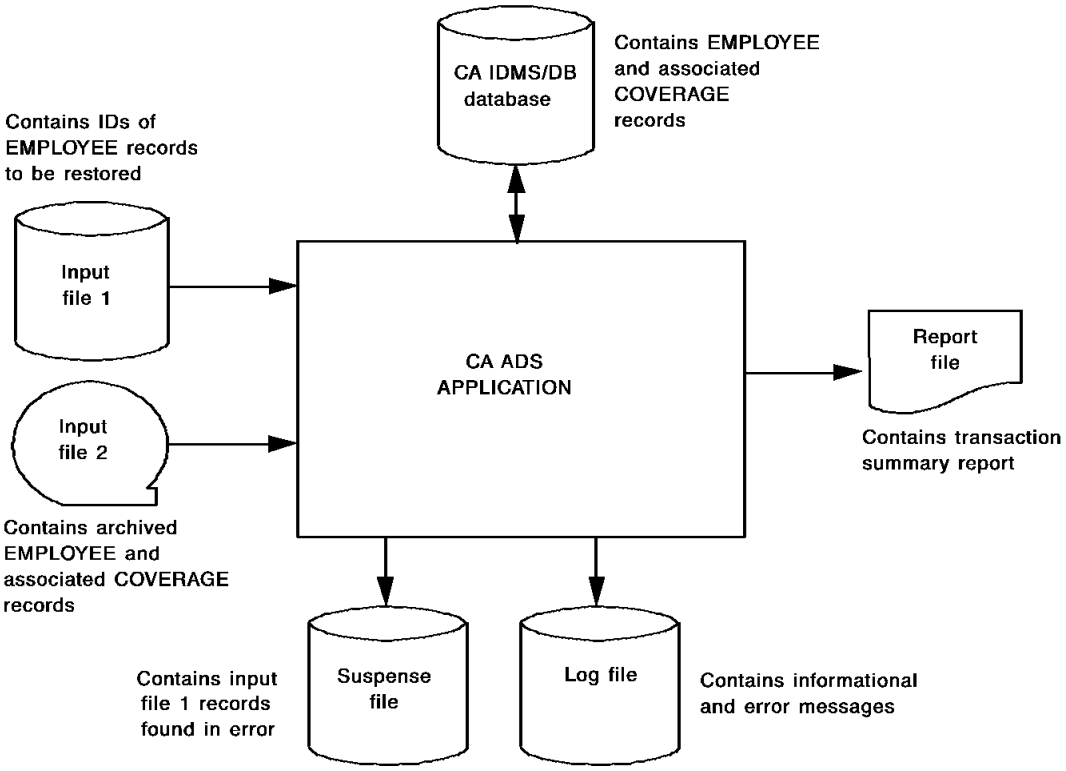
The following notes describe the application:

- Each input file record contains the id of an employee record to be archived.
- Each specified employee record is retrieved from the CA IDMS/DB database and written to an output file.
- All insurance coverage records associated with an employee being archived are retrieved from the database and written to the same output file.
- Coverage records are deleted as they are archived. Employee records are deleted after all associated coverage records have been archived.
- A transaction summary report file is created.
- Invalid input records are written to a suspense file.
- A log file accumulates error messages and other information.

Example 2: Employee-record Restore Application

The employee-record restore application restores to the database selected archived employee records and their associated coverage records.

The following figure shows the I/O performed by the employee-record restore application.



Application Notes

The following notes describe the application:

- One of the two input files contains the ids of employee records to be restored.
- The second input file contains the archived employee records and their associated coverage records.
- Based on employee ids specified in the first input file, the application restores employee and associated coverage records from the second input file.

- A transaction summary report file is created.
- A suspense file stores all invalid records from the first input file.
- No suspense file is allocated for the second input file. It is assumed that the archived employee and coverage records are formatted correctly. Note, however, that a suspense file can be allocated for each input file in an application.
- A log file accumulates error messages and other information.

Chapter 2: CA ADS Batch Concepts

This section contains the following topics:

Overview

Developing an CA ADS Batch application is, in many ways, similar to developing an online application with CA ADS. The major difference is that in CA ADS Batch you describe data transfer between variable storage and *files*, while in CA ADS, you describe data transfer between variable storage and *online terminals*.

Steps

To develop and execute an CA ADS Batch application, you perform the following steps:

1. **Create external file descriptions.** You describe your input and output files in the data dictionary with file, record, and element entities. You can define files by using DDDL.
2. **Define file maps.** You describe how data is to be transferred at runtime between the input and output files and variable storage. You can define file maps by using the online mapping facility.
3. **Create process modules.** You define in the data dictionary the process logic required by the application. The process command language includes new commands for batch processing. You can define process modules by using DDDL.
4. **Define dialogs.** You define dialogs that bring file maps and process modules together into an executable load module. You can define dialogs by using the online dialog compiler.

5. **Define the application structure.** As an optional step performed at any time before executing the application, you define an application structure that describes the application in terms of functions and responses. In the batch environment, application structures have a special use in creating applications that access input files with multiple record formats. You can define the application structure by using the online application compiler.
6. **Execute the application.** You execute the batch application by using the runtime system in batch mode. The application can access an CA IDMS/DB database, read from input files, and write to output files. In addition, the runtime system can write input records in error to a suspense file, and can write informational and error messages to a log file. A trace facility is available to help you debug an CA ADS Batch application.

While CA ADS Batch is based on concepts familiar to developers of CA ADS applications, it also includes concepts unique to batch application development. This chapter discusses the following CA ADS Batch concepts:

- Input and output files and file maps
- Suspense files
- Log files
- Batch dialog structure
- Application structure for applications defined using the application compiler

For more information about the new process commands, see Process Command Language (see page 35). The runtime system and the CA ADS Batch trace facility are described in Runtime Considerations (see page 79).

Input and Output Files and File Maps

Input and Output Files

Input files are sequential files from which batch transactions are read. **Output files** are sequential files to which batch transactions are written. CA ADS Batch provides access to sequential files, such as disk, tape, card, and printer, and to VSAM entry-sequenced data sets (ESDS) that are accessed sequentially. Records in input and output files (that is, file records) can be any type except variable spanned.

Note: CA ADS Batch does not support user and nonstandard tape labels. On input, tape labels are bypassed; on output, they cannot be written.

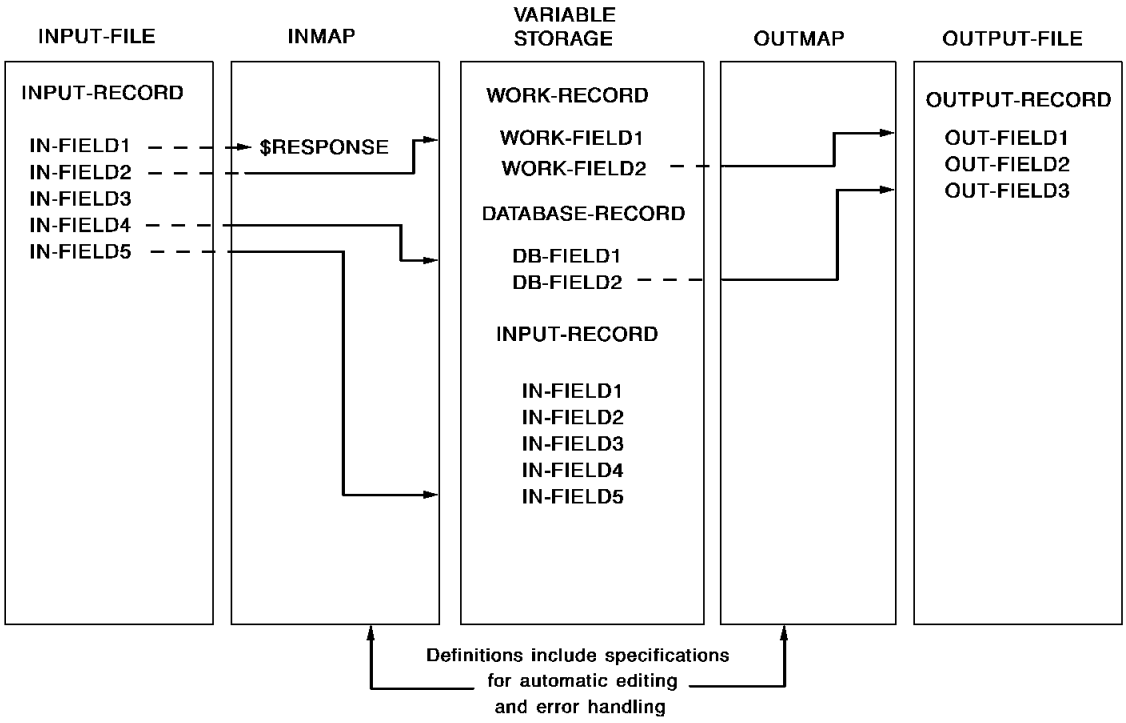
Describing Files

You describe files, consisting of file, record, and element entities, in the data dictionary using the IDD DDDL compiler.

Note: For more information about describing file, record, and element entities, see the *CA IDMS IDD DDDL Reference Guide*. For more information about specifying file characteristics, see Runtime Considerations (see page 79).

File Maps

A **file map** is defined for each file record type in an application. File maps are used at runtime to transfer data between file record elements and data fields in variable storage. The diagram below shows two file map definitions.



Mapin Operation

An **input file map**, such as INMAP in the above diagram, transfers data from fields in an input file record to data fields in variable storage. Reading a file record using a map is called a **mapin operation**.

Mapout Operation

An **output file map**, such as OUTMAP, transfers data from data fields in variable storage to fields in an output file record. Writing a file record using a map is called a **mapout operation**.

Components of a File Map

A file map consists of one external record and zero or more internal records, as follows:

- An **external record** describes the layout of data in a file. INPUT-RECORD of INPUT-FILE is the external record for map INMAP. OUTPUT-RECORD of OUTPUT-FILE is the external record for map OUTMAP.
- **Internal records** describe data in variable storage to which external record fields map. WORK-RECORD and DATABASE-RECORD are internal records for both INMAP and OUTMAP.

Fields Mapping to Themselves

Fields can map to themselves. For example, in the diagram, IN-FIELD5 maps to itself. On a mapin operation at runtime, IN-FIELD5 in the input buffer is moved into IN-FIELD5 in variable storage. Automatic editing and error handling can be performed on the field; the length of the field, however, cannot be changed.

Note: If a field maps to itself, the entire record is available for access and update as an internal record in variable storage; however, only those fields included in the map definition are mapped into or out of variable storage. In the example, INPUT-RECORD is available in variable storage because IN-FIELD5 maps to itself. On a mapin operation, however, data is moved only to IN-FIELD5 of INPUT-RECORD; nothing is moved to the other fields of the record.

Internal Record Fields

Map process commands refer to internal record fields. For example, a process module whose dialog is associated with INMAP could issue commands such as the following:

```
IF FIELD WORK-FIELD1 IS IN ERROR...

MODIFY MAP FOR FIELD DB-FIELD1 EDIT IS ERROR.
```

A map process command could also reference IN-FIELD5, but could not reference any of the other fields in INPUT-RECORD.

Input and Output File Maps

File maps can be used for both input and output. You specify that a file map is for input or output during dialog definition when you associate the map with a dialog. A map can be the input file map for one dialog and the output file map for another dialog. A map can be the input and the output file map for the same dialog. In the diagram, INMAP could be associated with a dialog as an output file map, and vice versa for OUTMAP.

Accessing Different Data Sets

The same map can access different data sets. You associate input and output file maps with data sets on a dialog-by-dialog basis. In this way, a single file map can be used to access several data sets in an application.

For example, in an application that copies one data set to another, only one file map is necessary. The map is the input file map for one dialog, and in that capacity, is associated with the input data set. The same map is the output file map for the same or another dialog, and in that capacity, is associated with the output data set.

Automatic Editing and Error Handling

Automatic editing and error handling specifications can be part of the map definition. For example, a field can be converted from a DISPLAY field to a COMPUTATIONAL field. Edit and code tables can be used to check fields for valid values and to convert fields from one value to another.

Response Fields

A **response field** can be included in the definition of a file map. You specify that a field is a response field by associating it with $RESPONSE during map definition. In the example, IN-FIELD1 is a response field. On a mapin operation at runtime, the value of the response field can determine the next dialog response process or application function that is executed, just as in the online environment.

An output file map can also include a response field. On a mapout operation, the value of the $RESPONSE system-supplied data field is mapped out to external record fields as appropriate.

A response field can be a single field or the concatenation of several fields in a file record. When two or more fields are used to form the response field, you specify the sequence in which the fields are concatenated. For example, a record consisting of fields A through F can have a response field made up of the concatenation of fields D, B, and F, in that order. The maximum length of the concatenated response field is 32 bytes.

Fields that compose a response field must have a usage of DISPLAY. Note that response fields used in applications defined with the application compiler can have a maximum length of eight bytes.

Excluded Fields

Fields can be excluded from a map definition. For example, in the diagram, IN-FIELD3 is not included in the definition for INMAP, and OUT-FIELD2 is not included in the definition for OUTMAP. On a mapin operation, data in excluded fields is not mapped into variable storage.

On a mapout operation, the excluded fields are **initialized** according to the following rules:

- Alphanumeric fields not defined with a VALUE clause are initialized to spaces.
- Numeric fields not defined with a VALUE clause are initialized to zero in the proper data format.
- Fields defined with a VALUE clause are initialized to the specified value in the proper data format.

Note: Note that since the output buffer is initialized, you can have an external record with FILLER fields.

For more information about mapping in and mapping out, see Batch Dialog Structure (see page 23) and Application Structure (see page 30).

Suspense Files

Suspense files store input file records found to contain edit errors at runtime. An input record field can be set in error either at mapin by automatic editing and error handling, or during premap or response process execution by a map modification command.

Associating a Suspense File with a Dialog

One suspense file can be associated with each dialog that has an input file. You associate a suspense file with a dialog simply by requesting one, either during dialog definition or at runtime.

Writing to a Suspense File

An input record in error is written to a suspense file at runtime by the dialog that performed the mapin operation on the record. The record can be written either immediately after the mapin operation or on the next mapout operation, as follows:

- **Immediately after the mapin operation**, if the selected response process has an EXECUTE ON EDIT ERRORS specification of NO. The runtime system writes the record to the suspense file, then reads the next input record.
- **On a mapout operation**. The mapout operation is initiated by a WRITE TRANSACTION command. If the dialog's input record is not in error, WRITE TRANSACTION performs a mapout to the dialog's output file. If the input record is in error, WRITE TRANSACTION instead writes the input record to the suspense file.

  Note: You can write the input record to the suspense file even if the record has no fields in error by including the keyword SUSPENSE in the WRITE TRANSACTION command.

Note: If a suspense record is to be written by a dialog that is not associated with a suspense file, the record is not written. However, error messages are still sent to the log file, which is described later in this chapter. The suspense file identifier in the messages is '********'.

Maximum Number of Error Records

The maximum number of error records that can be written to a suspense file at runtime is a system-generation option (ADSOBSYS (see page 115)). By default, an unlimited number of error records can be written. The ADSOBSYS default can be overridden at runtime with control statements for each dialog.

Log Files

Log files store error messages and other information produced during application execution. You can specify log file characteristics by using the ADSOBSYS utility; these characteristics can be overridden at runtime using control statements. For more information about specifying log file characteristics, ADSOBSYS (see page 115) and Control Statements (see page 99).

Print Log Facility

CA ADS Batch provides a print log utility that enables the user to print information from the log based on selection criteria, such as the user id and the type of record. For more information about this utility, see CA ADS Batch Print Log Utility (see page 133).

The following topics are discussed below:

- Information stored in a log file
- The log file prefix
- Archiving

Log File Information

Types of Log Information

Log files can store the following types of information:

- **User** ––Informational messages issued by WRITE TO LOG and CONTINUE commands, and by WRITE TRANSACTION commands that perform a mapout to the dialog's output file. The message is specified in the MESSAGE parameter of the command.
- **Debug**––Debugging information issued by SNAP commands and by the CA ADS Batch trace facility.
- **Edit error**––Error messages issued when an input record is written to the suspense file. The runtime system writes one record consisting of the associated suspense file record number and the first 32 bytes of the input record, followed by one record for each applicable error message. In addition, if the input record was written to the suspense file by a WRITE TRANSACTION command, any message specified in the MESSAGE parameter of the command is also written to the log file as an edit error message.
- **Operator**––Operator informational messages issued by WRITE TO OPERATOR commands. All messages sent to the operator are also sent to the log.
- **Abend**––Abnormal termination messages issued by the ABORT command and by the runtime system when an application abends. Abend messages include snap dumps.

  Note: If the abend occurs during a mapping operation, the snap dump goes instead to the z/OS data set associated with the ddname IDMSSNAP file associated with the linkname IDMSSNAP or, in VSE, to SYSLST, as described under JCL (see page 110).
- **Statistics**––Statistics records written by the runtime system when dialog statistics are being collected for the application.

Log File Prefix

Information Specified

Each log file record can include a prefix that specifies the following information:

- The date and time the record was written
- The name of the dialog that wrote the record
- The type of record written (for example, error text or statistics)
- The application user id

This information is used by the print log utility to select log records and format them for output.

By default, a prefix is included when the log file output device is tape or disk, and is omitted when the output device is printer. The default can be overridden through the ADSOBSYS utility and/or at runtime with control statements.

For developers creating their own reports from information in the log file, the DSECT for the log file prefix is provided in CA ADS Batch Print Log Utility (see page 133).

Archiving

Unlimited Log Information

Archiving enables z/OS users to archive unlimited log information to tape at runtime. The runtime system writes log records to the first log file. When the file is full, the runtime system writes to the second log file and archives the first log file to tape. When the second log file is full, the runtime system writes to the first log file and the second log file is archived, and so forth.

When Archiving is not Requested

If archiving is not requested, the runtime system performs log file wraparound. Up to two log files can be allocated for an application. If both are full, the runtime system overwrites the log records already written, starting at the beginning of the first log file.

Batch Dialog Structure

The structure of an CA ADS Batch dialog is similar to that of an CA ADS dialog. Both consist of premap and response processes, maps, work records, and a subschema.

CA ADS Batch Differences

CA ADS Batch dialogs differ in a number of ways. Most importantly, batch dialogs use input and output file maps to transfer data between variable storage and files, while online dialogs use online maps to transfer data between variable storage and online terminals.

Note: Online and batch dialogs are differentiated by the type of map with which they are associated. Dialogs associated with online maps can be executed only in the online environment. Dialogs associated with file maps can be executed only in the batch environment. Mapless dialogs can be executed in either environment.

The following topics are discussed below:

- Components of an CA ADS Batch dialog
- Batch control events
- Batch response field values
- Batch dialog execution

CA ADS Batch Dialog Components

An CA ADS Batch dialog can include the following components:

- Work records
- A subschema
- An input file map
- An output file map
- A premap process
- Response processes

All of these components are optional. However, a dialog must have either a premap process, an input file map, or both. A dialog that has no input file map cannot have any response processes.

Batch Dialog Options

A batch dialog can include most of the dialog options available to online dialogs, such as the request for activity logging. Additional dialog options are available that enable you to specify default z/OS ddnames (z/VSE filenames) for a dialog's input, output, and suspense files. A suspense file is associated with a dialog when you specify its ddname or filename, either during dialog definition or at runtime.

Batch Control Events

Batch control events are conditions that arise during file input. Batch control events include:

- **End of file**—The most recent input file read operation resulted in an end-of-file condition.
- **I/O error**—The most recent input file read operation resulted in a physical error condition. Note that errors on output cause the runtime system to abend the application.

Associating Batch Control Events

Batch control events can be associated with dialog response processes and application responses. On a map in operation, the occurrence of a batch control event causes execution of its associated response process or of the application function invoked by its associated application response. For more information about specifying selection criteria for responses and response processes, see Response Definition Screen (see page 56) and "Response Process Definition Screen" in Chapter 5.

Testing Batch Control Events

Batch control events can also be tested in premap or response processes, as in the following example:

```
IF $EOF
  LEAVE APPLICATION.
```

For more information about testing for batch control events, see Batch Control Event Status Conditions (see page 41).

Batch Response Field Values

CA ADS Batch extends the online concept of response fields to the batch environment. An input file map can include one or more external fields that make up the map's response field. On a map in operation, the value of the response field helps to determine the next dialog response process or application function that is executed, just as in the online environment.

For example, an input record can have a response field whose value is ADD, MOD, or DEL. A dialog can be defined with one response process that is executed when the response field value is ADD, another when it is MOD, and another when it is DEL. The response process executed at runtime after the dialog performs a map in operation depends on the input record's response field value.

Input Files with Multiple Record Layouts

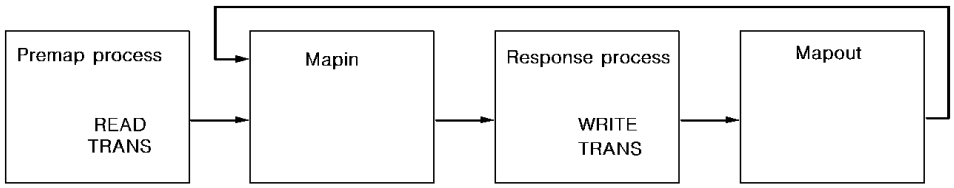
Response fields are especially useful in accessing input files that have multiple record layouts, as described under "Application structure" later in this chapter.

Batch Dialog Execution

Dialog Execution

Batch dialog execution is similar to online dialog execution: a batch dialog executes premap and response processes and performs mapin and mapout operations. The major difference between batch and online dialogs is in the sequence and handling of the mapin (read) and mapout (write) operations.

The diagram below shows a typical batch dialog. The dialog reads from an input file and writes to an output file.



Premap Process

The **premap process** is executed at the beginning of the dialog, unless the dialog's entry point is its mapin operation. The process executes until it issues a control command, including a READ TRANSACTION or WRITE TRANSACTION command. In the example shown above, a READ TRANSACTION command is issued, which terminates the process and passes control to the mapin operation.

Mapin Operation

The **mapin operation** is performed in any of the following cases:

- At the beginning of the dialog if the dialog's entry point is its mapin operation
- After a READ TRANSACTION command has been issued by a premap or response process
- After a mapout operation that has been issued by a WRITE TRANSACTION command that does not include the CONTINUE or RETURN keyword
- After a CONTINUE command has been issued in a dialog without a premap process

The mapin operation performs the following functions:

1. Reads a record from the dialog's input file into the file's input buffer
2. Maps all correct fields into variable storage according to the dialog's input file map definition
3. Selects a response process based on batch control event or response field value

Note: If the application is defined using the application compiler, the runtime system first examines the response field of the record and passes control to another application function, if required. Application execution using the application compiler is described later in this chapter.

Response Process

The selected **response process** is executed after the mapin operation. The response process executes until it issues a control command, including a batch READ TRANSACTION or WRITE TRANSACTION command. In the example, the response process issues a WRITE TRANSACTION command, which causes a mapout operation.

Mapout Operation

A **mapout operation** is performed when a premap or response process issues a WRITE TRANSACTION command. The mapout operation either maps a record to the dialog's output file or writes the input record to the dialog's suspense file, as follows:

- **A record is mapped to the output file** if the dialog's current input file record (if any) contains no errors and if the WRITE TRANSACTION command does not include the keyword SUSPENSE. The mapout operation writes a record to the dialog's output file from variable storage according to the dialog's output file map definition.
- **The input record is written to the suspense file** if the dialog's current input file record contains errors or if the WRITE TRANSACTION command includes the keyword SUSPENSE. Applicable error messages are written to the log file. Note that an input field can be set in error either automatically on mapin or by map modification commands in process code.

Determining How Control Will Be Transferred

After the mapout operation, transfer of control is determined by the keyword specified in the WRITE TRANSACTION command that caused the mapout operation:

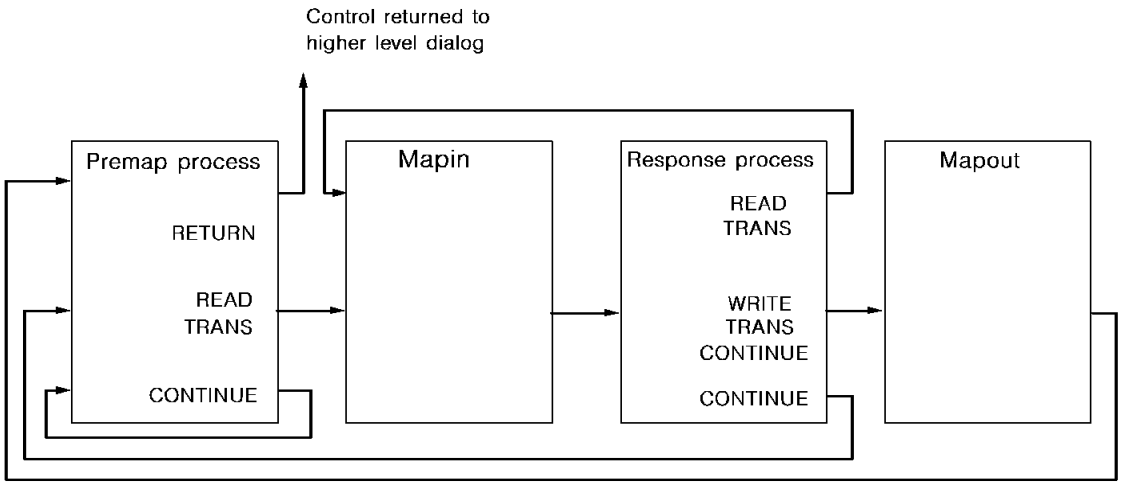
- **CONTINUE** executes the dialog's premap process.
- **RETURN** returns control to a higher level dialog or application function.

If no keyword is provided, the dialog's mapin operation is performed; this is the case below.

Note: Because the WRITE TRANSACTION command passes control to another part of the application, and because the command does not write to both the suspense file and the output file at the same time, it is difficult to write to both files if that is what you want. An alternative is to associate the output file map with a second dialog. The first dialog links to the second, which issues a WRITE TRANSACTION RETURN command. The command writes a record to the output file and returns to the first dialog. The first dialog then issues a WRITE TRANSACTION command that writes the input record to the suspense file.

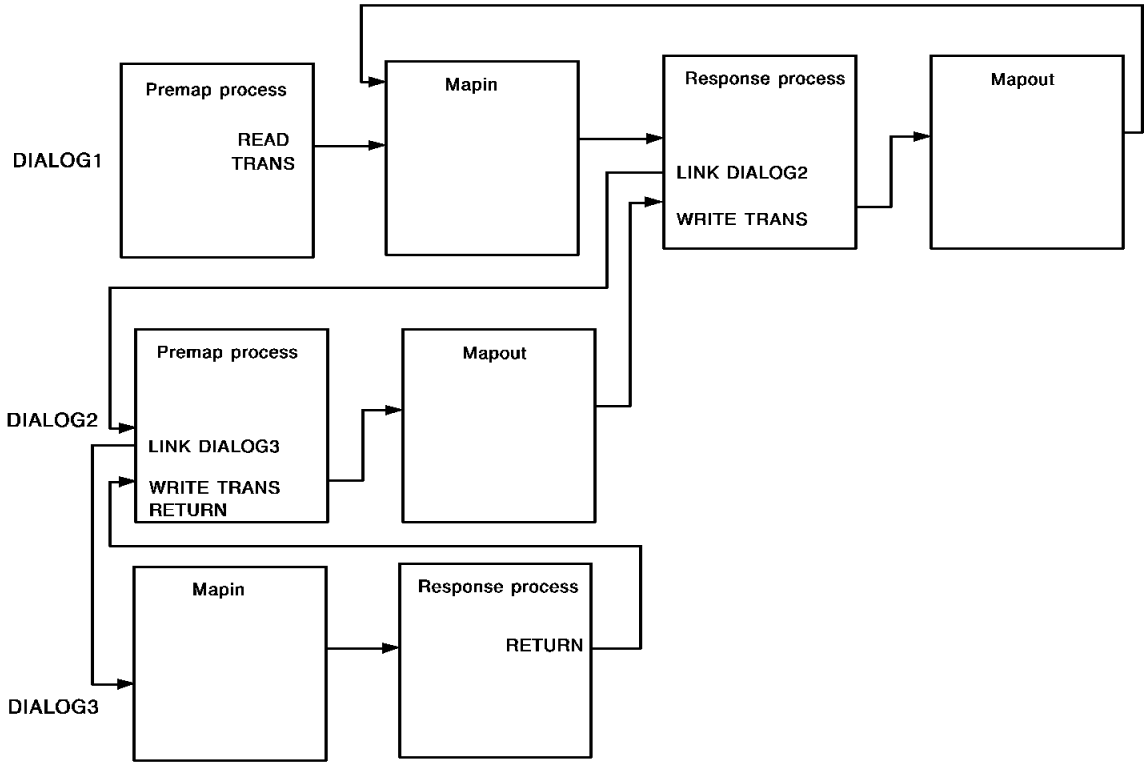
Control Command Example

Control commands, including READ and WRITE TRANSACTION commands, the READ TRANSACTION and WRITE TRANSACTION commands, affect runtime flow of control. A few of these commands are illustrated below.



Accessing Multiple Input and Output Files

Control commands enable a batch application to consist of multiple dialogs. Through multiple dialogs, an application can access multiple input and output files, as shown below.



Application Execution

Application execution is described below:

1. **DIALOG1's premap process** issues a READ TRANSACTION command, which reads a record from the dialog's input file, then selects a response process.
2. **DIALOG1's response process** issues a LINK command, which passes control to DIALOG2.
3. **DIALOG2's premap process** issues a LINK command, which passes control to DIALOG3.
4. **DIALOG3's mapin operation** (the dialog's entry point) reads a record from the dialog's input file, then selects a response process.
5. **DIALOG3's response process** issues a RETURN command, which returns control to DIALOG2 at the command that immediately follows the LINK command.

6. **DIALOG2's premap process** issues a WRITE TRANSACTION RETURN command, which performs a mapout operation, writing a record to the dialog's output file, and returns control to DIALOG1 at the command that immediately follows the LINK command.
7. **DIALOG1's response process** issues a WRITE TRANSACTION command, which performs a mapout operation, writing a record to the dialog's output file, and performs a mapin operation, reading another record from the dialog's input file and selecting a response process.

For detailed information on the flow of control in a mapin operation, see Runtime Flow of Control (see page 80).

Application Structure

Defining an Application Structure

An CA ADS Batch application structure can be defined using the application compiler. As with online applications, you define the application structure in terms of functions, responses, task codes, and global records.

Batch application structures differ from online application structures in the following ways:

- Certain functions, such as menu functions, are disallowed.
- Flow of control is slightly different.
- Application structures have a special use in accessing multiple record layouts for input files.

These differences are discussed separately below.

Disallowed Functions

The following types of functions are disallowed in a batch application structure:

- Menu functions
- Menu-related system functions, including POP, POPTOP, HELP, FORWARD, and BACKWARD
- Signon system functions, including SIGNON and SIGNOFF
- The ESCAPE system function

Specifying the Environment

The application compiler allows you to specify the environment in which an application can be executed, as follows:

- **Batch-only** applications can be executed only in the batch environment. The application compiler prevents you from defining disallowed functions.
- **Online-only** applications can be executed only in the online environment.

Application Flow of Control

Flow of control in applications defined using the application compiler is, for the most part, similar for both batch and online applications.

The following special considerations apply to CA ADS Batch applications.

Selection of Responses

Application responses are selected on the basis of batch control events and input record response field values. In the online environment, responses are selected on the basis of a control key pressed or a response field value entered by the terminal operator.

As in the online environment, batch application responses invoke application functions; when a response is selected, so is the function it invokes.

Immediately Executable Functions

Application functions are, by default, immediately executable. Using the application compiler, you can specify, whether a function is **immediately executable** or **deferred**. The runtime system uses the specification on mapin operations to determine the next dialog response process or application function to be executed. If, on a mapin operation, both a response process and a function are valid selections, transfer of control depends on the specification for the selected function:

- **Immediately executable**—Control passes to the selected function.

  Note: An exception is made when the selected function is the same as the current function. In such a case, the response process is executed.

- **Deferred**—Control passes to the selected response process. To pass control to the deferred function, the selected response process can issue an EXECUTE NEXT FUNCTION command.

In the batch environment, all functions are, by default, immediately executable. You can override the default for a function by using the new Response Definition screen of the application compiler. The specification is made for the response that invokes the function.

Differences Between Batch and Online

This difference between batch and online flow of control stems from the difference in transaction processing. In the online environment, even if the terminal operator requests transfer to another function, data on the current screen may still require processing by the current dialog before control is passed to the next function. Therefore, the current dialog's response process takes precedence.

In the batch environment, if the current record's response field value selects a different function, it is assumed that that function is required to process the current input record. Thus, by default, control passes immediately to that function.

Immediately executable functions enable applications to access input files that have multiple record layouts, as described later in this chapter.

Mapin Operations

Mapin operations are performed by the appropriate functions. For input files that have response fields, the runtime system first examines the response field. If the response field keeps control within the current function, the runtime system maps the record into variable storage and executes a response process. If, instead, the response field selects another function, the runtime system delays the mapin and passes control to the selected function. The next time a mapin operation is performed for that file, the runtime system immediately maps in the record.

Delayed Mapin

The major points regarding delayed mapin are as follows:

- Delaying mapin allows control to be passed to the dialog whose map handles the type of record being read.
- The delayed mapin can be performed only if the application is defined using the application compiler.
- The application functions should be defined as immediately executable, as they are by default.
- A dialog receiving control after a delayed mapin must perform a mapin operation to map the record into variable storage.

Delayed mapin enables applications to access input files that have multiple record layouts, as described below.

For more information about flow of control, see Runtime Flow of Control (see page 80).

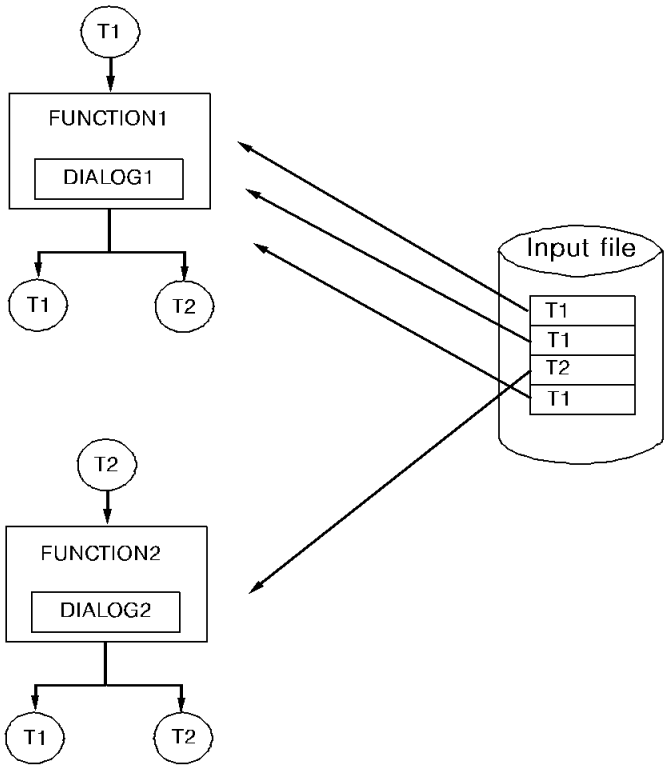
Accessing Input Files with Multiple Record Layouts

Application structures have a special use in CA ADS Batch in enabling applications to access input files with multiple record layouts.

For example, consider an application that reads a file containing two types of records stored together: TYPE1 records and TYPE2 records. Each type of record has its own record layout. Additionally, the first two bytes of each record identify the record type: T1 for TYPE1 records and T2 for TYPE2 records.

You define a map for each record layout and associate each map with a dialog (DIALOG1 and DIALOG2). At runtime, the runtime system must use the proper dialog for each input record. Since the sequence of record layouts on input is not predictable, the runtime system must know ahead of time which dialog to execute for each possible record layout. This is done by defining an application structure using the application compiler.

The diagram below shows the application structure and provides sample input data. Response T1 invokes function FUNCTION1, which executes dialog DIALOG1. Response T2 invokes function FUNCTION2, which executes dialog DIALOG2. Both responses are valid from both functions.



Application Execution

Application execution is described below:

1. FUNCTION1 executes DIALOG1. DIALOG1 performs a mapin operation.
2. The runtime system examines the response field of the first input record; it is a T1 record. Since T1 invokes the current function, the runtime system maps the record into variable storage and selects and executes a response process. The response process processes the record, then issues a READ TRANSACTION command, which terminates the current process and performs another mapin operation.
3. The runtime system examines the response field of the second input record; it, too, is a T1 record. The record is mapped in and the response process is executed. The response process issues a READ TRANSACTION command, which terminates the current process and performs another mapin operation.
4. The runtime system examines the third input record; it is a T2 record. Since T2 invokes FUNCTION2, control passes immediately to FUNCTION2, which executes DIALOG2. The record has not yet been mapped into variable storage. DIALOG2 performs a mapin operation.
5. The runtime system immediately maps the third record into variable storage (its response field was already examined) and selects and executes a response process. The response process issues a WRITE TRANSACTION command, which terminates the current process, maps a record to the dialog's output file, then performs a mapin operation.
6. The runtime system examines the response field of the fourth input record; it is a T1 record. The runtime system immediately invokes FUNCTION1, which executes DIALOG1. DIALOG1 performs a mapin operation.
7. The runtime system immediately maps the fourth record into variable storage, then selects and executes a response process. The response process issues a READ TRANSACTION command, which terminates the current process and performs a mapin operation.
8. The runtime system attempts to examine the response field of a fifth input record, but encounters an end-of-file condition.

   Two ways for the application to handle an end-of-file condition are described below:

   - The application can be defined so that the end-of-file condition selects a function whose dialog handles the condition.
   - A response process in DIALOG1 and DIALOG2 can be associated with the end-of-file condition; the response process would be selected and executed when the condition occurred.

Chapter 3: Process Command Language

This section contains the following topics:

Overview

Application Development System process commands are entered using the IDD DDDL compiler and are stored as process modules. A process module can be used in both online and batch dialogs. Some commands, such as DISPLAY, can be executed only in the online environment, while others, such as WRITE TRANSACTION, can be executed only in the batch environment.

A module can contain online-only and batch-only commands; however, at runtime, if a command in the wrong environment is to be executed, the application aborts. An exception is made for the Attributes and MODIFY MAP commands; if these commands are encountered in the batch environment at runtime, they are ignored.

Conditional Processing

Process modules can perform conditional processing based on the runtime environment, as shown below:

```
IF $ONLINE
    DISPLAY.
  ELSE
    WRITE TRANSACTION.
```

CA ADS Process Command Language

The following table summarizes the data fields, built-in functions, status conditions, and process commands that make up the Application Development System process command language, with notes on special usage and restrictions.

Several commands have MESSAGE parameters that send messages to the user's terminal in the online environment. In the batch environment, the messages are sent to the log file and/or the operator's console.

Category	Command or command variable	Comments
System-supplied data fields	$PAGE CURSOR-ROW CURSOR-COLUMN	Online only
	$ERROR-COUNT $INPUT-COUNT $OUTPUT-COUNT	Batch only
	All other fields	No restrictions
Built-in functions	Date functions	No restrictions
	All functions	No restrictions
Arithmetic and assignment commands	All commands	No restrictions
Conditional commands	All commands	No restrictions
	Cursor position condition	Online only
	Map field status condition	CHANGED and ERASED parameters are online only. In batch, the test applies to the dialog's input file map. Fields specified for a batch test refer to data fields in variable storage, not to fields in the input file record.
	Environment status conditions	No restrictions
	Map paging status conditions	Online only
	Batch control event status conditions	Batch only
	All other status conditions	No restrictions
Subroutine control commands	All commands	No restrictions

Category	Command or command variable	Comments
Control commands	CONTINUE EXECUTE NEXT FUNCTION INVOKE TRANSFER LEAVE APPLICATION NEXT TASK RETURN	No restrictions
	DISPLAY	Online only
	LEAVE ADS NEXT TASK	Online only
	LEAVE ADS CONDITION CODE	Batch only
	READ TRANSACTION	Batch only
	WRITE TRANSACTION	Batch only
	All other commands	No restrictions
Database commands	COMMIT	In an CA ADS Batch application, a COMMIT command issued when files are open results in no action, a warning message, or an abend, as specified at system generation and/or at runtime. The default action is abend.
	KEEP LONGTERM	Online only
	All other commands	No restrictions
Logical record commands	All commands	No restrictions
Map commands	Attributes	Online only Occurrence in the batch environment at runtime is ignored.
	CLOSE FILE MAPS	Batch only

Category	Command or command variable	Comments
	MODIFY MAP	The following parameters are online only: ■ CURSOR AT ■ WCC ■ BACKSCAN/NO- BACKSCAN ■ REQUIRED/ OPTIONAL ■ RIGHT/LEFT JUSTIFY ■ PAD ■ ATTRIBUTES In batch, the MODIFY MAP command applies to the input file map. Fields specified for a batch MODIFY MAP command refer to data fields in variable storage, not to fields in the input file record. Occurrence of an invalid parameter in the batch environment at run time is ignored.
Pageable map commands	All commands	Online only
Scratch management commands	All commands	Online only
Queue management commands	All commands	Allowed in batch only if the application is running central version
Utility commands	ACCEPT	RUN PARAMETERS parameter - batch only The following parameters are online only: ■ TASK CODE ■ TASK ID ■ LTERM ID ■ PTERM ID ■ SYSVERSION ■ SCREEN SIZE ACCEPT USER ID moves the user id specified in the USER (REQUESTOR) control statement into variable storage.
	WRITE TO LOG/OPERATOR	LOG parameter— no restrictions OPERATOR parameter— batch only
	SNAP	No restrictions

Category	Command or command variable	Comments
	WRITE PRINTER	Allowed in batch only if the application is running central version. SCREEN CONTENTS parameter— online only. CLASS or DESTINATION must be specified, unless a default is provided using the application compiler or at runtime in a control statement.

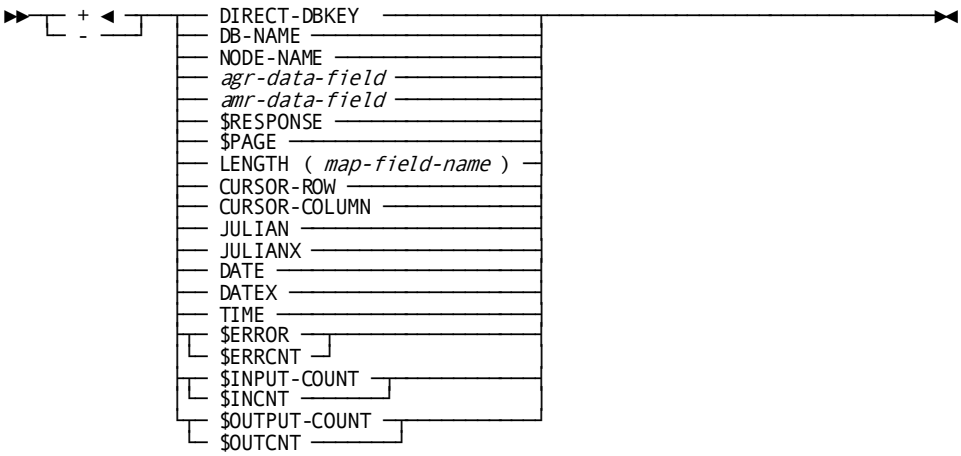
This chapter provides the following information about the CA ADS Batch specific process command language:

- Syntax rules for the following system-supplied data fields used with CA ADS Batch:
  - $ERROR-COUNT
  - $INPUT-COUNT
  - $OUTPUT-COUNT
- Syntax rules for the following status conditions:
  - Batch control event status conditions
  - Environment status conditions
  - Map field status condition
- Syntax rules for the following CA ADS Batch commands:
  - ACCEPT utility command
  - CLOSE
  - READ TRANSACTION
  - WRITE TRANSACTION

System-supplied Data Fields

System-supplied data fields are available that keep count of how many input, output, and suspense file records have been read or written by each dialog. A field is set to zero when the file it describes is opened; therefore, if an input file is opened, closed, and then opened again, the field is reset to zero when the file is opened a second time. Data in these fields can be moved to other data fields, but data cannot be moved into these fields.

Syntax for the system-supplied data fields, including the data fields used in the online environment, is shown below:



Syntax rules for the batch parameters follow:

$ERROR-COUNT

(Batch only) contains the number of input records that have been written to the suspense file for the current dialog. Data cannot be moved to $ERROR-COUNT.

Note: If a record is to be written to the suspense file, but a suspense file was not allocated for the dialog, nothing is written; however, $ERROR-COUNT is still incremented.

$INPUT-COUNT

(Batch only) contains the number of input records that have been read for the current dialog. Data cannot be moved to $INPUT-COUNT.

$OUTPUT-COUNT

(Batch only) contains the number of output records that have been written for the current dialog. Data cannot be moved to $OUTPUT-COUNT.

Status Conditions

The following status conditions support the batch environment:

- Batch control event status conditions
- Environment status conditions
- Map field status condition

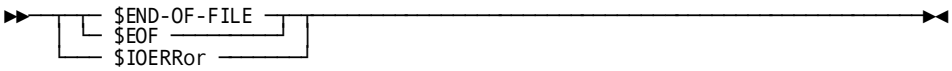
These status conditions are described separately below.

Batch Control Event Status Conditions

The batch control event status conditions are used to determine the occurrence of runtime events specific to batch input. Batch control event status conditions can be used only in the batch environment.

At the beginning of application execution, the batch control event status conditions are initialized and the outcome of each test is false.

Syntax for the batch control event status conditions is shown below:



$END-OF-FILE

Tests whether the most recent input file read operation resulted in an end-of-file condition.

$IOERROR

Tests whether the most recent input file read operation resulted in a physical input error. Note that a physical error on a write operation causes the application to abort.

Example

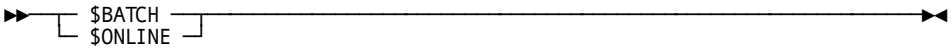
In the example below, execution of a set of commands continues until an end-of-file condition occurs:

```
WHILE NOT $EOF
  REPEAT.
    .
    .
    .
    WRITE TRANSACTION.
  END.
LEAVE APPLICATION.
```

Environment Status Conditions

Environment status conditions are used to determine the environment in which the application is being executed. The status conditions can be tested in both online and batch environments.

Syntax for the environment status conditions is shown below:



$BATCH

Is true when the application is executing in the batch environment.

$ONLINE

Is true when the application is executing in the online environment.

Example

In the following example, different types of processing are performed, depending on the runtime environment:

```
IF $ONLINE
  THEN
    DISPLAY.
  ELSE
    WRITE TRANSACTION.
```

Process Commands

CA ADS Batch provides the following process commands specific to the batch environment:

- CLOSE
- READ TRANSACTION
- WRITE TRANSACTION

These commands are described separately below.

CLOSE

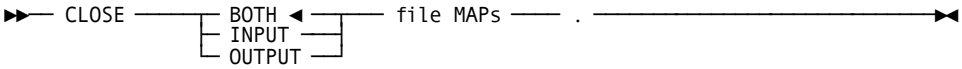
The CLOSE command is a batch-only command that closes a dialog's input and/or output file maps. If an application terminates with files still open, the runtime system automatically closes the files.

Note: A CLOSE command closes a file logically only if other dialogs using different maps have accessed the same file. To close the file physically, a CLOSE command must be issued for each map.

The CLOSE command is required when you want to close a file before the application terminates, as in the following cases:

- The application has been reading from or writing to a file and is required to start over at the beginning of the file.
- An output file to which records were written is to be read as an input file.
- A run-unit commit is to be performed by a COMMIT command or at the end of a run unit. If a COMMIT command is issued, but not all files used in the application are closed, the runtime system either takes no action, sends a warning message to the log, or abends the application, as specified at system generation and/or at runtime. The default action is abend.

Syntax for the CLOSE command is shown below:



CLOSE BOTH/INPUT/OUTPUT FILE MAPS

Specifies the file map to be closed, as follows:

- **BOTH** (default) specifies the dialog's input and output file maps. BOTH can be specified even if the dialog has only an input or an output file map.
- **INPUT** specifies the dialog's input file map.
- **OUTPUT** specifies the dialog's output file map.

READ TRANSACTION

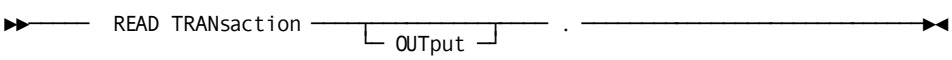
The READ TRANSACTION command is a batch only command that terminates the current process, performs a mapin operation, and then selects the next application function or response process to be executed.

Note: For applications defined using the application generator, the runtime system first examines the current record's response field. If the field selects an immediately executable function that is not the same as the current function, the runtime system passes control to the selected function. The next time a mapin operation is performed for the file, the runtime system immediately maps in the record.

For more information about the mapin process, see Batch Dialog Structure and Application Structure in CA ADS Batch Concepts (see page 15) and Runtime Flow of Control in Runtime Considerations (see page 79).

On a mapin operation, the runtime system automatically opens the file being read if the file is not already opened.

Syntax for the READ TRANSACTION command is shown below:



READ TRANSACTION

Terminates process execution and performs a mapin operation.

OUTPUT

Specifies that the file is to be opened as an input/output file. OUTPUT must be specified in the first READ TRANSACTION command for a VSAM entry-sequenced data set (ESDS) that is to be opened for both input and output. OUTPUT is ignored if the file is already opened; if the file is not a VSAM ESDS file, the application abends.

WRITE TRANSACTION

The WRITE TRANSACTION command is a batch only command that performs the following sequence of functions:

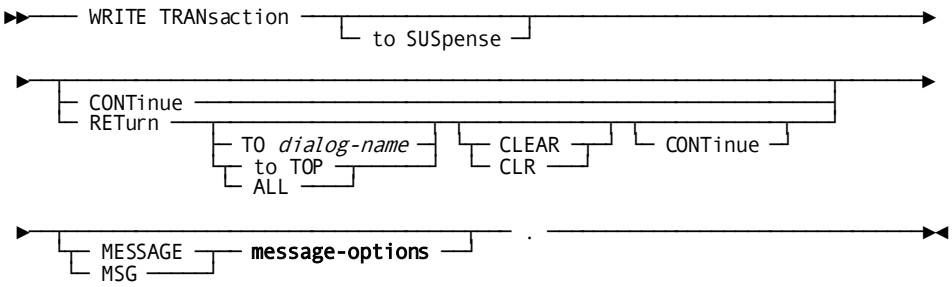
1. **Terminates the current process**.
2. **Performs a mapout operation**. If the dialog's current input file record (if any) contains no errors, and the keyword SUSPENSE is not included in the command, the mapout writes a record to the dialog's associated output file, according to the output file map definition. If the input file record contains errors or the keyword SUSPENSE is included in the command, the mapout writes the input record to the dialog's suspense file and sends applicable error messages to the log file.
3. **Passes control within the application**. Control can be passed to the dialog's premap process or mapin operation, or to a higher level dialog or application function.

If the write operation results in a physical output-error condition, the application terminates.

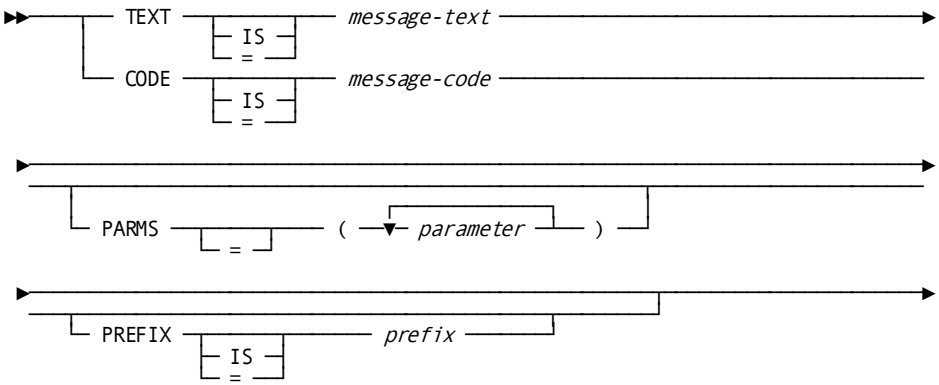
Note: A WRITE TRANSACTION command can be issued in a dialog that is not associated with an output file. In this case, the command is used only to write an input record to the suspense file. If the input record is not in error, nothing is written to the suspense file.

The WRITE TRANSACTION command also allows you to specify a message to be sent to the log file or to the operator's console. The destination of the message depends on the routing codes specified at system generation using CA-ADSOBSYS or at runtime in a control statement.

Syntax for the WRITE TRANSACTION command is shown below:



Expansion of Message-options



WRITE TRANSACTION

Terminates the current process and performs a mapout operation.

TO SUSPENSE

Specifies that the dialog's input record is to be written to the suspense file even if it does not contain errors. Nothing is written to the dialog's output file.

CONTINUE

Specifies that control is to be passed to the dialog's premap process.

RETURN TO *dialog-name*

Specifies that control is to be returned to a higher level dialog or application function.

Dialog-name specifies the 1- to 8-character name of a higher level dialog to which control is passed. *Dialog-name* is either the name of a variable data field that contains the dialog name or the dialog name itself enclosed in single quotation marks. The named dialog must not be higher than the top of a nested application structure in which the issuing dialog participates. If the named dialog is operative at more than one higher level, control passes to the lowest level dialog with the specified name.

RETURN TO TOP

Specifies the highest level to which control can pass. If the issuing dialog participates in a nested application structure, control returns to the top of the nested structure. If the issuing dialog does not participate in a nested structure, control returns to the mainline dialog at the top of the application thread.

If a RETURN statement does not specify a receiving dialog or TOP, control passes to the next higher level dialog or function. If the mainline dialog at the top of an application thread issues a RETURN command, the RETURN command is treated as a LEAVE APPLICATION command.

RETURN TO TOP CLEAR

Specifies that record buffers are reinitialized and currencies are released for the dialog receiving control. CLEAR is ignored if the receiving dialog is at the top of a nested application structure.

RETURN TO TOP CONTINUE

Specifies that control returns to the first command in the premap process of the dialog receiving control. If CONTINUE is not specified, control returns to the mapout operation of the dialog that receives control. CONTINUE is ignored if the receiving dialog is at the top of a nested application structure.

If neither CONTINUE nor RETURN is specified, control passes to the dialog's mapin operation. The runtime system maps the next record into variable storage, then selects the next application function or dialog response process to be executed.

Note: For applications defined using the application generator, the runtime system first examines the current record's response field. If the field selects an immediately executable function that is not the same as the current function, the runtime system passes control to the selected function. The next time a mapin operation is performed for the file, the runtime system immediately maps in the record.

MESSAGE TEXT IS *message-text*

Specifies the text of a message to be sent to the log file. *Message-text* is either the name of a variable data field that contains the message text or the text string itself enclosed in quotation marks. The text string can contain up to 240 displayable characters.

MESSAGE CODE IS *message-code*

Specifies the message dictionary code of a message to be sent to the log file and, if so directed by the destination specified in the dictionary for the message, to the operator. *Message-code* is either the name of a variable data field that contains the message code or the six-digit code itself expressed as a numeric literal.

PARMS=(*parameter*)

Specifies a replacement parameter for each variable field in the stored message identified by *message-code*. *Parameter* is either the name of an EBCDIC or unsigned zoned decimal variable data field that contains the parameter value, or the parameter value itself enclosed in single quotation marks. The parameter value must contain displayable characters. At runtime, each variable data field in a stored message expands or contracts to accommodate the size of its replacement parameter. A replacement parameter can be a maximum of 240 bytes.

Up to nine replacement parameters can be specified for a message; multiple parameters must be separated by blanks or commas. Multiple parameters must be specified in the order in which they occur in the stored message.

PREFIX IS *prefix*

Overrides the default prefix of a dialog and a map.

Prefix must either specify an EBCDIC or unsigned zoned decimal variable data field that contains a 2-character prefix or the 2-character prefix itself, enclosed in single quotation marks.

Chapter 4: Application Compiler

This section contains the following topics:

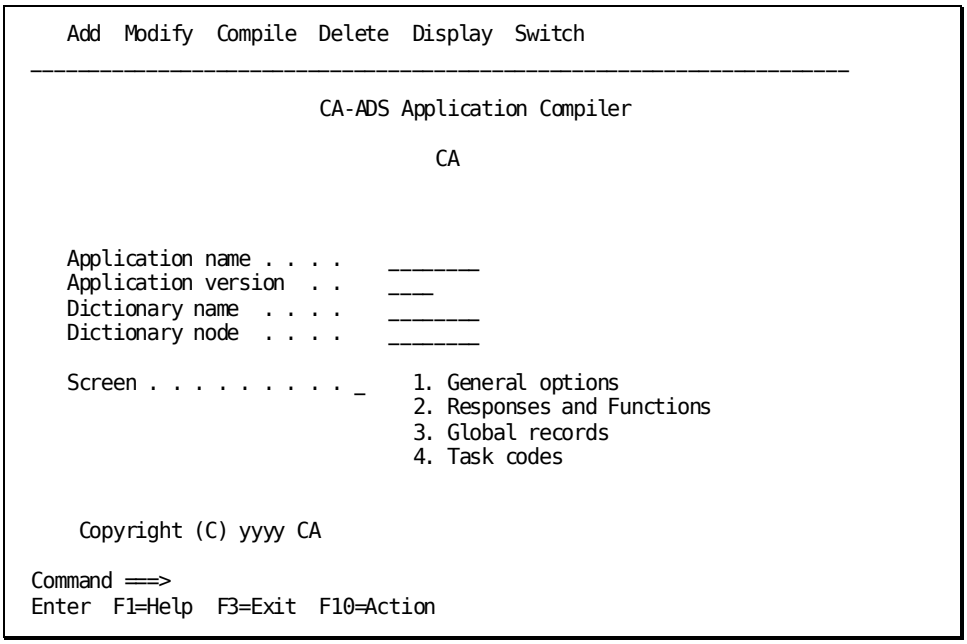
Overview

The application compiler enables you to predefine an application structure in terms of functions, responses, task codes, and global records. This chapter presents all of the application compiler screens, accompanied by notes on fields and functionality specific to CA ADS Batch.

Note: For more information about accessing these screens and descriptions of the fields, see the *CA ADS Reference Guide*.

Main Menu Screen

The Main Menu screen, shown below, is used to define basic information about the application.



General Options Screens

```
                          General Options              Page 1  of 2

Application name:  METAPPL1   Version:    1


    Description . . .  MIKE T'S APPLICATION

    Maximum responses . . . . . .  500

    Date format . . . . . .  1   1. mm/dd/yy  2. dd/mm/yy
                                 3. yy/mm/dd  4. yy/ddd

    Execution environment . . . . . .  2   1. Online    2. Batch

    Default execution mode. . . . . .  1   1. Step      2. Fast

    Default print destination . . . . . .

    Default print class . . . . . .  1



Enter  F1=Help  F3=Exit  F4=Prev  F5=Next  F8=Fwd
```

```
                          General Options              Page 2  of 2

Application name:  METAPPL1   Version:    1


   Security class. . . . . .  42

   Menus are . . . . . .  1   1. Not used   2. Security tailored
                              3. Untailored


   Signon is . . . . . .  1   1. Not used   2. Optional
                              3. Required

   Signon function is. . . .



Enter  F1=Help  F3=Exit  F4=Prev  F5=Next  F7=Bkwd
```

Batch Considerations

On the second General Options screen you can specify the environment in which the application can be executed, as follows:

- **BATCH** specifies that the application can be executed only in the batch environment.

  If BATCH is specified, the application compiler prevents you from entering online-only specifications, including:

  - Security specifications
  - Menu- and signon-related system functions in the **Function invoked** field on the Response/Function List screen. These functions include POP, POPTOP, HELP, FORWARD, BACKWARD, SIGNON, and SIGNOFF. Additionally, the ESCAPE system function cannot be specified.
  - Menu and menu/dialog functions on the Response/Function List screen and the Function Definition screen.

- **ONLINE** specifies that the application can be executed only in the online environment. The application compiler accepts all valid specifications on all screens.

Global Records Screen

The Global Records screen, shown below, enables you to specify records that are to be made available to all functions in an application.

Sample Screen

```
                                Global Records                Page  1 of  1

Application name:  TESTAPP1   Version:    1

                  Record name                  Version     Drop record (/)
  1.    ADSO-APPLICATION-GLOBAL-RECORD             1             _

  2.    ________________________________         ____          _

  3.    ________________________________         ____          _

  4.    ________________________________         ____          _

  5.    ________________________________         ____          _

  6.    ________________________________         ____          _

  7.    ________________________________         ____          _

  8.    ________________________________         ____          _


Enter  F1=Help  F3=Exit  F4=Prev  F5=Next  F7=Bkwd  F8=Fwd
```

Note: For more information about the Global Records screen, see the *CA ADS Reference Guide*.

Task Codes Screen

The Task Codes screen, shown below, enables you to specify task codes that initiate execution of the application.

Sample Screen

```
                                Task Codes                  Page  1 of  1

Application name:  TEST1      Version:    1

             Task Code           Function                 Drop (/)
        1.   DOIT                F1                          _

        2.   DOIT2               FUNC3                       _

        3.   UNKOLQ              IUADOLQF                    _

        4.   ________            ________                    _

        5.   ________            ________                    _

        6.   ________            ________                    _

        7.   ________            ________                    _

        8.   ________            ________                    _


Enter  F1=Help  F3=Exit  F4=Prev  F5=Next  F7=Bkwd  F8=Fwd
```

Note: For more information about the Task Codes screen, see the *CA ADS Reference Guide*.

Response/Function List Screen

Sample Screen

```
                          Response/Function List          Page   1 of   1

Application name:  METAPPL1   Version:    1

Select    Response    Assigned      Select    Function            Program/
 (/)      name        key            (/)      name/type(1,2,3)*   Dialog name

  _       R1           ENTER         _        F1       / 1        JPKSQLD1

  _       R3           PF02          _        FUNC3    / 1        STEVEDLG

  _       R2           PF02          _        FUNC2    / 3        ________

  _       R4           PF04          _        FUNC4    / 1        DIAL4

  _       R5           PF05          _        FUNC5    / 1        DIAL5

  _       LINKOLQR     PF06          _        LINKOLQF / 2        IDMSOLQS

                                      * Type: 1. Dialog  2. Program  3. Menu


Enter  F1=Help  F3=Exit  F4=Prev  F5=Next  F6=Search  F7=Bkwd  F8=Fwd
```

Batch Considerations

Special considerations apply to the following fields:

- **Response name**––Enables you to specify an input record response field value that initiates an application function.

  Note: If the response field for an input record is the concatenation of several fields, the response name you specify on the Response Definition screen must include any embedded blanks that would occur in a concatenation. For example, if a response field is the concatenation of two fields, the first being six bytes long, you would type the following response name to specify a response field value of ADD for the first field and E for the second field:

  ```
  ADD   E
  ```

- **Assigned key**—This field enables you to specify a batch control event. Valid batch control events are as follows:
  - **EOF**—The most recent input file read operation resulted in an end-of-file condition.
  - **IOERR**—The most recent input file read operation resulted in a physical input-error condition. Note that in CA ADS Batch, an output error causes the runtime system to terminate the application.
- **Function name**—The POP, POPTOP, HELP, FORWARD, BACKWARD, SIGNON, SIGNOFF, and ESCAPE system functions are disallowed entries for batch-only applications. In batch-only applications, you cannot specify a function type of **3** (Menu).

Note: For more information about the Response/Function List screen, see the *CA ADS Reference Guide*.

Response Definition Screen

The Response Definition screen, shown below, enables you to define application responses.

Sample Screen

```
                          Response Definition

Application name:   TEST1      Version:    1
Response name:      QUIT                                Drop response (/) _
Function invoked:   QUIT
Description . . . . _________________________

Response type. . . . . . . 2   1. Global      2. Local

Response execution . . . . 2   1. Immediate   2. Deferred

Assigned key . . . . . . . PF01
Control command. . . . . .  1   1. Transfer              2. Invoke
                                3. Link                  4. Return
                                5. Return continue       6. Return clear
                                7. Return continue clear  8. Transfer nofinish
                                9. Invoke nosave         10. Link nosave




Enter  F1=Help  F3=Exit  F4=Prev  F5=Next
```

Batch Considerations

- **Response execution** specifies whether the invoked function is immediately executable or deferred. In the batch environment, all functions are, by default, immediately executable. **Response execution** allows you to override these defaults.

  You override the default by entering **1** (immediately executable) or **2** (deferred) in the data field that immediately follows **Response execution**.

  For a discussion of immediately executable and deferred functions, see "Application Structure" in CA ADS Batch Concepts (see page 15).

- **Control command** displays the control command associated with the current application response.

  Note: For more information about control commands, see the *CA ADS Reference Guide*.

  You can select one of the following specifications by entering the appropriate number in the data field immediately following the prompt.

Function Definition Screen

The Function Definition screen, shown below, enables you to provide basic information about a function.

Sample Screen

```
                        Function Definition (Dialog)

Application name:  TEST1      Version:    1
Function name:     F2                                  Drop function (/) _
Description . . .  UNDEFINED

Associated dialog . . . . . D2          User exit dialog . . . . . ________
Default response  . . . . . ________

Valid                                    Valid
response(/)  Response Key   Function     response(/)  Response Key   Function

    _        ADD      PF02  F2               _        ________ _____ ________
    _        QUIT     PF01  QUIT             _        ________ _____ ________
    _        ________ _____ ________         _        ________ _____ ________
    _        ________ _____ ________         _        ________ _____ ________
    _        ________ _____ ________         _        ________ _____ ________
    _        ________ _____ ________         _        ________ _____ ________




Enter  F1=Help  F3=Exit  F4=Prev  F5=Next  F7=Bkwd  F8=Fwd
```

Note: For more information about the Function Definition screen, see the *CA ADS Reference Guide*.

Chapter 5: Dialog Compiler

This section contains the following topics:

Overview

The dialog compiler is used to define dialogs that perform transaction input and output, database retrieval and update, and any required processing within an application.

Associated Map

The environment within which a dialog can be executed depends on the map with which it is associated, as follows:

- A dialog associated with an **online map** can be executed only in the online environment.
- A dialog associated with a **file map** can be executed only in the batch environment.
- A **mapless dialog** can be executed in either environment.

At runtime, if the runtime system encounters a disallowed dialog, it abends the application.

Process Modules

Process modules associated with batch dialogs can also include online-only commands and command parameters, and vice versa. The dialog compiler, when compiling a process module, accepts both types of commands, regardless of the environment of the dialog. This allows a process module to be used in both the online and batch environments. If, however, the runtime system encounters a disallowed command or command parameter, the application abends. For more information about process commands in the batch and online environments, see CA ADS Batch Concepts (see page 15).

A dialog can be defined using the dialog compiler in either online or batch mode.

Note: Do not confuse batch and online *definition* modes with batch and online *execution* modes. The terms "batch" dialog and "online" dialog refer to the environment in which the dialog can be executed. Batch dialogs and online dialogs can be defined using the dialog compiler in online or batch mode, whichever is most convenient.

Batch Considerations

- You can associate a dialog with one input file map and one output file map.
- You can specify:
  - The z/OS ddnames (z/VSE filenames) of the dialog's input, output, and suspense files
- You can specify:
  - That the response process currently being defined is the dialog's default response process
  - That batch input record response field values and batch control events to associated with the response process

Dialog Compiler in Online Mode

The dialog compiler in online mode displays screens on which you specify information about the dialog you are defining. All screens are presented on the following pages, accompanied by notes on batch definition functionality.

Note: For more information about accessing screens and a description of each screen, see the *CA ADS Reference Guide*.

Main Menu Screen

The Main Menu screen, shown below, enables you to specify basic information about a dialog.

Sample Screen

```
   Add  Modify  Compile  Delete  Display  Switch
.________________________________________________________________________.

                            CA-ADS Online Dialog Compiler

                                      CA

   Dialog name . . . . . .   ________
   Dialog version  . . . .   ____
   Dictionary name . . . .   ________
   Dictionary node . . . .   ________

   Screen  . . . . . .  _   1. General options
                            2. Assign maps
                            3. Assign database
                            4. Assign records and tables
                            5. Assign process modules


   Copyright (C) 1972, 1996 CA

Command ===>
Enter  F1=Help  F3=Exit  F10=Action
```

Map Specifications Screen

The Map Specifications screen, shown below, enables you to specify information about the map associated with a dialog.

Sample Screen

```
                          Map Specifications

                      Dialog  JPKTD10   Version     1


                                        |
Map name  . . . .   ________            |   Input map . . . . .   ________
Version . . . . .   ____                |   Version . . . . . .   ____
                                        |   Label . . . . . . .   ________
Paging options     _ 1. Wait            |
                     2. No Wait         |   Output map  . . . .   ________
                     3. Return          |   Version . . . . . .   ____
                                        |   Label . . . . . . .   ________
Paging mode . . . _ Update              |
                  _ Backpage            |   Suspense file label   ________
                  _ Auto display        |
                                        |




 Enter  F1=Help  F3=Exit  F4=Prev  F5=Next  F6=Switch Protection
```

Use the F6 key to turn protection for the left and right side of the screen on and off. For example, press F6 if you want to enter information on the right side of the screen; press it again to protect the information that is displayed.

Batch Considerations

The **Map name** field on the screen allows you to associate map with the dialog and, by doing this, allows you to specify the environment in which the dialog can be executed, as follows:

- **Input map** specifies the name of an input file map and indicates that the dialog can be executed only in the batch environment.
- **Output map** specifies the name of an output file map and indicates that the dialog can be executed only in the batch environment.
- **Version** specifies the 1- to 4-digit version number of the corresponding map. If no version number is specified, **Version** defaults to 1.
- **Label** allows you to specify the z/OS ddname (z/VSE filename) of a batch dialog's input file map or output file map.

  Specifications made in these fields can be overridden at runtime.

- **Suspense file label** allows you to specify the z/OS ddname (z/VSE filename) of a batch dialog's suspense file.

  Specifications made in this field can be overridden at runtime.

  Runtime labels for an input map and suspense file can be specified only if the dialog is associated with an input file map. A runtime label for an output map can be specified only if the dialog is associated with an output file map.

  Note: When you supply a runtime label for a suspense file, either during dialog definition or at runtime, you implicitly specify that a suspense file is required for the dialog.

A dialog associated with an online map cannot be associated with an input or output file map, and vice versa. A dialog can be associated with both an input and an output file map. A dialog not associated with a map is called a mapless dialog and can be executed in both the batch and online environments.

Database Specifications

The Database Specifications map is shown below.

Sample Screen

```
                        Database Specifications

                     Dialog  NAME1     Version     1



  Subschema . . . . . . . . . . .   ________
  Schema  . . . . . . . . . . . .   ________
  Version . . . . . . . . . . . .   ____

  Access Module . . . . . . . . .   NAME1

  SQL Compliance  . . . . . . . .   _ 1. ANSI-standard SQL
                                      2. FIPS

  Date Default Format . . . . . .   _ 1. ISO   2. USA   3. EUR   4. JIS
  Time Default Format . . . . . .   _ 1. ISO   2. USA   3. EUR   4. JIS




 Enter  F1=Help  F3=Exit  F4=Prev  F5=Next
```

Options and Directives Screen

The Options and Directives screen, shown below, enables you to specify special options for a dialog.

Sample Screen

```
                          Options and Directives

                      Dialog  JPKTD10   Version     1


   Message prefix  . . . . . . . . . . . DC
   Autostatus record . . . . . . . . . . ADSO-STAT-DEF-REC
   Version . . . . . . . . . . . . . . .    1
   Description . . . . . . . . . . . . . ADS DIALOG

   Options and directives  . . . . . .   _ Mainline dialog
                                         _ Symbol table is enabled
                                         / Diagnostic table is enabled
                                         / Entry point is premap
                                         _ COBOL moves are enabled
                                         / Activity logging
                                         / Retrieval locks are kept
                                         / Autostatus is enabled




  Enter  F1=Help  F3=Exit  F4=Prev  F5=Next
```

Batch Considerations

Regardless of the entry point specification, a dialog without a batch input file map begins with its premap process. A dialog without a premap process begins with its first mapping operation.

Records and Tables Screen

The Records and Tables screen, shown below, enables you to associate work records with the dialog and to assign the new copy attribute to records known to the dialog.

Sample Screen

```
                              Records and Tables          Page    1 of 1

                         Dialog  JPKTD1    Version     1

   Name                                 Version    Work    New copy   Drop
1. WKB-INTERFACE-RECORD                     1       /         _        _

2. OTHER-REC                                1       /         _        _

3. ADSC-SCRATCH                             1       _         /        _

4. ADSC-PROD                                1       /         _        _

5. ADSC-DB                                  2       /         _        _

6. ADSC-TR                                  1       /         _        _

7. ADSC-PROTO-SYNTAX                        1       /         _        _



 Enter  F1=Help  F3=Exit  F4=Prev  F5=Next  F7=Bkwd  F8=Fwd
```

Note: For more information about the Records and Tables screen, see the *CA ADS Reference Guide*.

Process Modules Screen

The Process Modules screen, shown below, enables you to associate a process (premap, response, or declaration) with the dialog.

Sample Screen

```
                              Process Modules                Page    1 of    1

                         Dialog  NAME1     Version     1
  Name    ________________________________                  _  Type
  Version ____                                              _  Execute on errors
  Key     _____   Value  ________________________________   _  Drop

  Name    ________________________________                  _  Type
  Version ____                                              _  Execute on errors
  Key     _____   Value  ________________________________   _  Drop

  Name    ________________________________                  _  Type
  Version ____                                              _  Execute on errors
  Key     _____   Value  ________________________________   _  Drop

  Name    ________________________________                  _  Type
  Version ____                                              _  Execute on errors
  Key     _____   Value  ________________________________   _  Drop

* Type : 1=Declaration  2=Premap  3=Response  4=Default Response
DC498166 Neither a map nor premap are defined


 Enter  F1=Help  F3=Exit  F4=Prev  F5=Next  F7=Bkwd  F8=Fwd
```

Note: For batch dialogs, if the **Value** for an input record is the concatenation of several fields, the response value you specify on the Process Modules screen must include any embedded blanks that would occur in a concatenation. For example, if a response value is the concatenation of two fields, with the first being six bytes long, you would type the following value to specify a response field value of ADD for the first field and E for the second field:

```
ADD    E
```

Batch Considerations

Key specifies the batch control event that initiates the response process at runtime. Valid batch control events are listed below:

- **EOF**—The most recent input file read operation resulted in an end-of-file condition.
- **IOERR**—The most recent input file read operation resulted in a physical input-error condition. Note that in CA ADS Batch, output errors cause the runtime system to terminate the application.

Note: For more information about the Process Modules screen, see the *CA ADS Reference Guide*.

Dialog Compiler in Batch Mode (ADSOBCOM)

ADSOBCOM enables you to compile a dialog in batch mode.

Note: For more information about complete syntax and syntax rules for ADSOBCOM, see the *CA ADS Reference Guide*.

Chapter 6: CA IDMS/DC Mapping Facility

This section contains the following topics:

Overview

The CA IDMS/DC mapping facility enables you to define online maps and file maps. Maps can be compiled in both online mode and batch mode.

Note: A file map is not in itself an input or output file map. Its use within a dialog, as specified during dialog definition, determines whether it is an input file map, an output file map, or both.

The online and batch modes of the mapping facility are described separately below, following discussions of batch automatic editing and error handling and variable array processing.

Batch Automatic Editing and Error Handling

The automatic editing and error handling facility operates in the batch environment in much the same manner as it does in the online environment. On a mapin operation, the facility edits and performs error checking on data being transferred from an input file to variable storage. On a mapout operation, the facility edits data being transferred from variable storage to an output file.

The automatic editing and error handling facility in batch mode differs in the following ways.

Error Handling

In the online environment, the terminal operator's screen is redisplayed with error messages. In the batch environment, the error record is written to a suspense file and error messages are written to the log file.

Automatic Editing and Error Handling Options

In the batch environment, several automatic editing and error handling options are not applicable. Options you can specify for file map fields include:

- An external picture
- An edit table
- A code table
- The data transfer option on input
- The data transfer option on output
- A user-written edit module on input
- A user-written edit module on output
- An error message text or code

You can specify these options in MAPB on the User-defined Edit Modules screen, the Additional Edit Criteria screen, and the Map Read/Write options screen. With the batch mapping compiler, you use the MFLD statement.

Implied Decimal Point and Sign

The external picture of a file map field can include an implied decimal point (represented by the symbol V) and an implied sign (represented by the symbol S). This differs from map fields in the online environment, in which external pictures, which represent how fields are to be displayed on a screen, cannot include implied decimal points and signs.

Data Types

In the batch environment, the data type of an external field can be display, zoned decimal, packed decimal (COMPUTATIONAL-3) or binary (COMPUTATIONAL). In the online environment, the data type must be display.

Null Values

In the online environment, if the terminal operator enters nothing in a field, nothing is mapped into the associated field in variable storage. In the batch environment, unless the input data transfer option is set to NO, all input record fields included in the map definition are mapped into variable storage.

Variable Array Processing

Variable array records are records that contain a field that occurs a certain number of times depending on the value of another field in the record. An example is shown below:

```
SAMPLE-RECORD
03 FIELDA PIC S9(4) USAGE IS COMPUTATIONAL.
03 FIELDB PIC X(20) OCCURS 3 TO 10 TIMES
                    DEPENDING ON FIELDA.
```

Processing

When the external record of a file map includes a variable array, CA ADS Batch performs the following processing:

- **On a mapin operation**, the runtime system maps all fields included in the map definition into variable storage, even array fields that do not actually exist for the current input record. For example, in SAMPLE-RECORD, if FIELDB occurs seven times, the eighth, ninth, and tenth occurrences of FIELDB do not exist; however, the runtime system maps in whatever is in the record buffer for these occurrences. The record's control field (that is, FIELDA in the example), should be included in the map definition; otherwise, the field will not be mapped in and the application will not be able to distinguish the real data from the meaningless data in the array.
- **On a mapout operation**, the runtime system always maps out all fields that are included in the map definition, regardless of the value of the control field. The maximum record length for the map's associated external record is always written.

  Note: If a field contains meaningless data that cannot be mapped out successfully (that is, fails the automatic editing stage), the application aborts.

Online Mapping Facility

The online mapping facility (MAPB) enables you to compile maps in online mode. You use the following screens to define a map:

1. **Main Definition screen**––Used to specify basic information about the map, including the map name and the dictionary in which it resides.

   Note: The Main Definition screen records are called **internal records** and refer to records in variable storage. A map's **external record** is the record that describes a layout of data in the input or output file. Record elements in an external record map to data fields in internal records. If an external record element maps to itself, you do not specify the external record on the Map Definition screen.

2. **Field Definition screen**––Used to specify the external record. If the external record participates in several files, you also specify the applicable file.

   Note: Once an external record has been associated with the map, you must explicitly select it to display it.

3. **File Field Selection screen**––Used to select record elements in the external record for participation in the map and for editing.

4. **File Field Edit screen**––Used to associate an external record field with an internal record field and to specify special editing and error handling characteristics.

5. **Map Definition screen**––Used to compile the file map load module.

The MAPB screens are shown below, accompanied by notes on new and changed fields and functionality.

Map Definition Screen

The Map Definition screen, shown below, enables you to provide basic information about the map.

Sample Screen

```
                                    CA                                  volser
    CA IDMS/DC FILE MAPPING REL nn.n          ***  MAP DEFINITION  ***    STEP


ACTION:     MAPNAME: JMAOUT   VER:    1   DICTNAME:          DICTNODE:
EDIT(Y/N): Y
                              USING RECORDS
        RECORD NAME              VER           ROLE NAME                 DEL
:
:
:
:
_____________________________________________________________________________


COPY FROM MAPNAME:          VER:
          COPY ACTION (ALL/FMT):



_____________________________________________________________________________
SELECT NEXT FUNCTION:      _ MAP DEFINITION
_ EXTENDED MAP DEFINITION  _ FIELD SELECTION          _ ADDITIONAL RECORDS
_ FILE FIELD SELECTION     _ FILE FIELD EDIT
                           _ QUIT                     _ SUSPEND
```

Batch Considerations

The **Copy Format** option on the **Add** activity on the action bar on the Map Definition screen has no meaning for file maps. If you specify **Format** for file maps, MAPB changes the action to **All**.

Extended Map Definition Screen

The Extended Map Definition screen, shown below, enables you to provide further basic information about the map, including the external file and record to be associated with a file map.

```
      CA IDMS/DC FILE MAPPING REL nn.n     *** EXTENDED MAP DEFINITION ***  STEP




_____________________________________________________________________________

      FOR EXTERNAL FILE ACCESS
          RECORD NAME: JMAOUT                          VER:    1
            FILE NAME: JMAOUT                          VER:    1

_____________________________________________________________________________
MSG PREFIX: DC        DECIMAL POINT IS COMMA (Y/N): N
_____________________________________________________________________________
  WRITE CONTROL CHARACTER:   UNLOCK KEYBOARD.....: Y     RESET MDT: Y
                             START PRINT.........: N     ALARM....: N
                             NEWLINE(NL/40/64/80):
_____________________________________________________________________________



SELECT NEXT FUNCTION:      _ MAP DEFINITION
_ EXTENDED MAP DEFINITION  _ FIELD SELECTION          _ ADDITIONAL RECORDS
_ FILE FIELD SELECTION     _ FILE FIELD EDIT
                           _ QUIT                     _ SUSPEND
```

Batch Considerations

Batch fields on the screen are described below:

- **Record name** specifies the name of an external record to be associated with a file map. The record must be described in the data dictionary and included within a file.
- **File name** specifies the name of the external file in which the named record participates. If no file name is provided, MAPB uses the first file in the dictionary's file-record set for the named record.

  Note that the record's file description in the data dictionary need not include file characteristics; these can be specified instead at runtime. Nevertheless, MAPB displays a warning message, such as the following:

  ```
  FILE TYPE NOT DEFINED TO IDD
  DETERMINED AT EXECUTION
  ```

- **Version** specifies the 1- to 4-digit version number of the corresponding record or file name. If **Version** is not specified, it defaults to the data dictionary default version number, as defined by the DDDL SET OPTIONS statement.

File Field Selection Screen

The File Field Selection screen, shown below, enables you to select external record elements for inclusion in the map definition and for editing on the File Field Edit screen. Fields not included in the map definition are not mapped in or mapped out at runtime.

The File Field Selection screen lists all of the fields in the external record. If the record contains more fields than fit on the screen, MAPB builds additional pages to accommodate all the fields. You can display other pages of the screen by specifying a page or by pressing ENTER to display the next page.

You select fields on the File Field Selection screen by keying a nonblank, non-underscore character (except the D character) at the underscore below the SELECT prompt that corresponds to the desired field. You cannot select both a group field and any of its subordinate-level fields.

By selecting an external field for inclusion in the map, you also select it for editing on the File Field Edit screen, displayed later in the map definition. Once a field has been selected and edited, you can again use the File Field Selection screen to reselect the field for editing or to deselect the field from the map definition.

```
     CA IDMS/DC FILE MAPPING REL nn.n        *** FILE FIELD SELECTION *** STEP


MAPNAME: JMAOUT                                              PAGE:    1 OF:    1
VER....:    1                                                NEXT PAGE....:

    SELECT      LEVEL     FIELD NAME                           OCCURRENCE
      X          02       LAST-NAME
      X          02       FIRST-NAME
      X          02       ID
      _          02       FILLER









SELECT NEXT FUNCTION:            _ MAP DEFINITION
_ EXTENDED MAP DEFINITION        _ FIELD SELECTION      _ ADDITIONAL RECORDS
_ FILE FIELD SELECTION           _ FILE FIELD EDIT
                                 _ QUIT                 _ SUSPEND
```

Batch Considerations

Specific fields on the screen are described below:

- **SELECT** specifies that the corresponding record field is to be included in the map definition. To select a field, key a nonblank, non-underscore character (except the D character) at the underscore below the SELECT prompt that corresponds to the desired field. You cannot select both a group field and any of its subordinate-level fields.

  Selecting a field also selects it for editing on the File Field Edit screen. To reselect a field after it has been edited, key another nonblank, non-underscore character (except for D) over the X.

  To deselect a selected field, key a D, underscore, or blank over the X.

File Field Edit Screen

The File Field Edit screen, shown below, enables you to associate a selected external record field with an internal record field and to specify special editing and error handling characteristics. The File Field Edit screen is displayed once for every external field selected for editing on the File Field Selection screen.

Note: An error message that you specify for a map field is written to the log file at runtime if the field is in error and the input record is being written to a suspense file.

```
             CA IDMS/DC FILE MAPPING REL nn.n          *** FILE FIELD EDIT ***


MAP: JMAOUT       VER:    1                                              STEP

EXTERNAL FIELD
  FIELD   : LAST-NAME                       OCCURS:
  OF REC  : JMAOUT                          VER:    1
  FILE-PIC: X(15)

INTERNAL FIELD
  DFLD    : LAST-NAME                       OCCURS:
  OF REC  : JMAOUT                          VER:    1

EDIT TABLE:           VER:       _  (LINK Y/N) _  (VALID Y/N/D-DICT)
CODE TABLE:           VER:       _  (LINK Y/N) Y  (EDIT Y/N)

FOR INPUT   Y  (DATA Y/N)
            <EDIT MODULE:           _  (WITH EDIT B/A/N)>

FOR OUTPUT  Y  (DATA Y/N/E)
            <EDIT MODULE:           _  (WITH EDIT B/A/N)>
N  (ERROR MSG T-TEXT/I-ID/N-NULL)   PREFIX:       ID:                   TEXT:
```

Batch Considerations

Specific fields on the screen are described below:

- **EXTERNAL FIELD** identifies the external record field being edited and specifies its external picture, as described below:
  - **FIELD** specifies the field name. This field is protected.
  - **OCCURS** specifies the occurrence of the field if the field is multiply-occurring. This field is protected. Levels of occurrences are separated by commas.
  - **OF REC** specifies the external record to which the field belongs. This field is protected.
  - **VER** specifies the version number of the external record. This field is protected.
  - **FILE-PIC** specifies the external picture of the file field. You can modify the picture for use by automatic editing. If the external field is to map to itself, you cannot specify a picture that changes the field's length.

    Note: For more information about specifying external pictures, see the *CA IDMS Mapping Facility Guide*.
- **INTERNAL FIELD** identifies the internal record field with which the external record field is to be associated, as described below:
  - **DFLD** specifies the name of an internal record field.

    Note: If no internal field is specified, the external field maps to itself, from the record buffer into variable storage, as described under "Input and Output Files and File Maps" in CA ADS Batch Concepts (see page 15).

    Alternatively, you can enter $RESPONSE (or $R) to specify that the associated external field is a response field. Several external fields can be designated as response fields. In this case, the response field is composed of the concatenation of all the fields designated as response fields. The maximum length of a concatenated response field is 32 bytes. For more information about response fields, see "Batch Dialog Structure" in CA ADS Batch Concepts (see page 15).
  - **OCCURS** specifies the occurrence of the field, if any. Up to three levels can be specified. Levels must be separated by commas.

    For a response field composed of the concatenation of several external fields, OCCURS is used to specify the sequence of the external field in the concatenation. By default, external fields are concatenated in the order in which they were associated with $RESPONSE.
  - **OF REC** specifies the record to which the external field belongs. OF REC is required only if DFLD is not unique among the fields in the records associated with the map.
  - **VER** specifies the version number of the record. This field is required only if more than one version of the internal record is associated with the map.

Batch Map Compiler (RHDCMAP1)

RHDCMAP1 enables you to compile maps in batch mode. To define a file map in batch mode, you use automatic panel definition statements (MAP AUTOPANEL and MFLD).

Note: You cannot define a file map using the manual panel definition statements.

Syntax for the MAP AUTOPANEL and MFLD statements can be found in the *CA IDMS Mapping Facility Guide*.

Chapter 7: Runtime Considerations

This section contains the following topics:

Overview

CA ADS Batch applications are executed by running ADSBATCH, the batch version of the Application Development System runtime system. This chapter discusses the following runtime topics:

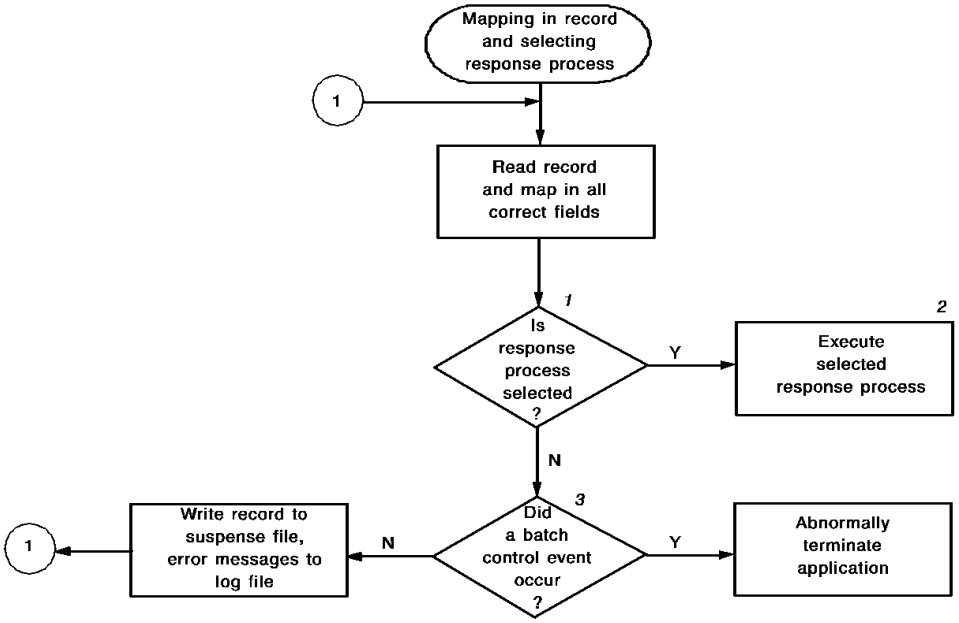
- Flow of control in an application
- Specifying characteristics of files accessed in an application
- The relationship between application components and physical data sets
- Application restartability
- CA ADS Batch trace facility
- Multiple external run units under CA ADS Batch
- Operator shutdown of an application
- Runtime control parameters, including PARM field parameters and control statements
- Loading runtime components
- JCL for ADSBATCH

Runtime Flow of Control

Flow of control at runtime determines which dialog in an application is executed and, within a dialog, which premap or response process is executed and whether a mapin or mapout operation is performed. Most batch flow of control considerations are similar to those for online applications.

An overview of batch runtime flow of control is provided under "Batch Dialog Structure" and "Application Structure" in CA ADS Batch Concepts (see page 15). A detailed description of the flow of control on a mapin operation is provided below.

The diagram below shows the flow of control on a mapin operation for an application not defined using the application compiler.



Notes:

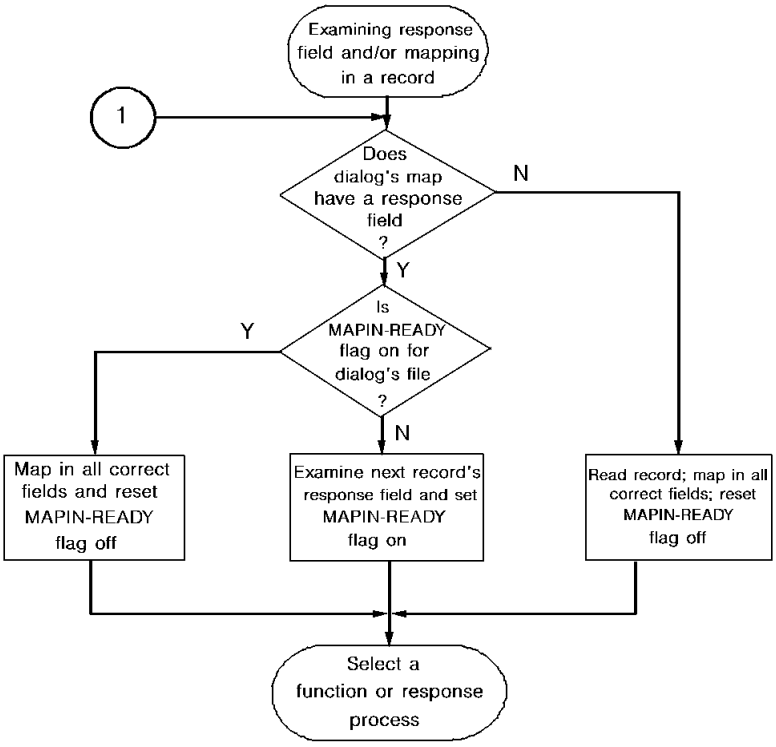
1 A response process is selected by the occurrence of a batch control event (BCE) or response field value (RFV). If the BCE or RFV do not invoke a response process and if a physical input error did not occur, the dialog's default response process, if any, is selected.

2 The response process is not executed if the record contains input errors and EXECUTE ON EDIT ERROS is NO for the response process. Instead, the record is written to the suspense file, error messages are sent to the log file, and another record is read and mapped in.

3 The application aborts if the same dialog encounters an end-of-line condition for a second time.

The following two diagrams show the runtime flow of control on a mapin operation for an application that is defined using the application compiler, as follows:

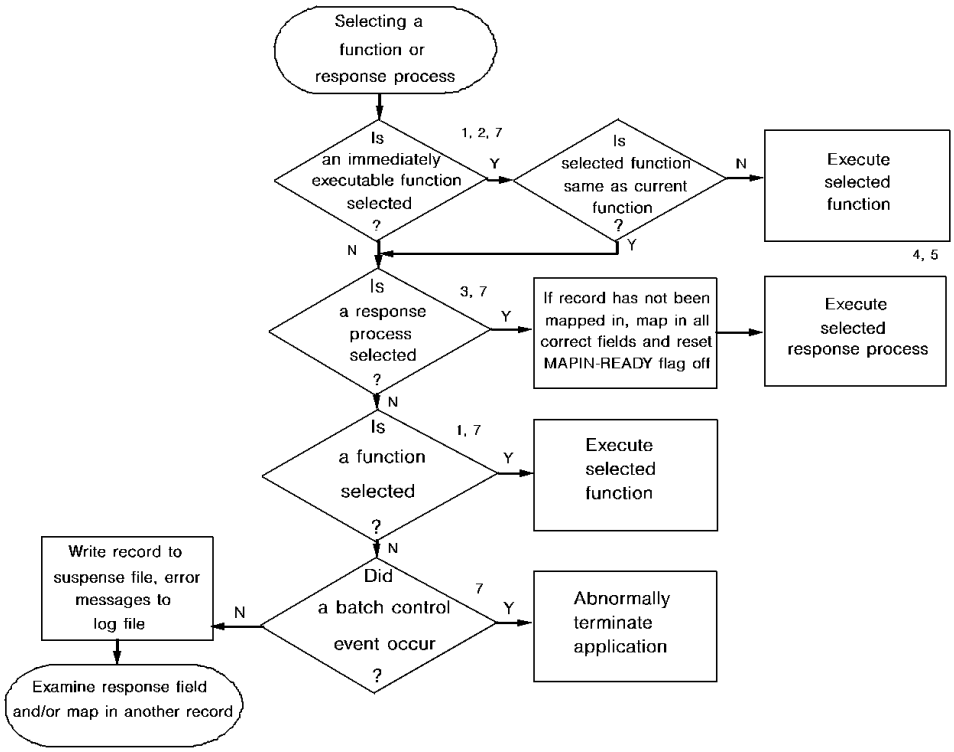
- The first diagram below shows the examination of the input record's response field and the mapin of the record. At runtime, when a mapin operation is to be performed, the runtime system first examines the record's response field, if any. If the response field value invokes an immediately executable function that is different from the current function, control passes to that function and the record is not mapped in. The next time a mapin operation is performed for that file, the record is mapped in immediately.



The runtime system maintains a MAPIN-READY flag for each input file in the application. The flag is set ON when the response field for a record has been examined, but the record has not yet been mapped into variable storage. When the flag is OFF, a mapin operation begins with an examination of the record's response field. When the flag is ON, a mapin operation begins with the record being mapped into variable storage.

- The following diagram shows the selection of an application function or dialog response process. The order of precedence in function or response process selection is as follows:

  1. A valid immediately executable function that is not the same as the current function.
  2. A valid dialog response process. If a response process is selected and the current record has not yet been mapped in, it is mapped in and the MAPIN-READY flag is reset OFF.
  3. A valid function.



Notes:

1 A function is selected by the occurrence of a batch control event (BCE) or a response field value (RFV). If the BCE or RFV is associated with an application response that is valid for the current function, then the function invoked by the response is selected. The function invoked by the current function's default response is selected if no valid function is selected, if a physical input error did not occur, and if the RFV is spaces (or the map has no response field or an end-of-file condition has occurred).

2 By default, all functions in the batch environment are immediately executable.

3 A response process is selected by the occurrence of a BCE or RFV. If the BCE and RFV do not invoke a response process, and if a physical input error did not occur, the dialog's default response process, if any, is selected.

4 The response process is not executed if the record contains input errors and EXECUTE ON EDIT ERRORS is NO for the response process. Instead, the record is written to the suspense file, error messages sent to the log file, and another record is examined and/or mapped in.

5 If a deferred function was also selected, it can be executed by an EXECUTE NEXT FUNCTION command.

6 In this case, the selected function may be the same as the current function. If so, the associated dialog is reexecuted, beginning with its premap process or mapin operation, as applicable.

7 The application abends if the same dialog encounters an end-of-file condition for a second time.

Runtime Flow of Control

The runtime flow of control is as follows:

1. A dialog performs a mapin operation that examines the response field value of the next input record. The record's response field value invokes an immediately executable function that is not the same as the current function. Control passes to that function, which executes its associated dialog. The record has not yet been mapped into variable storage.
2. The second dialog performs a mapin operation on the same file. The record is immediately mapped into variable storage.
3. The response field value for the record mapped in is associated with an immediately executable function; this time, however, the function is the same as the current function, so the function is not reexecuted.
4. The runtime system selects and executes a response process. (If no response process can be selected, the current function is re-executed from the beginning of the dialog).

Specifying File Characteristics

You specify file characteristics for each file that the application accesses. File characteristics include record format, block size, logical record length, file type, and device type.

The following topics are discussed separately below:

- Where to specify file characteristics
- Specifying a record length
- Specifying a block size
- Default file characteristics
- Specifying characteristics of concatenated data sets
- Specifying characteristics of suspense files
- Tape labels

Where to Specify File Characteristics

z/OS

Under z/OS, you can specify a file's characteristics at any of the following levels:

- **In the file's associated IDD file entity**—Any characteristics you specify here do not have to be specified in the JCL or data set label. However, characteristics specified in IDD are enforced at the other levels; if a characteristic is specified both in IDD and at another level, the IDD specifications take precedence.

  Note: Do not specify a file type or device type at the IDD level. If you specify a characteristic in IDD and you want to change it later, you must recompile all maps that use the file. Therefore, you may not want to provide any characteristics for a file at the IDD level.
- **In the JCL**—Characteristics you specify in the JCL are enforced at the data set label level.
- **As part of the data set's label**.

z/VSE

Under z/VSE, you can specify a file's characteristics at either of the following levels:

- **In the file's associated IDD file entity**—Any characteristics you specify here do not have to be specified in IDMSFILE control statements. However, characteristics specified in IDD are enforced at the IDMSFILE level; if a characteristic is specified both in IDD and at the IDMSFILE level, the IDD specifications take precedence.

  Note: Do not specify a file type or device type at the IDD level. If you specify a characteristic in IDD and you want to change it later, you must recompile all maps that use the file. Therefore, you may not want to provide any characteristics for a file at the IDD level.
- **In IDMSFILE control statements**—IDMSFILE is a program that enables you to specify file characteristics at runtime in the form of control statements. For a detailed description of IDMSFILE, see z/VSE File Characteristics Program (see page 123).

Defaults

CA ADS Batch applies defaults to characteristics that are not specified at any level. For physical sequential files, the record format defaults to undefined, the logical record length to 0, and the block size to 6233 for disk files and 32760 for tape files. For VSAM ESDS files, CA ADS Batch uses the information supplied from the catalog.

Specifying a Record Length

The record length you specify for a file must be consistent with the record lengths of the IDD record entities that are associated with the file and that are used in the application, as follows.

Fixed Record Format

If the record format is fixed, the record length of the file must be equal to the record length of each associated record entity.

Variable Record Format

If the record format is variable, the record length of the file must be greater than or equal to the record length of the longest associated record entity plus four.

Undefined Record Format

If the record format is undefined, the record length of the file must be greater than or equal to the record length of the longest associated record entity.

The RECORD SIZE specification for a file can be removed by using IDD to set it to 0 (zero).

Specifying a Block Size

The block size you specify for a file must be consistent with the logical record length of the file, as follows.

Fixed Blocked Record Format

If the record format is fixed blocked, the block size must be an integral multiple of the logical record length.

Fixed Unblocked Record Format

If the record format is fixed unblocked, the block size must equal the logical record length.

Variable Blocked Record Format

If the record format is variable blocked, the block size must be greater than or equal to the logical record length plus four.

Variable Unblocked Record Format

If the record format is variable unblocked, the block size must equal the logical record length plus four.

Undefined Record Format

If the record format is undefined, the block size must be greater than or equal to the logical record length.

The RECORD SIZE specification for a file can be removed by using IDD to set it to 0 (zero).

Default File Characteristics

If you do not specify a file's record format, logical record length, and/or block size, CA ADS Batch supplies defaults for the omitted characteristics.

If the **record format** is not specified, CA ADS Batch supplies one of the following characteristics:

- **U** (undefined), for a physical sequential file
- **F** (fixed), for a SYSIN file
- **FA** (fixed, with control characters), for a SYSOUT file

Once the record format is determined, CA ADS Batch supplies defaults for **block size** and **logical record length** based on the record format:

Fixed or Fixed-block Record Format

- **If the logical record length and the block size are zero (not specified)**, CA ADS Batch uses a block size of 6233 for physical sequential files, 80 for SYSIN files, and 133 for SYSOUT files, and uses a logical record length equal to the block size.
- **If the logical record length is not zero and the block size is zero**, CA ADS Batch uses a block size equal to the logical record length.
- **If the logical record length is zero and the block size is not zero**, CA ADS Batch uses a logical record size equal to the block size.

Variable or Variable-blocked Record Format

- **If the logical record length and the block size are zero (not specified)**, CA ADS Batch uses a block size of 6233 for physical sequential files, 80 for SYSIN files, and 133 for SYSOUT files, and uses a logical record length equal to the block size minus 4.
- **If the logical record length is not zero and the block size is zero**, CA ADS Batch uses a block size equal to the logical record length plus 4.
- **If the logical record length is zero and the block size is not zero**, CA ADS Batch uses a logical record size equal to the block size minus 4.

Undefined Record Format

- **If the logical record length is not zero and the block size is zero**, CA ADS Batch uses a block size equal to the logical record length.
- **If the block size is zero**, CA ADS Batch uses a block size of 6233 for physical sequential files, 80 for SYSIN files, and 133 for SYSOUT files.

Specifying Characteristics of a Concatenated Data Set

You define a concatenated data set (z/OS) by coding multiple DD statements in your JCL for a single ddname. For CA ADS Batch, characteristics that you specify for the file at the IDD level are enforced for every data set.

Rules for Concatenated Data Sets

Concatenated data sets must conform to the following rules:

- Data sets can be of different device types; for example, disk, tape, or scheduler (SYSIN).
- The data set with the largest block size must appear first in the concatenation.

- The record formats must be equivalent; that is:
  - All fixed or fixed blocked, or a mixture of the two
  - All variable or variable blocked, or a mixture of the two
  - All undefined
- The record lengths must correspond to the record format:
  - If the record format is **fixed** or **fixed blocked**, the logical record lengths must be equal.
  - If the record format is **variable** or **variable blocked**, the logical record lengths must be less than or equal to the first logical record length.
  - If the record format is **undefined**, the logical record length is not applicable.

In addition, the data sets must conform to standard z/OS rules for concatenated data sets; for example, the maximum number of data sets allowed in a concatenation is 255.

Specifying Characteristics of a Suspense File

Suspense files use the IDD-level characteristics specified for the file's associated input file. Suspense file characteristics are enforced when they are defined in the IDD file entity associated with the map. For example, if an input file's IDD description specifies a record format of fixed, that characteristic is enforced for the suspense file.

All characteristics supplied in the JCL or data set label of a suspense file must match those of the associated input file.

Tape Labels

Tape labeling options include the following:

SL	IBM standard labels
SUL	IBM standard and user labels
AL	ANSI labels
AUL	ANSI and user labels
NSL	Nonstandard labels
NL	No labels (unlabeled)

CA ADS Batch processes IBM standard and ANSI labels, but bypasses user labels and nonstandard labels on input. On output, user labels and nonstandard labels cannot be written to a tape file.

Application Components and Physical Data Sets

The relationship between application components and physical data sets begins with IDD file entities and ends with the description of the files in the JCL or data set labels.

IDD File Entities

You define IDD file entities to represent the input and output files in the application. You do not define file entities for suspense and log files. Suspense files use the file entity descriptions of their associated input files. Log files are not described in the data dictionary.

A single file entity can represent several files, as long as the characteristics included in the file entity are common to all files that the entity represents. For example, you do not have to specify any characteristics as part of the file entity; therefore, you can define one file entity to represent all files in an application.

More information on specifying file characteristics is provided earlier in this chapter.

IDD Record Entities

You define record entities that describe the layout of data for all input and output files in an application. You must associate each record entity with the appropriate file entity; if you are using the IDD DDDL compiler, you do this by specifying the INCLUDE WITHIN FILE clause of the RECORD statement.

Defining File Maps

You define file maps that associate file record entities with dialog variables storage.

Defining Dialogs

You define dialogs that associate file maps with the application.

Described below are some different combinations you can use when associating maps with dialogs in an application:

- You can associate a map with several dialogs.
- You can specify that a map is one dialog's input map and is the same or another dialog's output map.
- In an application that accesses a file with multiple record layouts, you define a map for each layout and associate each map with its own dialog.

You can associate a dialog with one input map, one output map, or both. You assign a suspense file to a dialog that is associated with an input file by specifying the runtime label (z/OS ddname, z/VSE filename) of the suspense file either during dialog definition or in a runtime control statement.

Note that IDD **elements** in records used by an CA ADS Batch application should not specify an external picture that includes P (for numeric data).

Runtime Labels

Runtime labels (z/OS ddname, z/VSE filename) associate file maps and suspense files with physical data sets. You specify runtime labels during dialog definition or in runtime control statements. Specifications made in control statements supersede those made during dialog definition.

You can specify the same runtime label for different file maps. For example, in an application that accesses a file with multiple record layouts, you define a file map for each record layout. During dialog definition or at runtime, you specify the same runtime label for each file map.

You can specify different runtime labels for the same file map. For example, in an application that reads records from one file and writes them to another, you can define a single file map for both files, since they have the same record layout. You associate the file map with one dialog as its input file map, and specify the runtime label of the input file. You associate the file map with the same dialog or another dialog as its output file map, and specify the runtime label of the output file.

Data Sets

You describe the application's physical data sets in the JCL and identify the data sets to the application by means of runtime labels.

Application Restartability

An application is restartable if it can be resumed at some checkpoint after an abend occurs. A restartable application must be able to reestablish the database, input, and output files, and the general application environment that existed at the checkpoint.

The following topics on application restartability are discussed separately below:

- The checkpoint record
- Taking a checkpoint at runtime
- Restarting an application after an abend

Checkpoint Record

The application should keep a checkpoint record that holds information about the status of the application. A checkpoint record should include:

- A checkpoint identification number
- Counts of the number of records, by dialog, that have been written to or read from each input, output, and suspense file
- A copy of all variable storage information that will have to be restored if the application abends, such as:
  - Condition flags
  - Subscripts
  - Report totals

Taking a Checkpoint at Runtime

Your application should take its checkpoints with the object of being able to restart after an abend at the highest level dialog.

The highest level dialog should perform the checkpoint or should pass control to a dialog whose purpose is to take the checkpoint. The checkpoint should do the following:

1. Update the input, output, and suspense file record count fields of the checkpoint record.
2. Close all files in order to clear the file buffers; the disposition of each file should be MOD.
3. Perform a database commit. Between checkpoints, you should leave the run unit open by using the NOSAVE parameter in LINK and INVOKE commands, and the NOFINISH parameter in TRANSFER commands. Committing the database at any point other than the checkpoint can jeopardize restartability.
4. Move required variable and status information to the checkpoint record.
5. Increment the checkpoint record identification number.
6. Write the checkpoint record to an output file or a queue file.
7. If the record is being written to an output file, close the file to ensure that the record is physically written.

The first two tasks must be performed by the dialogs that access the files. The dialog performing the checkpoint can link to each of these dialogs, with a flag set that indicates that a checkpoint is being performed. The dialogs can have premap processes that perform special processing when the flag is set. The processing includes incrementing the record count fields of the checkpoint record and closing the files.

Considerations

The following considerations apply:

- Dialogs to which control will be transferred during a checkpoint must not have as their entry points the mapin operation. If such an entry point is desired, the premap process can issue a READ TRANSACTION command immediately, unless a checkpoint is being performed.
- If a file has been accessed by more than one dialog, you must link to and issue a CLOSE command in each dialog that has accessed the file. If you do not, the file will not be physically closed.
- The record counts for a dialog's input, output, and suspense files are contained in the system-supplied fields $INPUT-COUNT, $OUTPUT-COUNT, and $ERROR-COUNT.
- You update the checkpoint record's record count fields by *adding* the system-supplied data fields for each dialog to the checkpoint record fields. Each time you issue a CLOSE command in a dialog, the system-supplied data fields for the closed files are reset to zero. At a checkpoint, therefore, the value of a data field equals the number of records read or written only since the last checkpoint.

Your application should have a checkpoint interval, based on a value such as the number of input records read or the time elapsed since the last checkpoint was taken. The checkpoint interval should not be so frequent as to substantially increase processing time, nor be so infrequent as to leave uncommitted more transactions than you would be willing to have to reprocess should the application abend.

Restarting the Application After an Abend

In central version, if the application abends, the runtime system automatically rolls back the database to its state at the time of the last database commit, which should have occurred at the time of the last checkpoint. (In local mode, you must use journals and rollback utilities to roll back the database.) Therefore, the database should already be set for restart.

Restart at the Highest Level Dialog

Your application should be restarted at the highest level dialog; the dialog can link to a dialog that performs the restart and then returns control to the highest level dialog.

Restart Steps

The restart should include the following steps:

1. **Read the checkpoint record—**Read the record, then, if the record is being stored in a sequential file, close the file.
2. **Restore the variable fields—**Move the saved variable fields from the checkpoint record to the appropriate records in dialog variable storage. You do not (and, in fact, cannot) restore the record count fields.
3. **Set the input files' pointers—**For each input file to be restored, link to a dialog whose sole function is to read the file until the file pointer is set at the proper position. The dialog determines how many records to read by using the record count field stored in the checkpoint record. After setting the file pointers, the dialog should close the file. The disposition of the restored file should be MOD.

   Note: If the file was accessed by several dialogs, you may have several checkpoint record fields for the file. If so, you should use the total count of the fields to determine how many records to read.

   If the file has multiple record layouts, you will have to use multiple dialogs to read the record and you must have an application structure that determines which dialog reads the various types of records.
4. **Set the output and suspense files' pointers—**For each output and suspense file to be restored, link to a dialog whose sole function is to copy the file to a new file, until the proper number of records have been written.

   Consider the following scenario. An application writes 100 records to an output file before taking a checkpoint. The application writes another 10 records to the file, then abends. The runtime system empties the buffer; as a result, 110 records have been physically written to the file. Your restart procedure must create a new file that consists of only the first 100 records.

   The dialog responsible for restoring the output file reads the file's records and writes them to a new file. The dialog determines how many records to copy by using the record count field stored in the checkpoint record. After the file has been restored, the dialog should close both files. The disposition of the new file should be MOD. The dialogs that will be writing to the new output or suspense file should specify the runtime label of the new file.

   Note: If the file was accessed by several dialogs, you may have several checkpoint record fields for the file. If so, you should use the total count of the fields to determine how many records to read.

   If the file has multiple record layouts, you will have to use multiple dialogs to copy the record and you must have an application structure that determines which dialog reads the various types of records.
5. **Return to the highest level dialog and resume processing**.

CA ADS Batch Trace Facility

The CA ADS Batch trace facility is a debugging aid that traces the flow of control and commands executed in an application at runtime.

Steps in the Trace Facility

To use the trace facility, perform the following steps:

1. **During dialog definition**, specify that a symbol table is to be included in the dialogs you want to trace. Symbol tables are required by the facility to list the process commands executed at runtime. If you are using the online dialog compiler, you request a symbol table on the Options and Directives screen.
2. **At runtime**, include TRACE statements as part of your control statements. TRACE statements specify:
   - **The dialogs to be traced—**One, several, or all dialogs in an application can be traced.
   - **When the dialogs are to be traced—**By default, a selected dialog is traced every time it becomes operative in the application thread. Alternatively, you can specify that tracing begin only after the dialog has become operative for the *n*th time, that tracing end when the dialog becomes operative for the *n*th time, or that tracing occur only every *n*th time that the dialog becomes operative.
   - **The level of detail of the trace—**You can specify one of three levels of detail for the trace of a dialog:
     a. **Dialog and process module execution—**The trace facility documents the beginning of dialog and premap and response process execution.
     b. **Subroutine execution—**The trace facility documents dialog and process module execution, as in level one, and additionally documents the beginning of subroutine execution.
     c. **Command execution—**The trace facility documents dialog, process module, and subroutine execution, as in level two, and additionally documents all commands executed within the dialog.

     Additionally, detailed information is provided on database commands and on database currency saves and restores.

   Syntax for the TRACE statement is provided under Runtime Control Parameters (see page 98).

The trace facility writes trace records to the log file as DEBUG records. You can print the records by using the CA ADS Batch print log utility, as discussed in CA ADS Batch Print Log Utility (see page 133). Alternatively, you can send all log file output directly to the printer at runtime.

Sample Output

Sample output from a trace facility run is shown in the report below. The output columns are as follows:

- **Execution number** (first column)—The execution occurrence for the dialog. The lines in the sample were created while the dialog was operative for the first time (00000001).
- **Dialog name** (second column)—The name of the dialog currently executing. The sample shows that dialog REPTEDIT executes first, then dialog HDRLINE, and then dialog REPTEDIT again.
- **Process name** (third column)—The name of the premap or response process currently executing. JEB-EMPL-REPT-EDIT is the first process executed in the sample trace facility run.

  Note: A line that documents the beginning of a dialog has the following literal in the process name field:

  ```
  ********  DIALOG-ENTRY  ********
  ```

- **Subroutine name** (fourth column)—The name of the process subroutine currently executing. PAGHDR is the first subroutine executed in the sample trace facility run.

  Note: A line that documents the beginning of a dialog, the beginning of a process, or a process command that is not within a subroutine has the following literal in the subroutine name field:

  ```
  **MAIN**
  ```

- **Sequence number field** (fifth column)—The IDD sequence number of the command being executed. The first four commands executed in the sample trace facility run begin at lines 100, 300, 400, and 500. These line numbers are the same ones used by ADSORPTS in its report of a dialog's processes.

  Note: For more information about ADSORPTS, see the *CA ADS Reference Guide*. The line that documents the beginning of a dialog, process, or subroutine has 00000000 in this field.

- **Process command** (sixth column)—The process command currently executing. The first command executed in the sample trace facility run is IF.

  Note: A line that documents the beginning of a dialog, process, or subroutine has the literal ENTRY in this field.

- **Command offset** (last column)—The hexadecimal offset of the command from the beginning of the dialog's fixed dialog block (FDB). These offsets are the same ones used by ADSORPTS in its list of command elements (CMEs) for a process.

Note: For more information about ADSORPTS, see the *CA ADS Reference Guide*.

The process being traced in the sample trace facility report does not include database commands or database currency saves and restores. However, if a process did and if tracing were being performed at the command execution level, the report would include detailed information about these database functions.

If the module is an included module, the included module name appears in the report after the last column.

```
00000001 REPTEDIT ********   DIALOG-ENTRY   ******** **MAIN** 00000000 ENTRY...
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . **MAIN** 00000000 ENTRY...
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . **MAIN** 00000100 IF     . 0004F8
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . **MAIN** 00000300 DC ACC . 000524
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . **MAIN** 00000400 MOVE   . 000538
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . **MAIN** 00000500 CALL   . 000574
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . CVRTDATE 00000000 ENTRY...
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . CVRTDATE 00013700 MOVE   . 0019DC
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . CVRTDATE 00013800 MOVE   . 001A94
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . CVRTDATE 00000000 MOVE   . 001B4C
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . CVRTDATE 00000000 GOBACK . 001C04
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . **MAIN** 00000600 MOVE   . 00058C
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . **MAIN** 00000700 CALL   . 0005C8
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . PAGHDR . 00000000 ENTRY...
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . PAGHDR . 00010000 MOVE   . 0013C0
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . PAGHDR . 00010100 ADD    . 0013FC
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . PAGHDR . 00010200 MOVE   . 001450
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . PAGHDR . 00010300 MOVE   . 00148C
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . PAGHDR . 00010400 MOVE   . 0014C8
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . PAGHDR . 00010500 MOVE   . 001504
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . PAGHDR . 00010600 CALL   . 001540
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . CENTJUST 00000000 ENTRY...
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . CENTJUST 00012400 MOVE   . 001774
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . CENTJUST 00012500 COMPUTE. 001818
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . CENTJUST 00012600 MOVE   . 00189C
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . CENTJUST 00012700 MOVE   . 0018D8
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . CENTJUST 00012800 GOBACK . 0019C4
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . PAGHDR . 00010700 MOVE   . 001558
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . PAGHDR . 00010800 LINK   . 001594
00000001 HDRLINE. ********   DIALOG-ENTRY   ******** **MAIN** 00000000 ENTRY...
00000001 HDRLINE. JEB-HEADER-WRITE                . **MAIN** 00000000 ENTRY...
00000001 HDRLINE. JEB-HEADER-WRITE                . **MAIN** 00000100 ADD    . 0003A0
00000001 HDRLINE. JEB-HEADER-WRITE                . **MAIN** 00000200 WRT TRAN 0003F4
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . **MAIN** 00000000 ENTRY...
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . PAGHDR . 00010900 MOVE   . 0015A4
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . PAGHDR . 00011000 MOVE   . 0015E0
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . PAGHDR . 00011100 MOVE   . 00161C
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . PAGHDR . 00011200 CALL   . 001658
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . CENTJUST 00000000 ENTRY...
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . CENTJUST 00012400 MOVE   . 001774
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . CENTJUST 00012500 COMPUTE. 001818
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . CENTJUST 00012600 MOVE   . 00189C
00000001 REPTEDIT JEB-EMPL-REPT-EDIT              . CENTJUST 00012700 MOVE   . 0018D8
```

Multiple External Run Units Under CA ADS Batch

Preparing the system to handle multiple external run units (ERUs) for CA ADS Batch is a DBA or system administrator issue, not a programmer issue.

However, an CA ADS Batch application run at a Distributed Database System (DDS) site should be designed so that a single database run unit is maintained for the application's duration. It is particularly important that the application avoid saving and then restoring database currencies if the application's run unit is accessing a database at a different DDS node than the default node for central version operations.

An CA ADS Batch application's database requests are routed to the batch interface module (IDMSINTB). IDMSINTB passes the request through the CA IDMS SVC (for central version requests) or directly to DC/UCF (for local mode requests). In DC/UCF, the database request is handled as an external run unit as is any other batch database request.

At system generation, it is necessary to specify a value for the MAXIMUM ERUS parameter in the SYSTEM statement. An CA ADS Batch application occasionally requires up to five ERUs at the same time:

- **User database run unit**, required if a dialog is accessing the database
- **Message run unit**, required to retrieve message text from the dictionary
- **Loader run unit**, required to retrieve load modules from the load area

If the MAXIMUM ERUS count is exceeded at runtime, your CA ADS Batch will be unable to load modules (for maps, dialogs, and so forth). If you receive a message to this effect, verify that the MAXIMUM ERUS specification for the system is at least 3. This specification is made in the system generation SYSTEM statement.

Note: For more information, see the *CA IDMS System Generation Guide*.

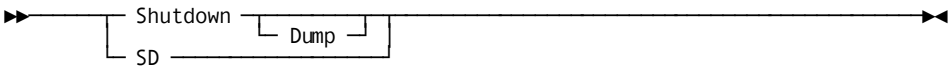
Operator Shutdown

CA ADS Batch enables the console operator to instruct the runtime system to terminate an application.

If operator shutdown is enabled at runtime, the runtime system begins by issuing a WTOR (write-to-operator with reply) macro. At any time during application execution, the operator can issue a SHUTDOWN command, described below, that causes the runtime system to terminate the application with an optional dump. If shutdown is disabled, the operator can terminate the application only by canceling the job.

The system default for operator shutdown is determined using the ADSOBSYS utility; the ADSOBSYS default is DISABLED. Control statements specified at runtime can override the system default. For more information about enabling operator shutdown, see ADSOBSYS (see page 115), and Control Statements (see page 99).

If operator shutdown is enabled, the operator can issue a SHUTDOWN command to terminate the application. Syntax for the SHUTDOWN command is shown below:



SHUTDOWN

Specifies that the runtime system is to terminate the application.

DUMP

Specifies that a system dump of the application is to be produced.

SD

Is equivalent to SHUTDOWN DUMP.

Note: Under BS2000/OSD, the SHUTDOWN command must be entered by means of the BS2000/OSD /INFORM-PROGRAM command. For example, if the application runs as an interactive task, enter:

```
/INFORM-PROGRAM MSG='SHUTDOWN DUMP'
```

Runtime Control Parameters

Runtime control parameters determine the CA ADS Batch runtime environment. These parameters can be specified at system generation using ADSOBSYS and/or at runtime using PARM field parameters and control statements.

When a runtime control parameter is specified in more than one place, the following order of precedence applies:

1. Control statements
2. PARM field parameters
3. ADSOBSYS

PARM field parameters and control statements are discussed separately below. ADSOBSYS is discussed in ADSOBSYS (see page 115).

PARM Field Parameters

Input parameters allow you to specify the following:

- **A mainline dialog or application task code**—The runtime system begins application execution at the specified dialog or at the function associated with the specified task code. The function must be associated with a mainline dialog.

  Alternatively, you can specify a dialog or task code using the ENTRY POINT control statement, as described later in this chapter. The ENTRY POINT statement overrides the input parameter specification.

- **Runtime parameter**—You can optionally specify an alphanumeric character string as a parameter that is to be passed to the application. Dialog processes can read the string into variable storage by issuing an ACCEPT RUN PARAMETERS command, as described in Process Command Language (see page 35).

Specifying Parameters

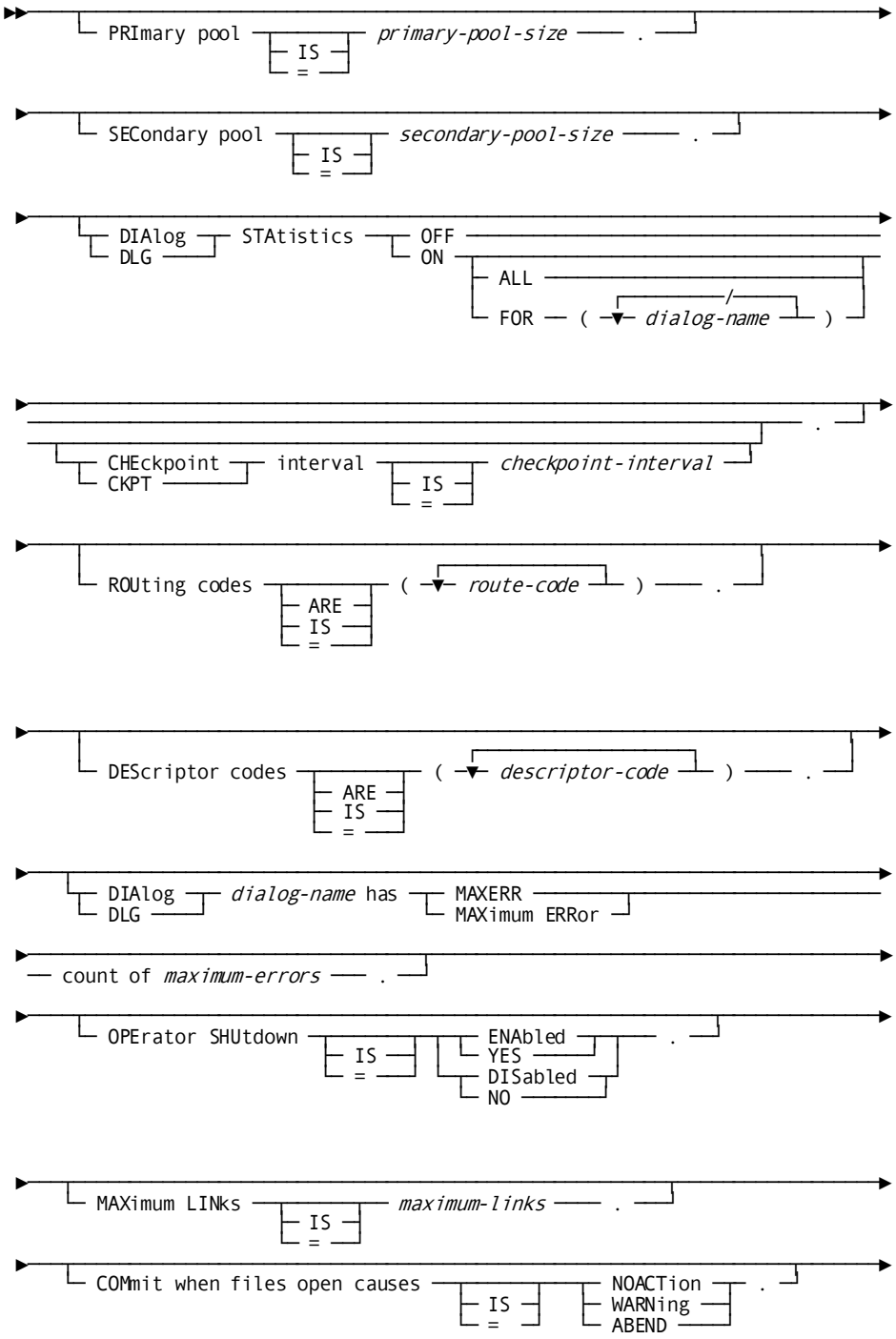
Under z/OS and z/VSE, you specify input parameters in the PARMS field parameter of the EXEC statement. The mainline dialog or application task code is specified as the first parameter. The runtime parameter is specified as the third parameter. The second parameter is reserved for future use. Parameters must be separated by commas. The value assigned to the job variable may not exceed 10 characters. For example:

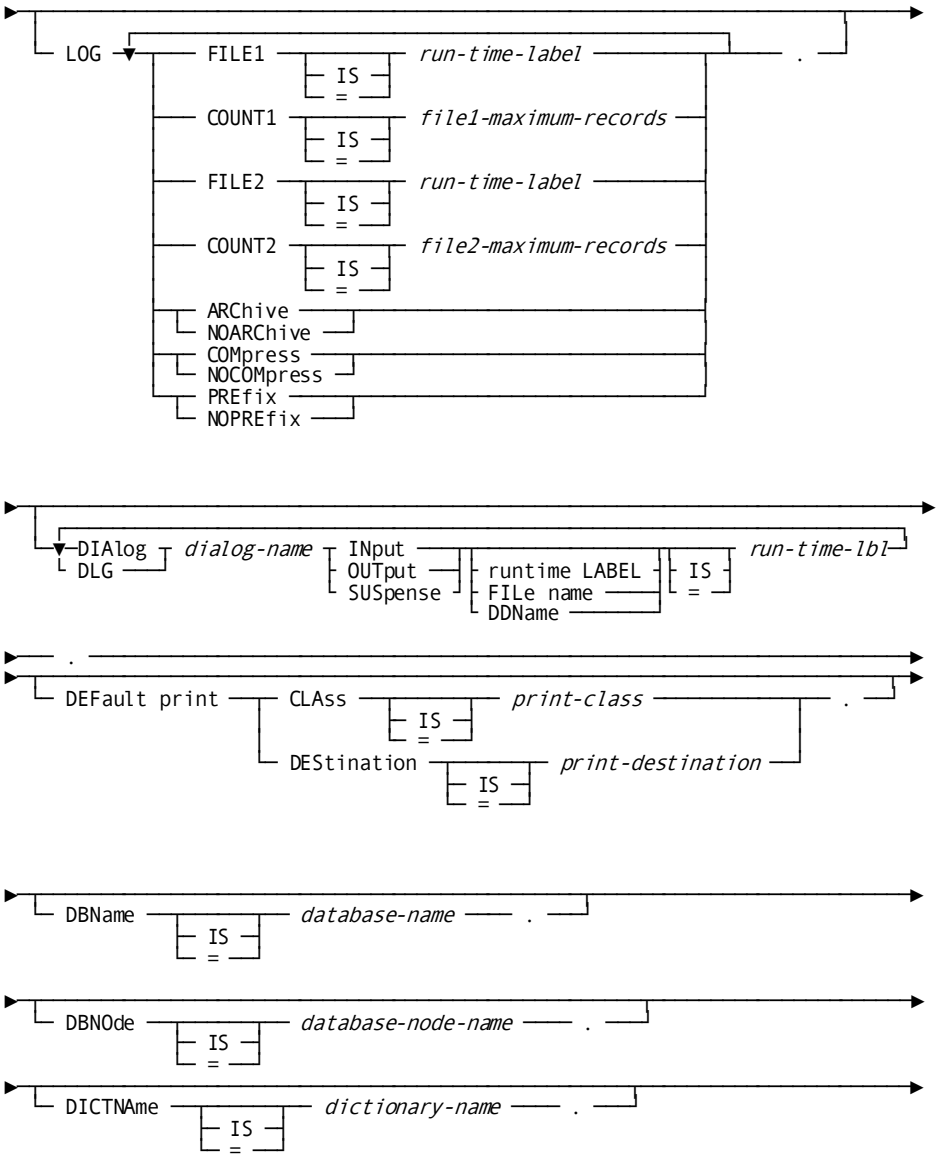
```
PARM='DIALOG1'

PARM='TASKA,,ABCDEFG'

PARM=',,ABCDEFG'
```

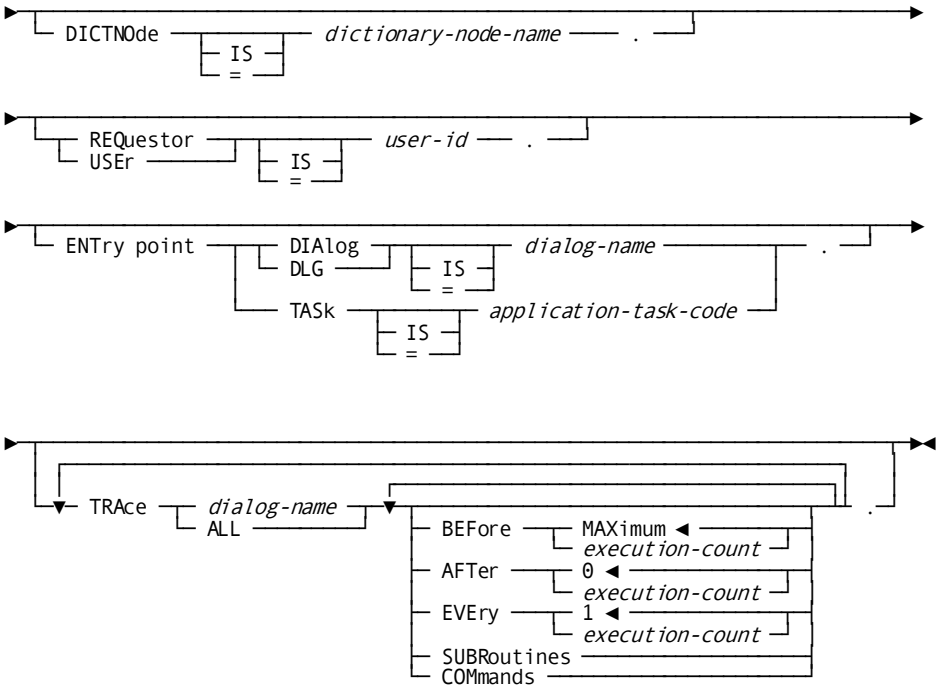
Control Statements

Control statements help to establish the runtime environment for an CA ADS Batch application. Each control statement must be terminated by a period. Control statements can appear in any order.

Syntax for the control statements is shown below:




LOG
FILE1
IS
=
run-time-label
.
COUNT1
IS
=
file1-maximum-records
FILE2
IS
=
run-time-label
COUNT2
IS
=
file2-maximum-records
ARChive
NOARChive
COMpress
NOCOMpress
PREfix
NOPREfix
DIAlog
DLG
dialog-name
INput
OUTput
SUSpense
runtime LABEL
FILe name
DDName
IS
=
run-time-lbl
.
DEFault print
CLAss
IS
=
print-class
.
DEStination
IS
=
print-destination
DBName
IS
=
database-name
.
DBNOde
IS
=
database-node-name
.
DICTNAme
IS
=
dictionary-name
.



PRIMARY POOL IS *primary-pool-size*

Specifies the size, in bytes, of the primary buffer pool. *Primary-pool-size* is an integer in the range 0 through 2,147,483,647. The default is determined by specifications made using ADSOBSYS; the ADSOBSYS default is set at system generation.

Note: For more information about specifying primary pools, see the discussion of the ADSO statement in the *CA IDMS System Generation Guide*.

SECONDARY POOL IS *secondary-pool-size*

Specifies the size, in bytes, of the secondary buffer pool. The secondary pool is allocated from the system storage pool when the primary buffer pool becomes full. *Secondary-pool-size* is an integer in the range 0 through 2,147,483,647. The default is determined by specifications made using ADSOBSYS; the ADSOBSYS default is set at system generation.

DIALOG STATISTICS OFF/ON

Specifies whether dialog statistics are to be collected during application execution.

OFF specifies that statistics are not to be collected.

ON specifies that statistics are to be collected. The default is determined by specifications made using ADSOBSYS; the ADSOBSYS default is OFF.

Parameters included in the ON clause are as follows.

ALL/FOR (*dialog-name*)

Specifies whether dialog statistics are maintained for all dialogs (ALL) or for only selected dialogs in an application. Multiple dialogs must be separated by commas.

The default is determined by specifications made using ADSOBSYS. ADSOBSYS allows you to specify ALL or SELECTED. If you specify SELECTED, you must list the selected dialogs in the DIALOG STATISTICS control statement at runtime.

CHECKPOINT INTERVAL IS *checkpoint-interval*

Specifies the frequency with which dialog statistics are to be written to the log file. Statistics are written to the log file once every *checkpoint-interval* time that statistics are accumulated. CA ADS Batch accumulates statistics every time a dialog issues a control command. The default checkpoint interval is determined by specifications made using ADSOBSYS; the ADSOBSYS default is set at system generation.

ROUTING CODES ARE (*route-code*)

(z/OS only) specifies the z/OS operator-message routing codes, as described in the applicable operating system supervisor services and macro instructions. This parameter supplies the value of the ROUTCDE parameter for WTO (write-to-operator) macros issued by the system. *Route-code* is an integer in the range 1 through 16. Multiple routing codes must be separated by commas. The default routing codes are determined by specifications made using ADSOBSYS; the ADSOBSYS default is 11.

DESCRIPTOR CODES ARE (*descriptor-code*)

(z/OS only) specifies the z/OS operator-message description code, as described in the applicable operating system supervisor services and macro instructions. This parameter supplies the descriptor code to the DESC parameter for WTO (write-to-operator) macros issued by the system. *Descriptor-code* is an integer in the range 1 through 16. The default descriptor codes are determined by specifications made using ADSOBSYS; the ADSOBSYS default is 7.

DIALOG *dialog-name* HAS MAXERR COUNT OF *maximum-errors*

Specifies the maximum number of error records that can be sent to the suspense file of the specified dialog. If this number is exceeded at runtime, the runtime system terminates the application. For example, if the maximum error count is 1, the runtime system terminates the application when the second error record is to be written. If the maximum error count is 0, the runtime system allows an unlimited number of error records to be sent to a suspense file. *Maximum-errors* is an integer in the range 0 through 32,767. The default is determined by specifications made using ADSOBSYS; the ADSOBSYS default is 0.

OPERATOR SHUTDOWN IS ENABLED/DISABLED

Specifies whether the operator can send a request to the runtime system to terminate an application. If operator shutdown is enabled, the runtime system begins application execution by issuing a WTOR (write-to-operator with reply) macro. At any time during application execution, the operator can issue a SHUTDOWN command, as described earlier in this chapter. The SHUTDOWN command causes the runtime system to terminate the application with an optional dump. If shutdown is disabled, the operator can terminate the application only by canceling the job. The default is determined by specifications made using ADSOBSYS; the ADSOBSYS default is DISABLED.

COMMIT WHEN FILES OPEN CAUSES NOACTION/WARNING/ABEND

Specifies the action to be taken when a run unit is committed before all files used in the application have been closed, as follows:

- **NOACTION** specifies that no action is taken.
- **WARNING** specifies that a warning message is sent to the log file.
- **ABEND** specifies that the application is abended.

The default is determined by specifications made using ADSOBSYS; the ADSOBSYS default is ABEND.

Committing a run unit when files are open can jeopardize application restartability after an abend, as described under Application Restartability (see page 90).

LOG

LOG parameters specify defaults for the log file; all LOG parameters are optional.

FILE1 IS *run-time-label*

Specifies the runtime label (z/OS ddname, z/VSE filename) of the primary log file. The default is determined by specifications made using ADSOBSYS; the ADSOBSYS default is ADSLOGA.

COUNT1 IS *file1-maximum-records*

Specifies the number of log records that are written to the primary log file before the file is considered full. This parameter is meaningless for spool or tape log files.

If *file1-maximum-records* is reached at runtime, the runtime system switches to the secondary log file or, if no secondary log file is allocated, wraps around to the beginning of the primary log file.

If 0 is specified, no predefined limit is placed on the number of records written to the primary log file. If space for a disk log file is exceeded at runtime, the runtime system abnormally terminates the application.

File1-maximum-records is an integer in the range 1 through 99999. The default is determined by specifications made using ADSOBSYS; the ADSOBSYS default is 0.

FILE2 IS *run-time-label*

Specifies the runtime label of the secondary log file. The default is determined by specifications made using ADSOBSYS.

COUNT2 IS *file2-maximum-records*

Specifies the number of log records that are written to the secondary log file before the file is considered full.

If *file2-maximum-records* is reached at runtime, the runtime system switches back to the primary log file.

If 0 is specified, no predefined limit is placed on the number of records written to the secondary log file. If space for a disk log file is exceeded at runtime, the runtime system abnormally terminates the application.

File2-maximum-records is an integer in the range 1 through 99999. The default is determined by specifications made using ADSOBSYS; the ADSOBSYS default is 0.

ARCHIVE/NOARCHIVE

(z/OS only) specifies whether log file archiving is to be performed at runtime when a log file is full. For more information about log file archiving, see "Log files" in CA ADS Batch Concepts (see page 15). The default is determined by specifications made using ADSOBSYS; the ADSOBSYS default is NOARCHIVE.

COMPRESS/NOCOMPRESS

Specifies whether log records are compressed in the log file to save space at runtime. The default is determined by specifications made using ADSOBSYS. If neither ADSOBSYS nor a runtime control statement specifies COMPRESS or NOCOMPRESS, the runtime system uses the z/OS, or z/VSE device-type assignment to determine whether to compress the records:

- In **z/OS**, records are not compressed if the assignment is SYSOUT; otherwise, records are compressed.
- In **z/VSE**, records are not compressed if the device type is PRINTER (as specified at system generation using ADSOBSYS or at runtime); otherwise, records are compressed.

PREFIX/NOPREFIX

Specifies whether a prefix is to precede each log record. Prefixes are used by the print log utility to select records. The default is determined by specifications made using ADSOBSYS. If neither ADSOBSYS nor a runtime control statement specify PREFIX or NOPREFIX, the runtime system uses the z/OS or z/VSE device type assignment to determine whether to include a prefix:

- In **z/OS**, a prefix is not included if the assignment is SYSOUT; otherwise, a prefix is included.
- In **z/VSE**, a prefix is not included if the device type is PRINTER (as specified at system generation using ADSOBSYS or at runtime); otherwise, a prefix is included.

DIALOG

The **DIALOG** statement associates a dialog's input, output, and suspense files with runtime labels (z/OS ddnames, z/VSE filenames). Each input, output, and suspense file of each dialog must be associated with a runtime label.

Note: You can associate a file with a runtime label during dialog definition as well. In such a case, you do not have to specify a runtime label in a control statement at runtime. If a runtime label is specified in both places, the control statement specification takes precedence.

DIALOG statement parameters are described below.

dialog-name

The name of the dialog.

INPUT/OUTPUT/SUSPENSE

Specifies whether the file identification information that follows pertains to the dialog's input, output, or suspense file.

RUNTIME LABEL/FILE NAME/DDNAME IS *run-time-label*

Specifies the runtime label input, output, or suspense file. RUNTIME LABEL, FILE NAME, and DDNAME are synonymous.

Note: To request that a suspense file not be allocated for the input file, even though a suspense file runtime label was specified during dialog definition, specify blanks within single quotation marks, as in the example below:

```
DIALOG XXX INPUT RUNTIME LABEL INFILE
           SUSPENSE RUNTIME LABEL ' '.
```

The following example specifies information about a z/OS dialog's input, output, and suspense files:

```
DIALOG DIALOG1
  INPUT DDNAME IS INPUT1
OUTPUT DDNAME IS OUTPUT1
  SUSPENSE DDNAME IS SUS1.
```

DEFAULT PRINT CLASS IS *print-class*/DESTINATION IS *print-destination*

Specifies the default print class to which a report is assigned or the default print destination to which a report is routed. A report is created using WRITE PRINTER commands. *Print-class* must be an integer in the range 1 through 64. *Print-destination* is a 1- to 8-character destination not enclosed in quotation marks.

DBNAME IS *database-name*

Optionally specifies the CA IDMS/DB database to be accessed by the application.

DBNODE IS *database-node-name*

Optionally specifies the DDS node that controls the CA IDMS/DB database to be accessed by the application.

DICTNAME IS *dictionary-name*

Optionally specifies the data dictionary in which the definitions and load modules for the requested application are stored.

DICTNODE IS *dictionary-node-name*

Optionally specifies the DDS node that controls the data dictionary in which the definitions and load modules for the requested application are stored.

REQUESTOR IS *user-id*

Specifies the 1- to 32-character id of a user. The user id can be accepted into dialog variable storage by issuing an ACCEPT USER ID command, and is automatically moved to the prefix of all log file records written during the application. *User-id* is not enclosed in quotation marks.

ENTRY POINT

Specifies the entry point into the application.

If the ENTRY POINT statement is not included, an entry point must be specified as an input parameter, as discussed earlier in this chapter. The ENTRY POINT statement overrides the input parameter.

DIALOG IS *dialog-name*

Specifies the name of a mainline dialog.

TASK IS *application-task-code*

Specifies a task code associated with an application function. The function must be associated with a mainline dialog.

TRACE

Activates the CA ADS Batch trace facility for a dialog.

For a detailed discussion of the trace facility, see CA ADS Batch Trace Facility (see page 94).

TRACE statement parameters are described below.

***dialog-name*/ALL**

Specifies the dialog for which tracing is activated; the parameters BEFORE, AFTER, EVERY, COMMANDS, and SUBROUTINES apply to the named dialog. ALL specifies that tracing is performed for all dialogs; the parameters BEFORE, AFTER, EVERY, COMMANDS, and SUBROUTINES apply to all dialogs that have not been named explicitly in the TRACE statement.

To selectively activate tracing for several dialogs, repeat the parameters of the TRACE statement for each dialog.

BEFORE MAXIMUM/*execution-count*

Specifies that tracing is to be performed each time the dialog becomes operative, up to but not including the *execution-count*th time. MAXIMUM (default) specifies tracing is to be performed every time the dialog becomes operative, unless restricted by AFTER and EVERY specifications.

AFTER 0/*execution-count*

Specifies that tracing is to be performed each time the dialog becomes operative beyond the *execution-count* time. Zero (default) specifies tracing is to be performed every time the dialog becomes operative, unless restricted by BEFORE and EVERY specifications.

EVERY 1/*execution-count*

Specifies that tracing is to be performed each time the execution count for the dialog reaches a multiple of *execution-count*. One (default) specifies tracing is to be performed every time the dialog becomes operative, unless restricted by AFTER and BEFORE specifications. specifies that tracing is to be performed only every *execution-count*th time that the dialog becomes operative. One (default) specifies that tracing is performed each time the dialog becomes operative, unless restricted by BEFORE and AFTER specifications.

SUBROUTINES

Specifies that subroutine entry points are to be documented, in addition to dialog and process entry points.

COMMANDS

Specifies that process commands are to be documented, in addition to dialog, process, and subroutine entry points.

If neither SUBROUTINES nor COMMANDS is specified, only dialog and process entry points are documented for the specified dialog.

Multiple TRACE statements can be specified for the same dialog. The last BEFORE, AFTER, and EVERY specifications coded for a dialog apply to that dialog for the entire execution of the application. The most detailed level of documentation coded applies to the dialog for the entire execution; COMMANDS is more detailed than SUBROUTINES, which is more detailed than no specification at all.

Loading Runtime Components

Order of Precedence

Runtime components (such as subschema load modules, dialog load modules, application definition blocks, and map load modules) can be stored in load areas and load libraries. The runtime system uses the following order of precedence when loading a component:

1. Main memory of the batch region, in the event that the component has already been loaded
2. The load area of the secondary dictionary, if any, specified in a control statement
3. The load area of the system's primary dictionary
4. A load library specified in the STEPLIB statement, if any

   Note: If an application snaps the application definition block (ADB) or fixed dialog block (FDB) of an application or dialog that was loaded from STEPLIB, only the header portion is snapped.

   An error status of 1474 may be returned if an CA ADS Batch program running in local mode attempts to use a subschema that was compiled for use on CA IDMS/DC without a DMCL name.

   CA ADS Batch can link to a user-written program. The program can perform database commands. The program, however, cannot perform QUEUE, WRITE PRINTER, or other DC/UCF-oriented commands.

   A user-written COBOL, or PL/I program accessed by an CA ADS Batch application must have been submitted to the DML preprocessor with a mode of BATCH.

JCL

The JCL for using the CA ADS Batch runtime system (ADSBATCH) to execute an application is shown below.

Sample z/OS JCL Under Central Version ADSBATCH (z/OS)

```
//ADSBATCH EXEC PGM=ADSBATCH,REGION=1024K,parm=parameter
//STEPLIB  DD   DSN=idms.dba.loadlib,DISP=SHR
//             DSN=idms.loadlib,DISP=SHR
//IDMSSNAP DD   SYSOUT=A
//sysctl   DD   DSN=idms.sysctl,DISP=SHR
//adsloga  DD   DSN=log.file,DISP=OLD
//file.descriptions
//SYSLST   DD   SYSOUT=A
//SYSIDMS  DD   *
DMCL=dmcl-name
DICTNAME=dictionary-name
Other SYSIDMS parameters, as appropriate
/*
//SYSIPT   DD   *
control statements
/*
//SYSUDUMP DD   SYSOUT=A
```

parm=parameter	an optional PARM parameter
idms.dba.loadlib	data set name of the load library containing the DMCL and database name table load modules
idms.loadlib	data set name of the load library containing the CA IDMS executable modules
sysctl	ddname of SYSCTL file
idms.sysctl	data set name of SYSCTL file
adsloga	ddname of the log file
log.file	data set name of the log file
file.descriptions	file descriptions of all input, output, and suspense files. The ddnames specified must match those specified in control statements or dialog definitions.

Note: To get a snap dump when an abend occurs during a mapping operation, you must specify a DD statement with the name IDMSSNAP. The IDMSSNAP file stores the snap dump. an example of a mapping abend is when an input record that contains errors cannot be written to a suspense file because the MAXIMUM ERRORS for the dialog, which is specified at system generation or at runtime, has been exceeded.

Archive Log File

For an archive log file, specify a ddname of ARCLOG*n*, where *n* is 1 for the first archive file, 2 for the second archive file, and so forth, up to 9. The following sample JCL allocates the first archive log file:

```
//ARCLOG1  DD  DSN=MEN.CC.RQE.ARCTAP1,DISP=(NEW,CATLG,DELETE),
//           UNIT=(TAPE,,DEFER),LABEL=(,SL),VOL=(,RETAIN),
//           DCB=(RECFM=VB,LRECL=320,BLKSIZE=12804)
```

Local mode

To execute ADSBATCH in local mode, the following steps are required:

1. Remove the sysctl DD statement.
2. Add the following statements:

```
//sysjrnl DD   DSN=idms.tapejrnl,DISP=NEW,UNIT=tape
//dictdb  DD   DSN=idms.dictdb,DISP=OLD
//dloddb  DD   DSN=idms.dloddb,DISP=SHR
//dmsgdb  DD   DSN=idms.dmsgdb,DISP=SHR
//dlogdb  DD   DSN=idms.dlogdb,DISP=SHR
additional journal file assignments, as required
```

Note: Under local mode, if you are running a multi-level application which transfers control using a LINK command without the NOSAVE option, you must code the LOCKING=ON parameter in the SYSIDMS parameter file.

sysjrnl	ddname of journal file
idms.tapejrnl	data set name of journal file
tape	symbolic device name of journal file
dictdb	ddname of data dictionary
idms.dictdb	data set name of data dictionary
dloddb	ddname of data dictionary load area
idms.dloddb	data set name of data dictionary load area
dmsgdb	ddname of data dictionary message area
idms.dmsgdb	data set name of data dictionary message area
dlogdb	ddname of data dictionary log area
idms.dlogdb	data set name of data dictionary log area

Sample z/VSE JCL Under Central Version ADSBATCH (z/VSE)

```
// DLBL SYSCTL,'idms.sysctl', 1999/365,SD
// DLBL     ciln,'idms.cilib'
// EXTENT   ,nnnnnn
// LIBDEF   *,SEARCH=(ciln),TEMP
// DLBL     adsloga,'log.file'
// EXTENT   sysnnn,VOLSER,,,start-track,end-track
// ASSGN    sysnnn,DISK,VOL=VOLSER,SHR
// DLBL SYSIDMS,'#SYSIPT'
// file.descriptions
// EXEC     IDMSFILE,SIZE=(AUTO,128K)
SYSIDMS control statements
IDMSFILE control statements
ADSBATCH control statements
DIALOG control statements
```

idms.sysctl	data set name of SYSCTL file
ciln	filename of core-image library
idms.cilib	file-id of DC/UCF core-image library
nnnnnn	volume serial number
adsloga	filename of the log file
log.file	file-id of the log file
sysnnn	logical unit of the device on which the log file is located
start-track	starting track of the log file data set
end-track	ending track of the log file data set
file.descriptions	file descriptions of all input, output, and suspense files used in the application. Filenames specified must match those specified in control statements.
SYSIDMS control statements	control statements used by SYSIDMS to describe physical runtime environments as described in *CA IDMS Common Facilities Guide*—Volume 2.
IDMSFILE control statements	control statements used by IDMSFILE to describe the characteristics of the input, output, suspense, and log files used in the application, as described in z/VSE File Characteristics Program (see page 123). This must include the 'RUN PROGRAM ADSBATCH' card.

DIALOG control statements	control statements required if any of the dialogs to be executed have an input map with the 'SYSIPT' label.
ADSBATCH control statements	control statements used by ADSBATCH to establish the runtime environment.

What to Consider

A /* must follow each SYSIPT file. If an abend occurs during a file mapping operation, the snap dump is written automatically to SYSLST.

Local Mode

To execute ADSBATCH in local mode, the following steps are required:

1. Remove the SYSCTL DLBL card
2. Add the following statements:

```
// DLBL     dictdb,'idms.dictdb',,DA
// EXTENT   sys005,nnnnnn
// ASSGN    sys005,DISK,VOL=nnnnnn,SHR
// DLBL     dloddb,'idms.dloddb',,DA
// EXTENT   sys017,nnnnnn
// ASSGN    sys017,DISK,VOL=nnnnnn,SHR
// DLBL     dmsgdb,'idms.dmsgdb',,DA
// EXTENT   sys016,nnnnnn
// ASSGN    sys016,DISK,VOL=nnnnnn,SHR
// DLBL     dlogdb,'idms.dlogdb',,DA
// EXTENT   sys019,nnnnnn
// ASSGN    sys019,DISK,VOL=nnnnnn,SHR
// TLBL     sysjrnl,'idms.tapejrnl',,nnnnnn,,f
// ASSGN    sys009,TAPE,VOL=nnnnnn
additional journal file assignments, as required
```

dictdb	filename of data dictionary
idms.dictdb	file-id of data dictionary
sys005	logical unit assignment for data dictionary
nnnnnn	volume serial number
dloddb	filename of data dictionary load area
idms.dloddb	file-id of data dictionary load area
sys017	logical unit assignment for data dictionary load area
dmsgdb	filename of data dictionary message area
idms.dmsgdb	file-id of data dictionary message area

sys016	logical unit assignment for data dictionary message area
dlogdb	filename of data dictionary log area
idms.dlogdb	file-id of data dictionary log area
sys019	logical unit assignment for data dictionary log area
sysjrnl	filename of tape journal file
idms.tapejrnl	file-id of tape journal file
f	file number of tape journal file
sys009	logical unit assignment for tape journal file

Local Mode

To execute ADSBATCH in local mode, the following steps are required:

1. Remove the ADD-FILE-LINK statement for sysctl
2. Add the following statements:

```
/ADD-FILE-LINK L-NAME=dictdb,F-NAME=idms.appldict.dictdb,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=dloddb,F-NAME=idms.appldict.dloddb,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=dcmsg,F-NAME=idms.sysmsg.ddldcmsg,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=sysjrnl,F-NAME=idms.tapejrnl
additional user database file assignments
additional journal file assignments, as required
```

dictdb	linkname of data dictionary file
idms.appldict.dictdb	filename of data dictionary file
dloddb	linkname of data dictionary load area file
idms.appldict.dloddb	filename of data dictionary load area file
dcmsg	linkname of data dictionary message area file
idms.sysmsg.ddldcmsg	filename of data dictionary message area file
sysjrnl	linkname of tape journal file
idms.tapejrnl	filename of tape journal file

Appendix A: ADSOBSYS

This section contains the following topics:

Overview

ADSOBSYS is a utility that builds a load module (ADSOOPTI) that supplies system generation parameters to ADSOBCOM and the CA ADS Batch runtime system. ADSOBSYS must be run once for each DC/UCF system at an installation. Additionally, ADSOBSYS must be run whenever Application Development System system generation parameters are changed.

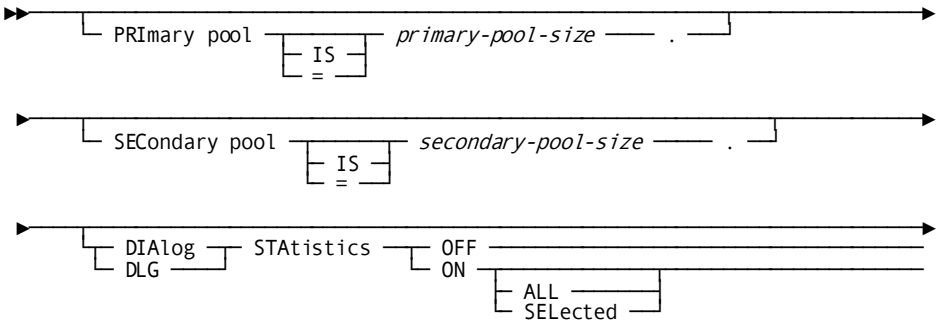
Control Statements

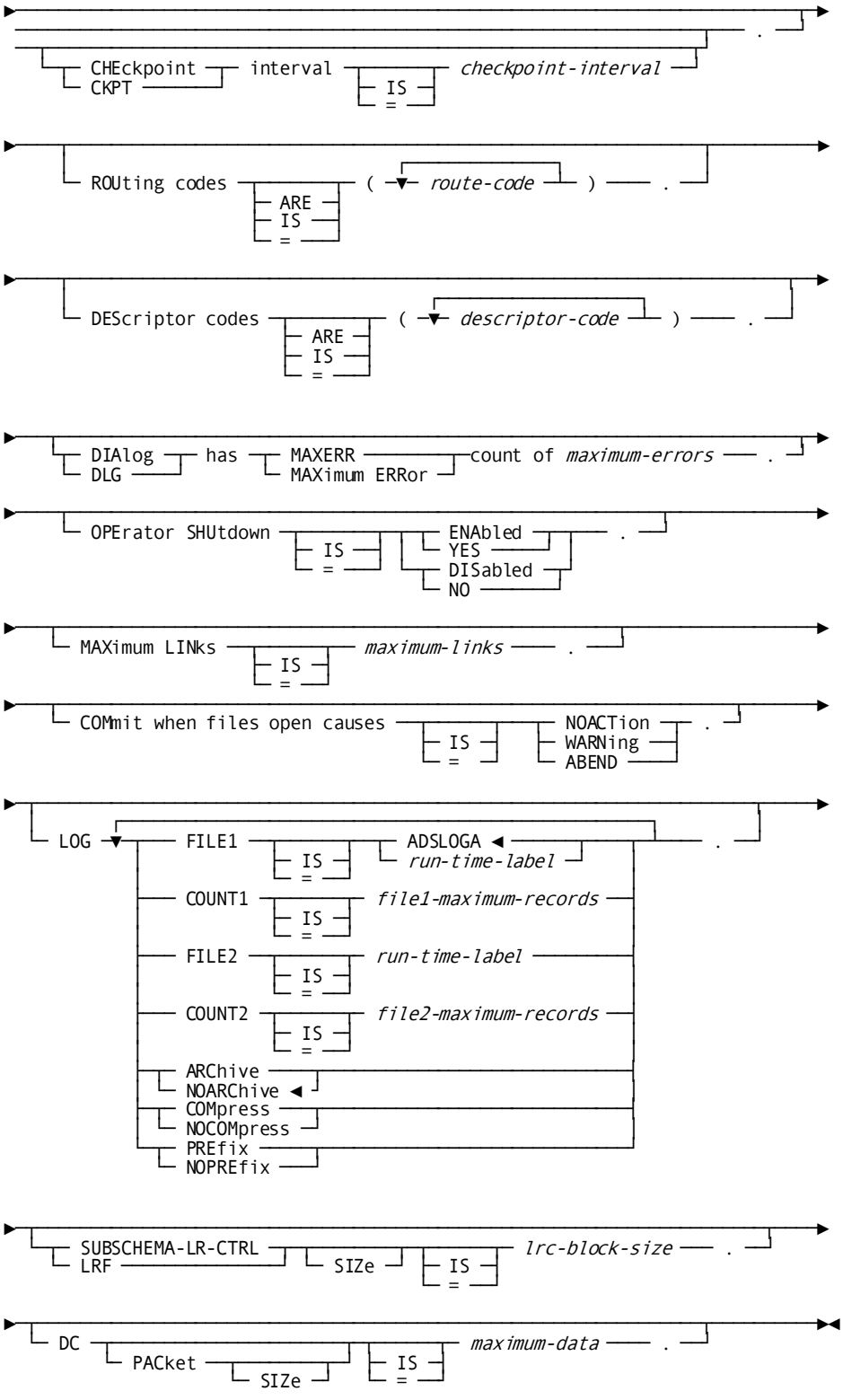
Control statements are available that enable you to specify default parameters for CA ADS Batch application execution. At runtime, you can override any of these parameters by using control statements, as described in Runtime Considerations (see page 79).

The control statements must follow the SYSTEM statement in the ADSOBSYS job stream.

Note: The ADSOBSYS parameters shown in this appendix affect only the CA ADS Batch runtime environment. They do not affect the CA ADS online runtime environment. For example, the STATISTICS parameters specifies how statistics are to be collected only for an CA ADS Batch application. You specify the CA ADS online environment at system generation in the ADSO statement.

Syntax for the CA ADS Batch control statements is shown below:




CHEckpoint
CKPT
interval
IS
=
checkpoint-interval
ROUting codes
ARE
IS
=
(
route-code
)
DEScriptor codes
ARE
IS
=
descriptor-code
DIAlog
DLG
has
MAXERR
MAXimum ERRor
count of
maximum-errors
OPErator SHUtdown
IS
=
ENAbled
YES
DISabled
NO
MAXimum LINks
IS
=
maximum-links
COMmit when files open causes
IS
=
NOACTion
WARNing
ABEND
LOG
FILE1
IS
=
ADSLOGA
run-time-label
COUNT1
IS
=
file1-maximum-records
FILE2
IS
=
run-time-label
COUNT2
IS
=
file2-maximum-records
ARChive
NOARChive
COMpress
NOCOMpress
PREfix
NOPREfix
SUBSCHEMA-LR-CTRL
LRF
SIZe
IS
=
lrc-block-size
DC
PACket
SIZe
IS
=
maximum-data

PRIMARY POOL IS *primary-pool-size*

Specifies the size, in bytes, of the primary buffer pool. *Primary-pool-size* is an integer in the range 0 through 2,147,483,647. The default is determined at system generation; the system generation default is 4,000.

Note: For more information about specifying primary pools, see the discussion of the ADSO statement in *CA IDMS System Generation Guide*.

SECONDARY POOL IS *secondary-pool-size*

Specifies the size, in bytes, of the secondary buffer pool. The secondary pool is allocated from the system storage pool when the primary buffer pool becomes full. *Secondary-pool-size* is an integer in the range 0 through 2,147,483,647. The default is determined at system generation; the system generation default is 2,000.

DIALOG STATISTICS OFF/ON

Specifies whether dialog statistics are to be collected during application execution.

OFF specifies that statistics are not to be collected.

ON specifies that statistics are to be collected.

The default is determined at system generation; the system generation default is OFF.

Parameters included in the ON clause are as follows.

ALL/SELECTED

Specifies whether dialog statistics are maintained for all dialogs or for only selected dialogs in an application. The default is determined at system generation.

Dialogs are selected at runtime through the DIALOG STATISTICS control statement.

CHECKPOINT INTERVAL IS *checkpoint-interval*

Specifies the frequency with which dialog statistics are to be written to the log file. Statistics are written to the log file once every time statistics are accumulated for the *checkpoint-interval* time. CA ADS Batch accumulates statistics every time a dialog issues a control command. The default checkpoint interval is determined at system generation; the system generation default is 200.

ROUTING CODES ARE (*route-code*)

(z/OS) specifies the OS operator-message routing codes, as described in the applicable operating system supervisor services and macro instructions. This parameter supplies the value of the ROUTCDE parameter for WTO (write-to-operator) macros issued by the system. *Route-code* is an integer in the range 1 through 16; the default is 11. Multiple routing codes must be separated by commas.

DESCRIPTOR CODES ARE (*descriptor-code*)

(z/OS) specifies the z/OS operator-message description code, as described in the applicable operating system supervisor services and macro instructions. This parameter supplies the descriptor code to the DESC parameter for WTO (write-to-operator) macros issued by the system. *Descriptor-code* is an integer in the range 1 through 16; the default is 7.

DIALOG HAS MAXERR COUNT OF *maximum-errors*

Specifies the maximum number of error records that can be sent to the suspense file of any dialog. If this number is exceeded at runtime, the runtime system terminates the application. For example, if the maximum error count is 1, the runtime system terminates the application when the second error record is to be written. If the maximum error count is 0, the runtime system allows an unlimited number of error records to be sent to a suspense file. *Maximum-errors* is an integer in the range 0 through 32,767; the default is 0.

OPERATOR SHUTDOWN IS ENABLED/DISABLED

Specifies whether the operator can send a request to the runtime system to terminate an application. If operator shutdown is enabled, the runtime system begins application execution by issuing a WTOR (write-to-operator with reply) macro. At any time during application execution, the operator can issue a SHUTDOWN command, as described in Runtime Considerations (see page 79).

The SHUTDOWN command causes the runtime system to terminate the application with an optional dump. If shutdown is disabled, the operator can terminate the application only by abnormally terminating the runtime system. The default specification is DISABLED.

MAXIMUM LINKS IS *maximum-links*

Specifies the maximum number of dialog levels that can be established by each CA ADS Batch application thread. *Maximum-links* is an integer in the range 0 through 32,767. The default is determined at system generation; the system generation default is 10.

COMMIT WHEN FILES OPEN CAUSES NOACTION/WARNING/ABEND

Specifies the action to be taken when a database commit is to be performed before all files used in the application have been closed, as follows:

- **NOACTION** specifies that no action is taken.
- **WARNING** specifies that a warning message is sent to the log file.
- **ABEND** (default) specifies that the application is abended.

LOG

Specify defaults for the log file. All LOG statement parameters are optional. The parameters are as follows.

FILE1 IS ADSLOGA/*run-time-label*

Specifies the runtime label (z/OS ddname, z/VSE filename) of the primary log file.

COUNT1 IS *file1-maximum-records*

Specifies the number of log records that are written to the primary log file before the file is considered full.

If *file1-maximum-records* is reached at runtime, the runtime system switches to the secondary log file or, if no secondary log file is allocated, wraps around to the beginning of the primary log file. If archiving is requested, the runtime system archives the primary log file to tape.

If 0 is specified, no predefined limit is placed on the number of records written to the primary log file. If space for a disk log file is exceeded at runtime, the runtime system abnormally terminates the application.

File1-maximum-records is an integer in the range 1 through 99999; the default is 0.

FILE2 IS *run-time label*

Specifies the runtime label of the secondary log file.

COUNT2 IS *file2-maximum-records*

Specifies the number of log records that are written to the secondary log file before the file is considered full.

If *file2-maximum-records* is reached at runtime, the runtime system switches back to the primary log file. If archiving is requested, the runtime system archives the secondary log file to tape.

If 0 is specified, no predefined limit is placed on the number of records written to the secondary log file. If space for a disk log file is exceeded at runtime, the runtime system abnormally terminates the application.

File2-maximum-records is an integer in the range 1 through 99999; the default is 0.

ARCHIVE/NOARCHIVE

(z/OS only) specifies whether log file archiving is to be performed at runtime when a log file is full. For more information about log file archiving, see Log Files (see page 21). The default is NOARCHIVE.

COMPRESS/NOCOMPRESS

Specifies whether log records are compressed in the log file to save space at runtime. If neither COMPRESS nor NOCOMPRESS is specified, the runtime system uses the z/OS and z/VSE device-type assignment to determine whether to compress the records:

- In **z/OS**, records are not compressed if the assignment is SYSOUT; otherwise, records are compressed.
- In **z/VSE**, records are not compressed if the device type is PRINTER (as specified using ADSOBSYS or at runtime); otherwise, records are compressed.

PREFIX/NOPREFIX

Specifies whether a prefix is to precede each log record. Prefixes are required by the print log utility. If neither PREFIX nor NOPREFIX is specified, the runtime system uses the z/OS and z/VSE device-type assignment to determine whether to include a prefix:

- In **z/OS** , a prefix is not included if the assignment is SYSOUT; otherwise, a prefix is included.
- In **z/VSE**, a prefix is not included if the device type is PRINTER (as specified using ADSOBSYS or at runtime); otherwise, a prefix is included.

SUBSCHEMA-LR-CTRL SIZE IS *lrc-block-size*

Specifies the space, in bytes, reserved for the logical record request WHERE clause (PXE), which is passed internally in the LRC block. The default is 512.

The larger the WHERE clause, the more space is required for the PXE. The default of 512 is large enough to include approximately 32 operators, operands, and literals.

The SUBSCHEMA-LR-CTRL SIZE parameter enables you to override the default allocation, usually to enlarge it. If the allocation is insufficient for a logical record command, the dialog abends with a minor code of 69.

Lrc-block-size can be any value from 1 through 32767.

DC PACKET SIZE IS *maximum-data*

Specifies the maximum size, in bytes, for the data stream in a QUEUE or WRITE PRINTER command. The default is 952.

Maximum-data does not include the 72 bytes that the run-time system allocates in addition to hold system and packet-header information in the data stream. Therefore, if you accept the default of 952, 1024 bytes will be allocated at run time for the entire data stream.

The DC PACKET SIZE parameter enables you to override the default allocation, usually to enlarge it. If the allocation is insufficient for a QUEUE or WRITE PRINTER command, the dialog abends with a minor code of 19.

Maximum-data can be any value from 1 through 32767.

Appendix B: z/VSE File Characteristics Program

This section contains the following topics:

Overview

The z/VSE file characteristics program (IDMSFILE) enables z/VSE users to specify file characteristics at runtime in the form of control statements, providing z/VSE users with the flexibility enjoyed by z/OS users.

In the CA ADS Batch environment, you use IDMSFILE in conjunction with the CA ADS Batch runtime system (ADSBATCH) and the CA ADS Batch print log utility (ADSOBPLG). For example, to execute an CA ADS Batch application, you submit the JCL presented in Runtime Considerations (see page 79). Note that the EXEC statement executes the program IDMSFILE, and not ADSBATCH.

Control statements you supply as input to IDMSFILE describe the input, output, suspense, and log files used in the application. The last control statement specifies the program you want to execute; in this example, ADSBATCH.

Types of Control Statements

IDMSFILE allows you to specify four types of control statements:

- **SET OPTIONS—**Used to define the beginning and ending columns for the control statements that follow.
- **DEFINE CHARACTERISTICS—**Used to describe file characteristics. File characteristics include block size, record length, record format, tape label format, device type, data set organization, logical unit assignment, and printer control character format.
- **ACCEPT CHARACTERISTICS—**Used to specify a file that contains IDMSFILE control statements. IDMSFILE processes the control statements as regular input, then continues processing the explicitly coded control statements.
- **RUN PROGRAM—**Used to specify the program to be executed (for example, ADSBATCH) and to supply any input parameters that would, without IDMSFILE, be coded in the EXEC statement for the program.

Example 1

The example below illustrates the JCL and control statements you could provide to execute an CA ADS Batch application that accesses a data set whose filename is PAYROLL and a log file whose filename is ADSLOGA:

```
// DLBL SYSCTL, 'idms.sysctl',0,SD
// DLBL       ciln,'idms.cilib'
// EXTENT     ,nnnnnn
// ASSGN      CL,SEARCH=(ciln),TEMP
// DLBL       ADSLOGA,'log.file'
// EXTENT     sysnnn,VOLSER,,,start-track,end-track
// ASSGN      sysnnn,DISK,VOL=VOLSER,SHR
// DLBL       PAYROLL,'payroll.file'
// EXTENT     sysnnn,VOLSER,,,start-track,end-track
// ASSGN      sysnnn,DISK,VOL=VOLSER,SHR
// DLBL       REPFILE,'report.file'
// EXTENT     sysnnn,VOLSER,,,start-track,end-track
// ASSGN      sysnnn,DISK,VOL=VOLSER,SHR
// EXEC       IDMSFILE,SIZE=(ADSBATCH,128K)
DEFINE CHARACTERISTICS FOR FILE ADSLOGA
   BLKSIZE=4096
   LRECL=4092
   RECFM=VARUNB
   DEVTYPE=DISK ASSIGN TO SYS126.
DEFINE CHARACTERISTICS FOR FILE PAYROLL
   BLKSIZE=80
   LRECL=80
   RECFM=FIXUNB
   DEVTYPE=DISK ASSIGN TO SYS122.
DEFINE CHARACTERISTICS FOR FILE REPFILE
   BLKSIZE=132
   LRECL=132
   RECFM=FIXUNB
   DEVTYPE=DISK ASSIGN TO SYS125.
RUN PROGRAM ADSBATCH PARM='DIALOG1,,runtime-parm'.
/*
DICTNAME=APPLDICT.
LOG FILE1=ADSLOGA.
DIALOG DIALOG1 OUTPUT FILENAME IS PAYROL
DIALOG STATISTICS ON FOR(DIALOG1,DIALOG2).
/*
/*
```

Note that the control statements for the program to be executed (in this case, ADSBATCH) immediately follow the RUN PROGRAM statement. Also note that the IDMSFILE control statements allow you to specify file characteristics with JCL-like parameters. Alternatively, you can specify characteristics using COBOL- or IDD-like syntax or using greatly abbreviated syntax.

Syntax rules for the four IDMSFILE control statements are described separately below. JCL for ADSBATCH is provided in Runtime Considerations (see page 79). JCL for ADSOBPLG is provided in CA ADS Batch Print Log Utility (see page 133).

Example 2

The example below illustrates the JCL and control statements you could provide to execute an CA ADS Batch application where ADSLOGA goes to the printer and input is from cards. It also shows the definition of the suspense, archive, and report files:

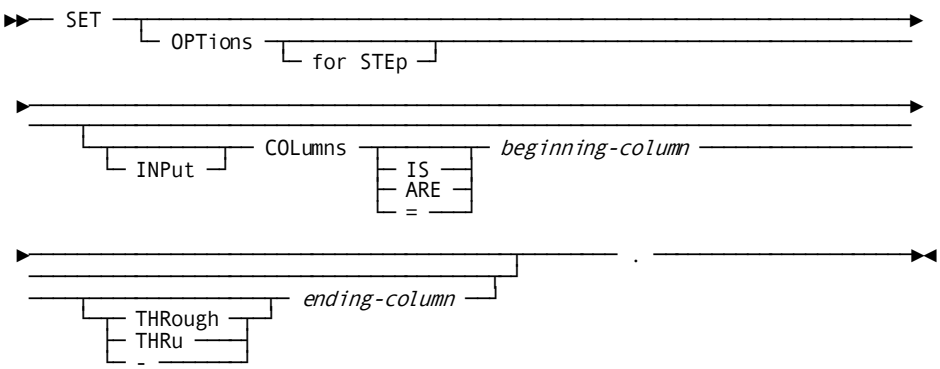
```
// DLBL       ARCFILE,'log.file'
// EXTENT     sysnnn,VOLSER,,,start-track,end-track
// ASSGN      sysnnn,DISK,VOL=VOLSER,SHR
// DLBL       SUSFILE,'payroll.file'
// EXTENT     sysnnn,VOLSER,,,start-track,end-track
// ASSGN      sysnnn,DISK,VOL=VOLSER,SHR
// DLBL       REPFILE,'report.file'
// EXTENT     sysnnn,VOLSER,,,start-track,end-track
// ASSGN      sysnnn,DISK,VOL=VOLSER,SHR
@SYSFILES
@SYSCTL
// DLBL SYSIDMS,'#SYSIPT'
// EXEC    IDMSFILE,SIZE=(AUTO,128K)
@SYSPARMS
/*
DEFINE CHARACTERISTICS FOR FILE ADSLOGA
  LRECL=133 RECFM=FIXUNB
  CONTROL CHARACTERS ARE ASA
  DEVTYPE=PRINTER ASSIGN TO SYSLST.
DEFINE CHARACTERISTICS FOR FILE INFILE1
```

```
  LRECL=80         RECFM=FIXUNB
  SYSNO=SYSIPT
DEFINE CHARACTERISTICS FOR FILE SUSFILE
  LRECL=80 BLKSIZE=4000 RECFM=FIXBLK
  DEVTYPE=DISK ASSIGN TO SYS124.
DEFINE CHARACTERISTICS FOR FILE ARCFILE
  LRECL=128 BLKSIZE=2564 RECFM=VARBLK
  DEVTYPE=DISK ASSIGN TO SYS123.
DEFINE CHARACTERISTICS FOR FILE REPFILE
  LRECL=132 BLKSIZE=132  RECFM=FIXUNB
  CONTROL CHARACTERS ARE ASA
  DEVTYPE=PRINTER ASSIGN TO SYSLST.
RUN PROGRAM ADSBATCH.
/*
DICTNAME=APPLDICT.
LOG FILE1 = ADSLOGA NOARCHIVE.
DIALOG ARCDM INP FIL INFILE1.
DIALOG ARCDM SUS FIL SUSFILE.
DIALOG ARCD2 OUT FIL ARCFILE.
DIALOG ARCD3 OUT FIL ARCFILE.
DIALOG ARCD5 OUT FIL REPFILE .
ENTRY DIALOG ARCDM.
/*
0028
0029
0458
0459
0450
/*
```

SET OPTIONS Statement

The SET OPTIONS statement enables you to define the beginning and ending columns for the IDMSFILE control statements that follow. By default, the beginning column is 1 and the ending column is 72. You can code multiple SET OPTIONS statements to set different ranges, as required.

Syntax for the SET OPTIONS statement is shown below:



beginning-col

Specifies the column at which IDMSFILE should begin scanning in the control statement input records that follow. Characters in columns preceding *beginning-col* are ignored.

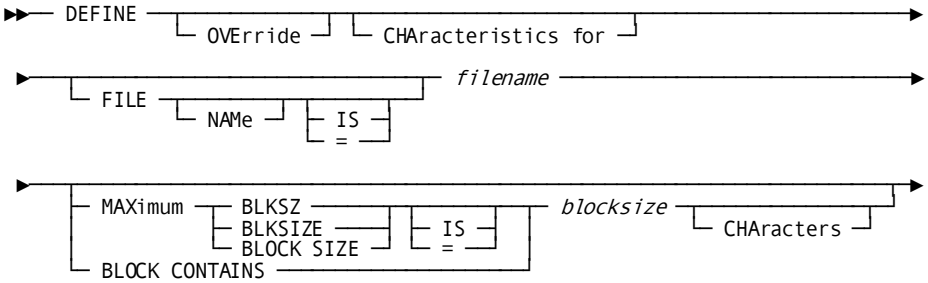
ending-col

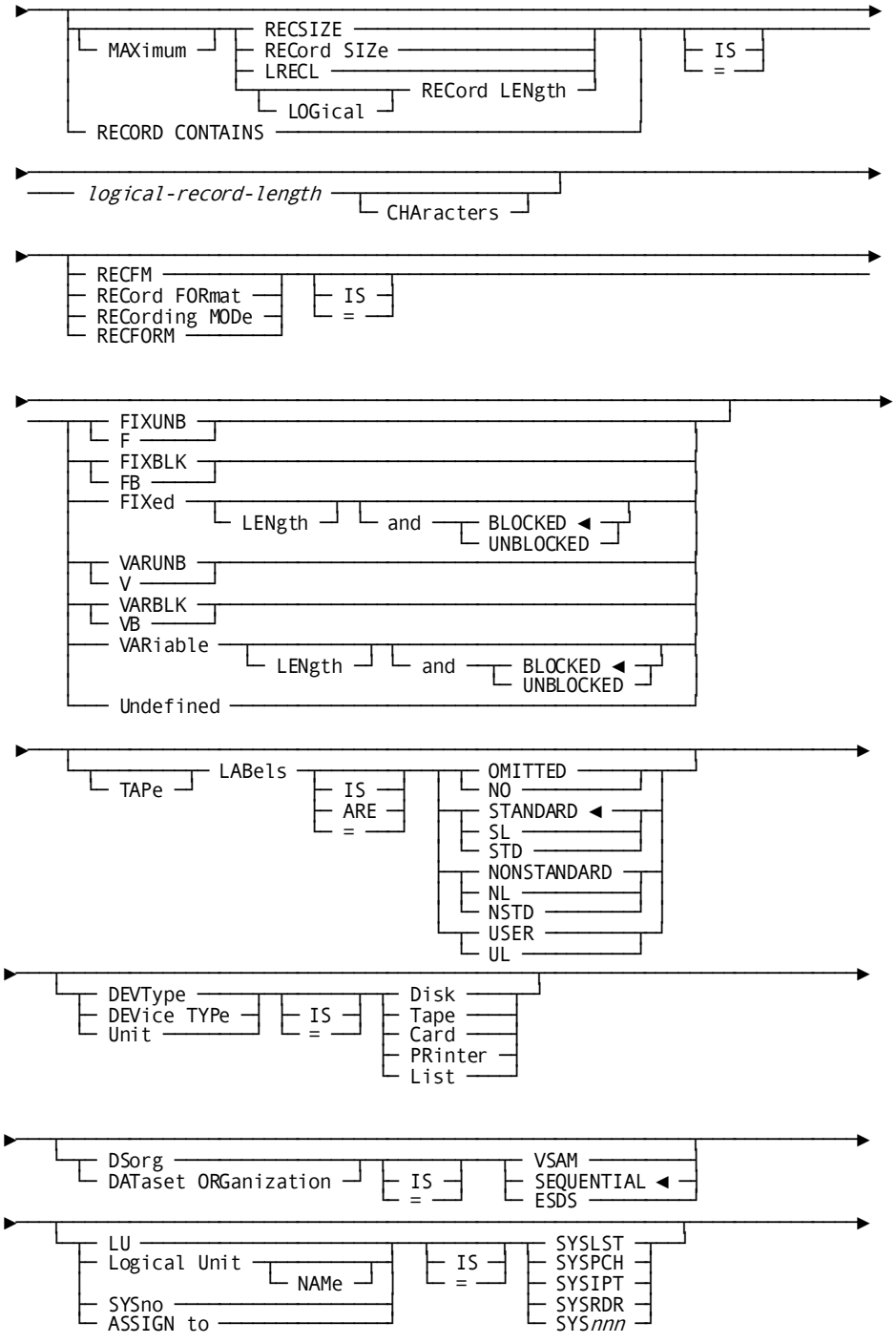
Specifies the last column that IDMSFILE should scan in the control statement input records that follow. Characters in columns following *ending-col-n* are ignored.

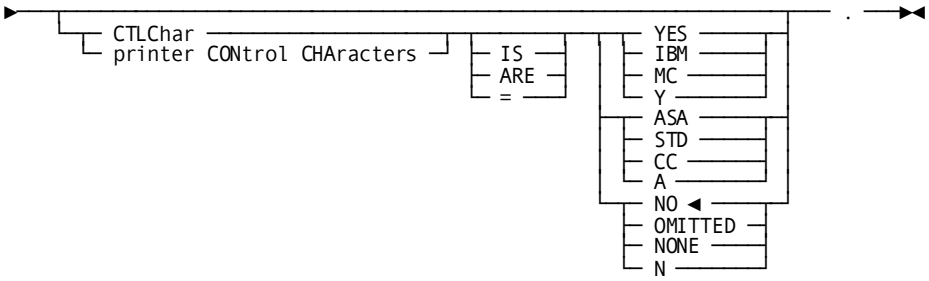
DEFINE CHARACTERISTICS Statement

The DEFINE CHARACTERISTICS statement enables you to describe the characteristics of a file. File characteristics include block size, record length, record format, tape label format, device type, data set organization, logical unit assignment, and printer control character format. Multiple DEFINE CHARACTERISTICS statements can be coded to describe several files.

Syntax for the DEFINE CHARACTERISTICS statement is shown below:




MAXimum
RECSIZE
RECord SIZe
LRECL
RECord LENgth
LOGical
RECORD CONTAINS
IS
=
logical-record-length
CHAracters
RECFM
RECord FORmat
RECording MODe
RECFORM
IS
=
FIXUNB
F
FIXBLK
FB
FIXed
LENgth
and
BLOCKED ◄
UNBLOCKED
VARUNB
V
VARBLK
VB
VARiable
LENgth
and
BLOCKED ◄
UNBLOCKED
Undefined
TAPe
LABels
IS
ARE
=
OMITTED
NO
STANDARD ◄
SL
STD
NONSTANDARD
NL
NSTD
USER
UL
DEVType
DEVice TYPe
Unit
IS
=
Disk
Tape
Card
PRinter
List
DSorg
DATaset ORGanization
IS
=
VSAM
SEQUENTIAL ◄
ESDS
LU
Logical Unit
NAMe
SYSno
ASSIGN to
IS
=
SYSLST
SYSPCH
SYSIPT
SYSRDR
SYSnnn



Note that the syntax allows you to specify characteristics in several formats, such as z/OS JCL, COBOL, and DDDL. Additionally, the syntax allows you to code abbreviated keywords.

DEFINE CHARACTERISTICS FOR FILE *filename*

Specifies the filename of the file being described.

OVERRIDE

Specifies that the file characteristics in the current DEFINE CHARACTERISTICS statement are to override the default characteristics of *filename*. OVERRIDE is valid only when the specified file is also named in a subsequent ACCEPT CHARACTERISTICS statement. The default characteristics of a file in an ACCEPT CHARACTERISTICS statement are:

- Logical record length of 80
- Block size of 80
- Record format of fixed

Specify OVERRIDE if the DEFINE CHARACTERISTICS statement is overriding any of these characteristics.

MAXIMUM BLOCK SIZE IS *blocksize* CHARACTERS

Specifies the block size of the file.

Note that BLKSZ, BLKSIZE, and BLOCK SIZE are synonymous. BLOCK CONTAINS is synonymous with MAXIMUM BLOCK SIZE IS.

MAXIMUM LOGICAL RECORD LENGTH IS *lrecl* CHARACTERS

Specifies the logical record length of the file.

Note that RECSIZE, RECORD SIZE, LRECL, and LOGICAL RECORD LENGTH are synonymous. RECORD CONTAINS is synonymous with MAXIMUM LOGICAL RECORD LENGTH.

RECFM IS FIXUNB/FIXBLK/FIXED/VARUNB/VARBLK/VARIABLE/UNDEFINED

Specifies the record format of the file, as follows:

- **FIXUNB** specifies that the record format is fixed unblocked.
- **FIXBLK** specifies that the record format is fixed blocked.
- **FIXED LENGTH AND BLOCKED/UNBLOCKED** specifies that the record format is fixed blocked or unblocked. The default is BLOCKED.
- **VARUNB** specifies that the record format is variable unblocked.
- **VARBLK** specifies that the record format is variable blocked.
- **VARIABLE LENGTH AND BLOCKED/UNBLOCKED** specifies that the record format is variable blocked or unblocked. The default is BLOCKED.
- **UNDEFINED** specifies that the record format is undefined.

TAPE LABELS ARE OMITTED/STANDARD/NONSTANDARD/USER

Specifies the tape label format of the file. The default is STANDARD.

DEVTYPE IS DISK/TAPE/CARD/PRINTER/LIST

Specifies the device type of the file.

DSORG IS VSAM/SEQUENTIAL

Specifies the data set organization of the file. The default is SEQUENTIAL.

LU IS SYSLST/SYSPCH/SYSIPT/SYSRDR/SYS *nnn*

Specifies the logical unit assignment of the file.

CTLCHAR IS YES/ASA/NO

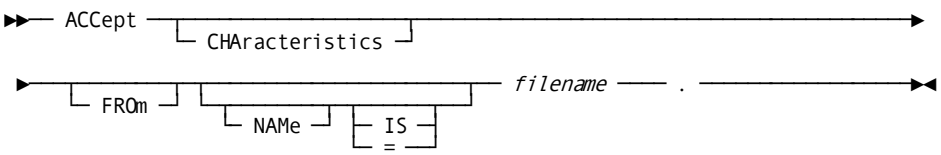
Specifies the printer control character format of a file assigned to a printer. The default is NO.

ACCEPT CHARACTERISTICS statement

The ACCEPT CHARACTERISTICS statement enables you to specify a file in which IDMSFILE control statements are stored. IDMSFILE processes the control statements in the specified file as regular input, then continues to process any explicitly coded control statements that follow the ACCEPT CHARACTERISTICS statement. Embedded ACCEPT CHARACTERISTICS statements are allowed.

Note that the object file can contain only IDMSFILE control statements. Control statements for the program to be executed, such as ADSBATCH, must be coded explicitly following the last IDMSFILE control statement. The object file can, however, contain a RUN PROGRAM statement.

Syntax for the ACCEPT CHARACTERISTICS statement follows:



filename

The name of the file that contains IDMSFILE control statements. *Filename* must be described by a DEFINE CHARACTERISTICS statement coded earlier.

ACCEPT CHARACTERISTICS uses default file characteristics for *filename*, as follows:

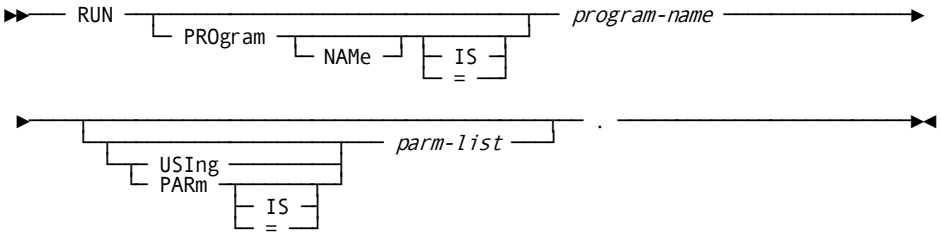
- Logical record length of 80
- Block size of 80
- Record format of fixed

To override any of these characteristics, specify OVERRIDE in the DEFINE CHARACTERISTICS statement that describes *filename*.

RUN PROGRAM Statement

The RUN PROGRAM statement enables you to specify the program to be executed, such as ADSBATCH or ADSOBPLG, and to specify any parameters that would, without IDMSFILE, be included in an EXEC statement for the desired program. RUN PROGRAM is the last IDMSFILE control statement you code, and is followed by control statements you code as input to the program specified in the RUN PROGRAM statement.

Syntax for the RUN PROGRAM statement is shown below:



RUN PROGRAM *program-name*

Specifies the program to be executed, such as ADSBATCH or ADSOBPLG.

USING *parm-list*

Specifies a parameter list that would, without IDMSFILE, be included in an EXEC statement for *program-name*. *Parm-list* must be enclosed in quotation marks if the list contains any delimiters, such as blanks, commas, semicolons, or periods.

Appendix C: CA ADS Batch Print Log Utility

This section contains the following topics:

Overview

The CA ADS Batch print log utility (ADSOBPLG) enables you to print formatted reports of selected information from a log file created during the execution of an CA ADS Batch application. The log file must contain an identifying prefix for each record; otherwise, the print log utility can print only an unformatted listing of all records in the file.

The print log utility also enables you to extract selected records from the log file without formatting. With this feature, you can, for example, extract all edit error messages from the log file in order to merge them with matching error records in the suspense file.

This appendix provides the following information about ADSOBPLG:

- A description of the print log utility
- The DSECT for the log file prefix
- Syntax rules for control statements
- JCL

How to Use the Print Log Utility

In a single execution of the print log utility, you can print any or all of the following types of information contained in the log file:

- **User**—Informational messages issued by WRITE TO LOG and CONTINUE commands, and by WRITE TRANSACTION commands that perform a mapout to the dialog's output file. The message is specified in the MESSAGE TEXT parameter of the command.
- **Debug**—Debugging information issued by SNAP commands and by the CA ADS Batch trace facility.

- **Edit error**—Error messages issued when an input record is written to the suspense file. The runtime system writes one record consisting of the associated suspense file record number and the first 32 bytes of the input record, followed by one record for each applicable error message.

  In addition, if the input record was written to the suspense file by a WRITE TRANSACTION command, any message specified in the MESSAGE TEXT parameter of the command is also written to the log file as an edit error message.

- **Operator**—Operator informational messages issued by WRITE TO OPERATOR commands. All messages sent to the operator are also sent to the log.

- **Abend**—Abnormal termination messages issued by ABORT commands and by the runtime system when an application abends. Abend messages include snap dumps.

  Note: If an abend occurs during a mapping operation, the snap dump goes instead to the z/OS data set associated with the ddname (linkname) IDMSSNAP or, in VSE, to SYSLST (as described under JCL (see page 110)). An example of a mapping abend is when an input record that contains errors cannot be written to a suspense file because the MAXIMUM ERRORS for the dialog, which is specified at system generation or at runtime, has been exceeded.

- **Statistics**—Statistics records written by the runtime system when dialog statistics are being collected for the application.

Selecting Log File Records

The print log utility enables you to select log file records by:

- **Dialog**—You can specify that only log file records created by one or more specified dialogs are included in the report.

- **User id**—You can specify that only log file records created by specific users are included in the report. The user is determined at runtime by the current value of the $USER ($REQUESTOR) system-supplied data field; $USER ($REQUESTOR) can be initialized at runtime through an assignment command.

- **Beginning time and date**—You can specify that only log file records created on or after a certain time and date are included in the report.

- **Ending time and date**—You can specify that only log file records created on or before a certain time and date are included in the report.

You can specify any or all of the above selection criteria. Log file records must meet all criteria you specify in order to be included in the report.

DSECT for the Log File Prefix

Each record written to the log file includes an optional prefix that specifies the following information:

- The date and time the record was written
- The name of the dialog that wrote the record
- The type of record written
- The application user id

This information is used by the print log utility to select log records and format them for output. The information may also be required by reports that you create.

DSECT

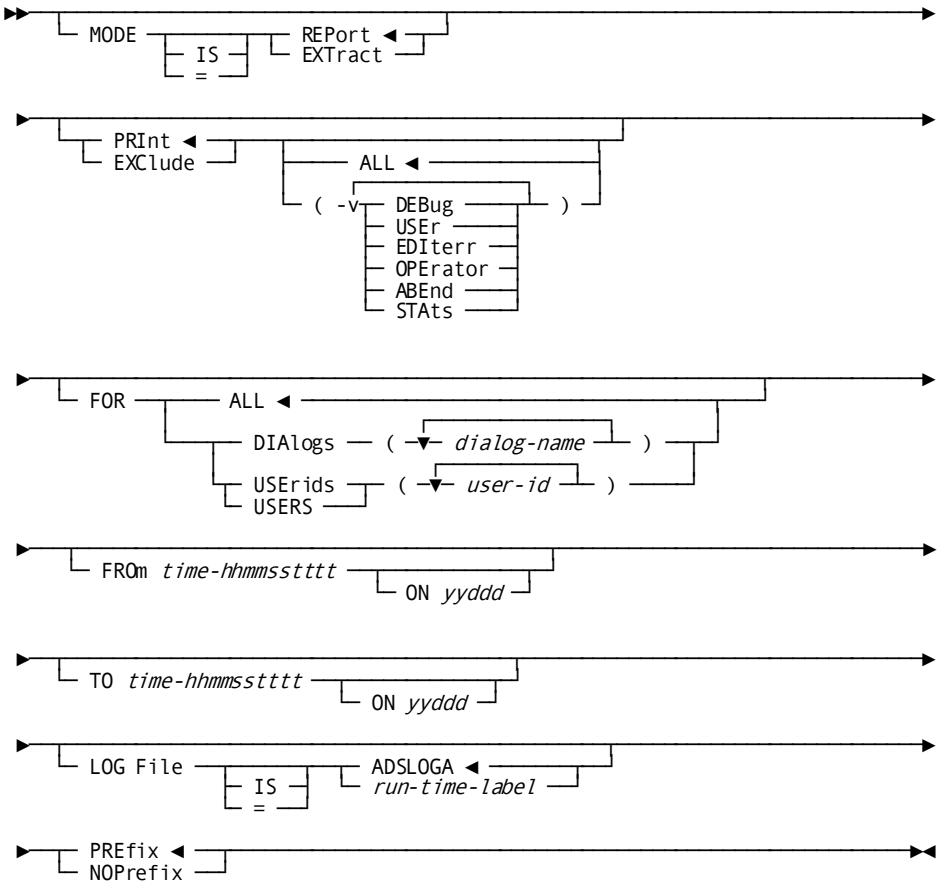
The DSECT for the log file prefix, #BLPDS, is shown below:

```
*********************************************************************** 00001000
***                                                                 *** 00002000
*** BLP:   BATCH LOG PREFIX DSECT                                    *** 00003000
***                                                                 *** 00004000
*********************************************************************** 00005000
BLP      DSECT                          10:41:34 mm/dd/yy            00006000
BLPDTK   DS    0CL8               DATE/TIME (KEY)                    00007000
BLPDATE  DS    CL4                DATE (00YYDDDC)                    00008000
BLPTIME  DS    CL4                TIME (1/10000 SEC)                 00009000
BLPPGNAM DS    CL8                PROGRAM/DIALOG NAME                00010000
BLPTYPE  DS    X                  RECORD TYPE                        00011000
         SPACE                                                       00012000
*   THE FOLLOWING ARE VALID VALUES FOR BLPTYPE                       00013000
*                                                                    00014000
BLPDBUG  EQU   1                  DEBUG                              00015000
BLPUSR   EQU   2                  USER MESSAGE                       00016000
BLPSPNSE EQU   3                  EDIT ERROR RECORD                  00017000
BLPWTO   EQU   4                  WRITE TO OPERATOR MESSAGE          00018000
BLPABND  EQU   5                  ABEND DUMP TEXT                    00019000
BLPSTAT  EQU   6                  STATISTICS                         00020000
BLPHITYP EQU   6                  HIGHEST VALID TYPE                 00021000
*                                                                    00022000
BLPRCMP  #FLAG X'80'              IF ON, LOG RECORD IS COMPRESSED    00023000
BLPFLG1  DS    X                  FLAG BYTE                          00024000
BLPRLEN  DS    H                  LEN OF LOG REC THAT FOLLOWS PREFIX 00025000
BLPULEN  DS    X                  LENGTH OF USER ID                  00026000
         DS    XL3                RESERVED                           00027000
BLPLEN   EQU   *-BLP              LENGTH OF FIXED PORTION OF PREFIX  00028000
BLPUSER  DS    0X                 START OF USER ID                   00029000
         EJECT                                                       00030000
```

Control Statements

The print log utility's control statements enable you to specify the type of reports you want, the criteria to be applied in selecting log file records, and the information about the log file.

Syntax for the control statements is shown below:



MODE IS REPORT/EXTRACT

Specifies whether the output is formatted:

- **REPORT** (default) specifies that the output is formatted with page headers and carriage control characters.
- **EXTRACT** specifies that the output is unformatted. The selected records are simply extracted from the log file; each record includes the log file prefix.

  The EXTRACT option enables you, for example, to extract all EDITERR records created during the execution of an application. You can then merge these records with the suspense files that the records describe.

PRINT/EXCLUDE

Specifies whether the named record types are included in or excluded from the report. PRINT is the default.

ALL/(DEBUG/USER/EDITERR/OPERATOR/ABEND/STATS)

Specifies the types of records requested, as follows:

- **ALL** (default) specifies all record types.
- **DEBUG** specifies debugging information issued by SNAP commands and by the CA ADS Batch trace facility.
- **USER** specifies messages issued by WRITE TO LOG and CONTINUE commands, and by WRITE TRANSACTION commands that perform a mapout to the terminal operator.
- **EDITERR** specifies error messages issued when an input record is written to the suspense file.
- **OPERATOR** specifies messages issued by WRITE TO OPERATOR commands.
- **ABEND** specifies messages issued by the ABORT command and by the runtime system when an application abends. These messages also include snap dumps of memory produced by abends.
- **STATS** specifies statistics written by the runtime system when dialog statistics are being collected for the application.

FOR ALL/DIALOGS (*dialog-name*)/USERIDS (*user-id*)

Specifies selection criteria by dialog and/or user id, as follows:

- **ALL** (default) specifies that the dialog name and user id are not used as selection criteria.
- **DIALOGS (dialog-name)** specifies that only log file records created by the named dialog or dialogs are included in the report. Multiple dialog names must be separated by commas. If the DIALOGS parameter is not specified, the dialog name is not used as a selection criterion.
- **USERIDS (user-id)** specifies that only log file records created by the named user or users are included in the report. Multiple user ids must be separated by commas. If the USERIDS parameter is not specified, the user id is not used as a selection criterion.

FROM *time-hhmmsstttt*

Specifies that only log file records created on or after the specified time are included in the report. *Time-hhmmsstttt* specifies the time in hours (1 through 24), minutes, seconds, and tenths of seconds. Two digits of the hour must be specified; the remainder is optional. For example, 9:45 a.m. is specified as 0945. To include a date using the ON parameter, you must specify a time; you can specify the beginning of a day as 00. The default is the beginning of the day.

ON *yyddd*

Specifies that only log file records created on or after the specified Julian date are included in the report. The default is the date on which the print log utility is executed.

TO *time-hhmmsstttt*

Specifies that only log file records created on or before the specified time are included in the report. *Time-hhmmsstttt* specifies the time in hours (1 through 24), minutes, seconds, and tenths of seconds. Two digits of the hour must be specified; the remainder is optional. For example, 9:45 a.m. is specified as 0945. To include a date using the ON parameter, you must specify a time; you can specify the end of a day as 24. The default is 24, which is the end of the day.

ON *yyddd*

Specifies that only log file records created on or before the specified Julian date are included in the report. The default is the date on which the print log utility is executed.

LOG FILE IS ADSLOGA/*ddname*

Specifies the z/OS ddname or z/VSE filename of the log file. The default is ADSLOGA.

PREFIX/NOPREFIX

Specifies whether the log file contains a prefix for each record. The default is PREFIX. NOPREFIX must be specified if no prefix is included, so that the print log utility does not attempt to interpret the beginning portion of each log file record as a prefix.

JCL

z/OS JCL Under Central Version

JCL for running the print log utility (ADSOBPLG) is shown below:

ADSOBPLG (z/OS)

```
//ADSOBPLG EXEC PGM=ADSOBPLG,REGION=1024K
//STEPLIB  DD   DSN=idms.dba.loadlib,DISP=SHR
//         DD   DSN=idms.loadlib,DISP=SHR
//sysctl   DD   DSN=idms.sysctl,DISP=SHR
//adslog   DD   DSN=ads.log.file,DISP=SHR
//SYSLST   DD   SYSOUT=A
//SYSIDMS  DD   *
DMCL=dmcl-name
DICTNAME=dictionary-name
Other SYSIDMS parameters, as appropriate
/*
//SYSIPT   DD   *
control statements
/*
//SYSUDUMP DD   SYSOUT=A
```

idms.dba.loadlib	data set name of the load library containing the DMCL and database name table load modules
idms.loadlib	data set name of DC/UCF load library
sysctl	ddname of SYSCTL file
idms.sysctl	data set name of SYSCTL file
adslog	ddname of CA ADS Batch log file
ads.log.file	data set name of CA ADS Batch log file

Local Mode

To execute ADSOBPLG in local mode, the following steps are required:

1. Remove the sysctl DD statement.
2. Add the following statements:

```
//sysjrnl DD   DSN=idms.tapejrnl,DISP=NEW,UNIT=tape
//dictdb  DD   DSN=idms.dictdb,DISP=OLD
//dloddb  DD   DSN=idms.dloddb,DISP=SHR
//dmsgdb  DD   DSN=idms.dmsgdb,DISP=SHR
//dlogdb  DD   DSN=idms.dlogdb,DISP=SHR
additional journal file assignments, as required
```

sysjrnl	ddname of journal file
idms.tapejrnl	data set name of journal file
tape	symbolic device name of journal file
dictdb	ddname of data dictionary
idms.dictdb	data set name of data dictionary
dloddb	ddname of data dictionary load area
idms.dloddb	data set name of data dictionary load area
dmsgdb	ddname of data dictionary message area
idms.dmsgdb	data set name of data dictionary message area
dlogdb	ddname of data dictionary log area
idms.dlogdb	data set name of data dictionary log area

z/VSE JCL Under Central Version ADSOBPLG (z/VSE)

```
// DLBL SYSCTL,'idms.sysctl',1999/365,SD
// DLBL SYSIDMS,'#SYSIPT'
// DLBL     ciln,'idms.cilib'
// EXTENT   ,nnnnnn
// LIBDEF   CL,SEARCH=(ciln),TEMP
// DLBL     adsloga,'log.file'
// EXTENT   sysnnn,nnnnnn
   ASSGN    sysnnn,DISK,VOL=nnnnnn,SHR
// EXEC     IDMSFILE,SIZE=(ADSOBPLG,128K)
SYSIDMS control statements
IDMSFILE control statements
ADSOBPLG control statements
```

idms.sysctl	data set name of SYSCTL file
ciln	filename of core-image library
idms.cilib	file-id of DC/UCF core-image library
nnnnnn	volume serial number
adsloga	filename of the log file
log.file	file-id of the log file
sysnnn	logical unit of the device on which the log file is located
SYSIDMS control statements	control statements used by SYSIDMS to describe physical runtime environments as described in *CA IDMS Common Facilities Guide*.

IDMSFILE control statements	control statements used by IDMSFILE to describe the characteristics of the log file, as described in z/VSE File Characteristics Program (see page 123).
ADSOBPLG control statements	control statements used by ADSOBPLG

Note: A /* card must follow each SYSIPT file.

Local Mode

To execute ADSOBPLG in local mode, the following steps are required:

1. Remove the SYSCTL DLBL card
2. Add the following statements:

```
// DLBL     dictdb,'idms.dictdb',,DA
// EXTENT   sys005,nnnnnn
// ASSGN    sys005,DISK,VOL=nnnnnn,SHR
// DLBL     dloddb,'idms.dloddb',,DA
// EXTENT   sys017,nnnnnn
// ASSGN    sys017,DISK,VOL=nnnnnn,SHR
// DLBL     dmsgdb,'idms.dmsgdb',,DA
// EXTENT   sys016,nnnnnn
// ASSGN    sys016,DISK,VOL=nnnnnn,SHR
// DLBL     dlogdb,'idms.dlogdb',,DA
// EXTENT   sys019,nnnnnn
// ASSGN    sys019,DISK,VOL=nnnnnn,SHR
// TLBL     sysjrnl,'idms.tapejrnl',,nnnnnn,,f
// ASSGN    sys009,TAPE,VOL=nnnnnn
additional journal file assignments, as required
```

dictdb	filename of data dictionary
idms.dictdb	file-id of data dictionary
sys005	logical unit assignment for data dictionary
nnnnnn	volume serial number
dloddb	filename of data dictionary load area
idms.dloddb	file-id of data dictionary load area
sys017	logical unit assignment for data dictionary load area
dmsgdb	filename of data dictionary message area
idms.dmsgdb	file-id of data dictionary message area
sys016	logical unit assignment for data dictionary message area

dlogdb	filename of data dictionary log area
idms.dlogdb	file-id of data dictionary log area
sys019	logical unit assignment for data dictionary log area
sysjrnl	filename of tape journal file
idms.tapejrnl	file-id of tape journal file
f	file number of tape journal file
sys009	logical unit assignment for tape journal file

Local Mode

To execute ADSOBPLG in local mode, the following steps are required:

1. Remove the ADD-FILE-LINK statement for sysctl
2. Add the following statements:

```
/ADD-FILE-LINK L-NAME=dictdb,F-NAME=idms.appldict.dictdb,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=dloddb,F-NAME=idms.appldict.dloddb,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=dcmsg,F-NAME=idms.sysmsg.ddldcmsg,SHARED-UPD=*YES
/ADD-FILE-LINK L-NAME=sysjrnl,F-NAME=idms.tapejrnl
additional journal file assignments, as required
```

dictdb	linkname of data dictionary file
idms.appldict.dictdb	filename of data dictionary file
dloddb	linkname of data dictionary load area file
idms.appldict.dloddb	filename of data dictionary load area file
dcmsg	linkname of data dictionary message area file
idms.sysmsg.ddldcmsg	filename of data dictionary message area file
sysjrnl	linkname of tape journal file
idms.tapejrnl	filename of tape journal file

Appendix D: CA ADS Batch Sample Applications

This section contains the following topics:

Overview

This appendix implements the following CA ADS Batch sample applications:

- **Employee-record archive application**—Archives selected employee records and their associated coverage records from an CA IDMS/DB database to an output file
- **Employee-record restore application**—Restores selected employee records and their associated coverage records from the archive file to the database
- **Employee-record report application**—Produces a report of employees by department

Subschema

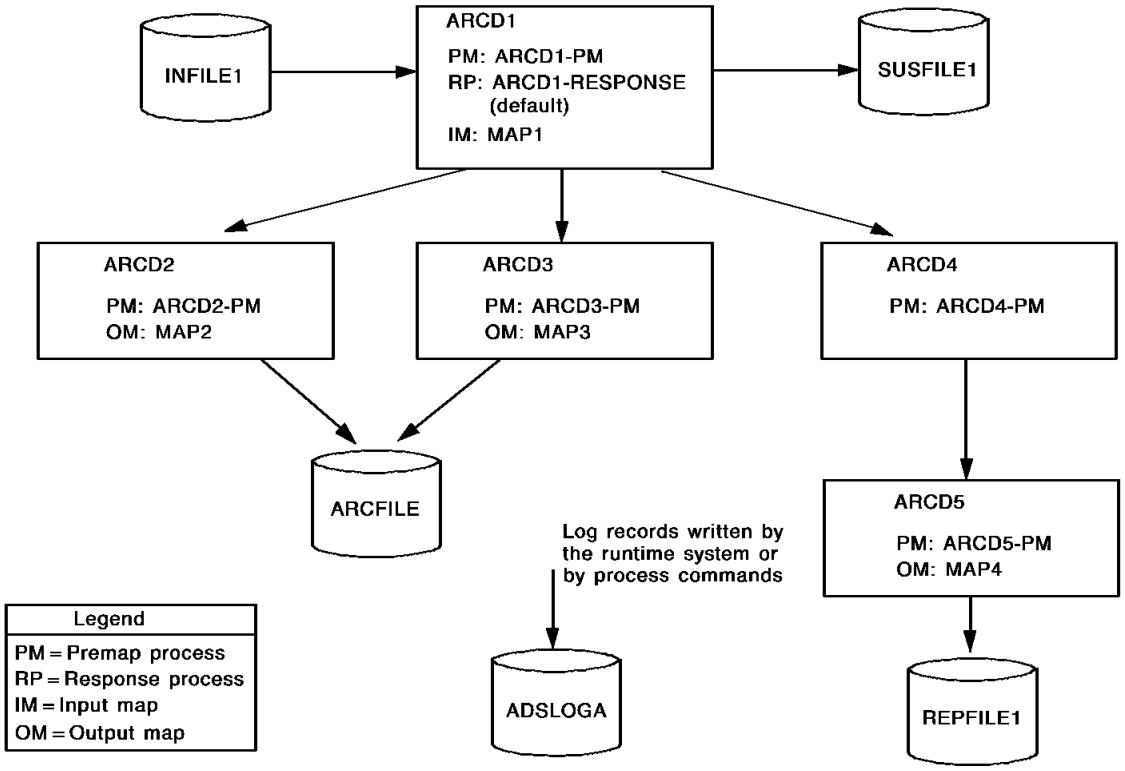
All applications access the **employee subschema**. The subschema includes the following database records.

Database record	Description
EMPLOYEE	Stores employee records. The CALC key for EMPLOYEE is EMP-ID-0415.
COVERAGE	Stores insurance coverage records for each employee. Several COVERAGE records can be associated with an EMPLOYEE record. The relationship is described by the EMP-COVERAGE set.
DEPARTMENT	Stores information about the company's departments. The CALC key for DEPARTMENT is DEPT-ID-0410. Each employee belongs to a department. The relationship is described by the DEPT-EMPLOYEE set.

Database record	Description
OFFICE	Stores information about the company's offices. The CALC key for OFFICE is OFFICE-CODE-0450. Each employee belongs to an office. The relationship is described by the OFFICE-EMPLOYEE set.

Employee-record Archive Application

The employee-record archive application writes selected employee records and their associated coverage records from an CA IDMS/DB database to an output file. The following diagram shows the file access, components, and flow of control for the application.



The files accessed and the dialogs used by the application are as follows.

Files

Name	Type	Description
INFILE1	Input file	Contains the ids of employee records to be archived

Name	Type	Description
SUSFILE1	Suspense file	Contains records from INFILE1 found in error
ARCFILE	Output file	Contains archived employee records and their associated coverage records
REPFILE1	Output file	Contains a transaction summary report for the application
ADSLOGA	Log file	Contains informational and error messages produced by the application

Dialogs

Name	Description
ARCD1	Reads input records and acts as a mainline routine, passing control to dialogs ARCD2, ARCD3, and ARCD4, as required by the application; writes erroneous input records to the suspense file
ARCD2	Writes an employee record to the archive file, then returns control to ARCD1
ARCD3	Writes the coverage records associated with an archived employee record to the archive file, then returns control to ARCD1
ARCD4	Determines the transaction report lines to be written to the report output file; for each line to be written, passes control to ARCD5; returns control to ARCD1 after all lines for the transaction have been written
ARCD5	Writes a report line to the report output file, then returns control to ARCD4

Steps to create the application

To create the application, you perform the following steps:

1. Describe the input and output files in the data dictionary.
2. Describe the layouts of file, map, and work records in the data dictionary.
3. Define the file maps that associate file records with variable storage.
4. Define the process modules for the application's dialogs.
5. Define the dialogs.

These steps are described below, followed by a discussion of executing the application.

Step 1: Describe the Files in the Data Dictionary

Each input and output file (excluding suspense and log files) must be described in the data dictionary as a file entity. You can define the files by using the IDD DDDL compiler or the IDD menu facility.

You can describe the file entities without specifying file characteristics. At runtime, under z/OS, the runtime system uses the file characteristics specified in the JCL or data set labels. Under VSE, the runtime system uses the characteristics specified by IDMSFILE control statements.

The following IDD DDDL statements add file entity descriptions for the input, archive, and report files:

```
ADD FILE IDD-INFILE1.
ADD FILE IDD-ARCFILE.
ADD FILE IDD-REPFILE1.
```

Alternatively, since you are not including file characteristics as part of the file descriptions, you can use one generic file entity to represent all three files:

```
ADD FILE IDD-GENERIC-FILE.
```

Step 2: Describe the Records in the Data Dictionary

All file, map, and work records used in the application must be described in the data dictionary. The employee-record archive application uses the following records.

Record	Description
WORK-RECORD1	Contains miscellaneous variable fields required by the application
INPUT-RECORD1	Describes the layout of the input file
ARCHIVE-RECORD-EMP	Describes the layout of archived employee records in the archive file
ARCHIVE-RECORD-COV	Describes the layout of archived coverage records in the archive file
REPORT-RECORD	Describes the layout of the report file
REPORT-HDR1 REPORT-HDR2 REPORT-DTL	Describe the heading and detail lines for the report

You can define records by using the IDD DDDL compiler or the IDD menu facility. The record definitions are illustrated on the following pages.

WORK-RECORD1

```
WORK-RECORD1.
   03  WORK-DATE-YYMMDD.
       05  WORK-YY1             PICTURE IS   9(2).
       05  WORK-MM1             PICTURE IS   9(2).
       05  WORK-DD1             PICTURE IS   9(2).
   03  WORK-DATE-MMDDYY.
       05  WORK-MM2             PICTURE IS   9(2).
       05  FILLER               PICTURE IS   X       VALUE IS '/'.
       05  WORK-DD2             PICTURE IS   9(2).
       05  FILLER               PICTURE IS   X       VALUE IS '/'.
       05  WORK-YY2             PICTURE IS   9(2).
   03  WORK-LINE-CTR            PICTURE IS   99.
   03  WORK-PAGE-CTR            PICTURE IS   999.
   03  WORK-STATUS              PICTURE IS   X(10).
   03  WORK-ARC-ID              PICTURE IS   9(4).
   03  WORK-ARCFILE-STATUS      PICTURE IS   X(3).
```

INPUT-RECORD1

```
INPUT-RECORD1.
   03  INPUT-ID                 PICTURE IS   9(4).

Include within file IDD-INFILE1.
```

ARCHIVE-RECORD- EMP

```
ARCHIVE-RECORD-EMP
   03  ARCHIVE-TYPE             PICTURE IS   X.
   03  ARCHIVE-DEPT-ID          PICTURE IS   9(4).
   03  ARCHIVE-OFFICE-CODE      PICTURE IS   XXX.
   03  ARCHIVE-EMPLOYEE-RECORD  PICTURE IS   X(116).

Include within file IDD-ARCFILE.
```

ARCHIVE-RECORD- COV

```
ARCHIVE-RECORD-COV.
   03  ARCHIVE-TYPE             PICTURE IS   X.
   03  ARCHIVE-COVERAGE-RECORD  PICTURE IS   X(16).

Include within file IDD-ARCFILE.
```

REPORT-RECORD

```
REPORT-RECORD.
   03  REPORT-CC              PICTURE IS   X.
   03  REPORT-LINE            PICTURE IS   X(131).

Include within file IDD-REPFILE1.
```

REPORT-HDR1

```
REPORT-HDR1.
   03  FILLER                 PICTURE IS   X(15)    VALUE IS SPACES.
   03  FILLER                 PICTURE IS   X(27)
                              VALUE IS 'TRANSACTION REPORT   DATE:'.
   03  REPORT-DATE            PICTURE IS   X(8).
   03  FILLER                 PICTURE IS   X(8)     VALUE IS '  PAGE:'.
   03  REPORT-PAGE            PICTURE IS   999.
```

REPORT-HDR2

```
REPORT-HDR2.
   03  FILLER                 PICTURE IS   X(15)    VALUE IS 'EMPLOYEE ID'.
   03  FILLER                 PICTURE IS   X(28)    VALUE IS 'NAME'.
   03  FILLER                 PICTURE IS   X(6)     VALUE IS 'STATUS'.
```

REPORT-DTL

```
REPORT-DTL.
   03  FILLER                 PICTURE IS   X(4)     VALUE IS SPACES.
   03  REPORT-ID              PICTURE IS   9(4).
   03  FILLER                 PICTURE IS   X(7)     VALUE IS SPACES.
   03  REPORT-LNAME           PICTURE IS   X(15).
   03  FILLER                 PICTURE IS   X        VALUE IS SPACES.
   03  REPORT-FNAME           PICTURE IS   X(10).
   03  FILLER                 PICTURE IS   X(2)     VALUE IS SPACES.
   03  REPORT-STATUS          PICTURE IS   X(30).
```

Step 3: Define the File Maps

You define a file map for each file record layout used in the application. The file maps in this application are described below.

Map	Description
MAP1	Associates record INPUT-RECORD1 with variable storage
MAP2	Associates record ARCHIVE-RECORD-EMP with variable storage
MAP3	Associates record ARCHIVE-RECORD-COV with variable storage

Map	Description
MAP4	Associates record REPORT-RECORD with variable storage

You define maps by using the CA IDMS/DC mapping facility. The file map definitions are illustrated below.

MAP1

```
Internal records:       None

External file:          INFILE1
External record:        INPUT-RECORD1

External field                  Internal field

INPUT-ID.....................INPUT-ID
```

MAP2

```
Internal records:       DEPARTMENT
                        OFFICE

External file:          ARCFILE
External record:        ARCHIVE-RECORD-EMP

External field                  Internal field

ARCHIVE-TYPE.................$RESPONSE
ARCHIVE-DEPT-ID..............DEPT-ID-0410
ARCHIVE-OFFICE-CODE..........OFFICE-CODE-0450
ARCHIVE-EMPLOYEE-RECORD.....ARCHIVE-EMPLOYEE-RECORD
```

MAP3

```
Internal records:       None

External file:          ARCFILE
External record:        ARCHIVE-RECORD-COV

External field                  Internal field

ARCHIVE-TYPE.................$RESPONSE
ARCHIVE-COVERAGE-RECORD.....ARCHIVE-COVERAGE-RECORD
```

MAP4

```
Internal records:       None

External file:          REPFILE1
External record:        REPORT-RECORD

External field                  Internal field

REPORT-CC.....................REPORT-CC
REPORT-LINE...................REPORT-LINE
```

Step 4: Define the Process Modules

The next step in creating the application is to define process modules consisting of Application Development System process commands. You can define process modules by using the IDD DDDL compiler or the IDD menu facility.

The process modules required by the application are presented under "Step 5: Define the Dialogs," as part of the discussion of the dialogs with which they are associated. In this way, you can see how the modules fit into the application structure.

Note: The process modules shown in Step 5 have embedded comment lines, indicated by an exclamation point (!) in column 1. You do not have to key in these lines.

Step 5: Define the Dialogs

The next step in defining the application is to define its dialogs. A dialog is a collection of application components created in earlier steps, including file maps and process modules. You can define a dialog by using the online dialog compiler.

The dialogs in the archive application are described below.

Dialog	Description
ARCD1	Reads input records and acts as a mainline routine, passing control to ARCD2, ARCD3, and ARCD4, as required by the application; writes erroneous input records to the suspense file
ARCD2	Writes the specified employee record to the archive file, then returns control to ARCD1
ARCD3	Writes the coverage records associated with the specified employee record to the archive file, then returns control to ARCD1

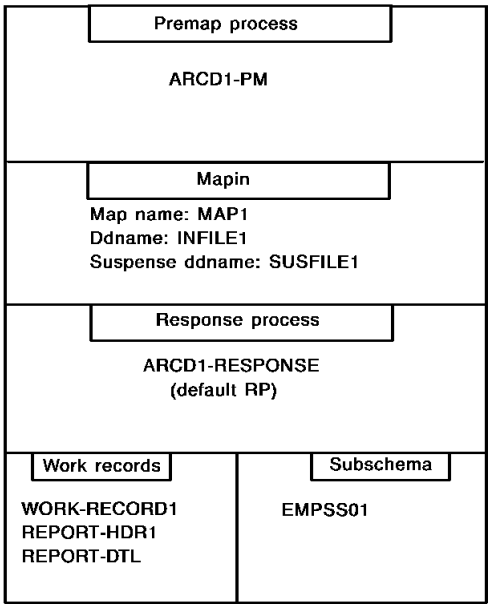
Dialog	Description
ARCD4	Determines the transaction report lines to be written to the report output file; for each line to be written, passes control to ARCD5; returns control to ARCD1 once all lines have been written for the transaction
ARCD5	Writes a report line to the report output file, then returns control to ARCD4

The definitions for these dialogs are provided separately below, along with the process modules associated with the dialogs and the mapin or mapout operations that the dialogs perform at runtime. For an illustration of how these dialogs fit together, see the diagram earlier in this section.

Dialog ARCD1

Dialog ARCD1 executes at the beginning of the application. ARCD1 reads an input file that contains a list of ids of employee records to be archived. ARCD1 serves as a mainline dialog, passing control to other dialogs, as required, to archive the employee and associated coverage records and to write report lines to a report file. ARCD1 also writes erroneous input records to a suspense file.

The dialog definition for ARCD1 is illustrated below:



The premap process, mapin operation, and response process are shown below.

Dialog ARCD1: Premap Process

```
!*********************************************************************
!*ARCD1-PM                                                           *
!*  -EXECUTED ONCE AT THE BEGINNING OF THE APPLICATION WHEN          *
!*   DLG ARCD1 BEGINS EXECUTION.                                     *
!*  -PERFORMS APPLICATION INITIALIZATION, THEN READS THE FIRST       *
!*   INPUT RECORD.                                                   *
!*********************************************************************
CALL INIT.
READ TRANSACTION.
!
!
!************************************************
!*SUBROUTINE INIT                                *
!*-SET UP FOR TRANSACTION REPORT.                 *
!************************************************
DEFINE INIT.
  MOVE DATE TO WORK-DATE-YYMMDD.
  MOVE WORK-YY1 TO WORK-YY2.
  MOVE WORK-MM1 TO WORK-MM2.
  MOVE WORK-DD1 TO WORK-DD2.
  MOVE WORK-DATE-MMDDYY TO REPORT-DATE.
  MOVE 55 TO WORK-LINE-CTR.
  GOBACK.
```

Dialog ARCD1: Mapin Operation

```
External field                Internal field

INPUT-ID.....................>INPUT-ID
```

Dialog ARCD1: Response Process

```
!*************************************************************************
!*ARCD1-RESPONSE                                                         *
!*  -EXECUTED AFTER DLG ARCD1'S MAPIN OPERATION.  DEFAULT RESPONSE       *
!*   PROCESS FOR ARCD1.                                                  *
!*  -PERFORMS MAINLINE PROCESSING OF INPUT RECORD.                       *
!*************************************************************************
READY USAGE-MODE UPDATE.
!
!
!************************************************
!*-TERMINATE APPLICATION ON AN EOF CONDITION     *
!************************************************
IF $EOF
  DO.
  WRITE LOG TEXT '***EOF ON INPUT***'.
  LEAVE APPLICATION.
  END.
!
!
!************************************************
!*-INPUT-ID CONTAINS THE ID OF THE EMP REC       *
!* TO BE ARCHIVED.                              *
!*-ATTEMPT TO RETRIEVE THE RECORD FROM THE       *
!* DATABASE.                                    *
!*-IF THE RECORD DOES NOT EXIST, CALL AN         *
!* ERROR ROUTINE.                               *
!*-ON ANY OTHER DB ERROR, TERMINATE THE APP.     *
!************************************************
MOVE INPUT-ID TO EMP-ID-0415.
OBTAIN CALC EMPLOYEE.
IF DB-REC-NOT-FOUND
  CALL ERRRTN.
IF DB-ANY-ERROR
  ABORT TEXT 'DB ERROR ON EMPLOYEE OBTAIN'.
!
!
!************************************************
!*-PASS CONTROL TO DLG ARCD2, WHICH ARCHIVES     *
!* THE EMPLOYEE RECORD.                         *
!*-PASS CONTROL TO DLG ARCD3, WHICH ARCHIVES     *
!* THE ASSOCIATED COVERAGE RECORDS.             *
!************************************************
LINK NOSAVE 'ARCD2'.
LINE NOSAVE 'ARCD3'.
```

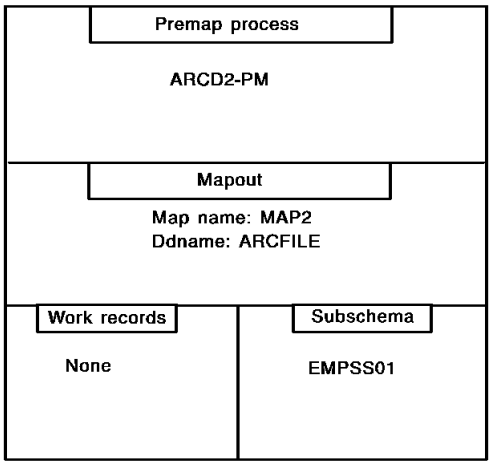
```
!
!
!************************************************
!*-SET UP FOR TRANSACTION REPORT, THEN PASS      *
!* CONTROL TO DLG ARCD4, WHICH, ALONG WITH       *
!* ARCD5, WRITE A REPORT LINE.                   *
!************************************************
MOVE 'RECORD ARCHIVED' TO REPORT-STATUS.
MOVE EMP-ID-0415 TO REPORT-ID.
MOVE EMP-LAST-NAME-0415 TO REPORT-LNAME.
MOVE EMP-FIRST-NAME-0415 TO REPORT-FNAME.
LINK NOSAVE 'ARCD4'.
!
!
!************************************************
!*-DELETE THE EMPLOYEE RECORD FROM THE           *
!* DATABASE ALONG WITH ALL COVERAGE RECORDS.     *
!*-ON ANY DB ERROR, TERMINATE THE APPL.          *
!************************************************
OBTAIN CALC EMPLOYEE.
ERASE EMPLOYEE ALL MEMBERS.
IF DB-ANY-ERROR
  ABORT TEXT 'DB ERROR ON EMPLOYEE ERASE'.
!
!
!************************************************
!*-READ THE NEXT INPUT RECORD.                   *
!************************************************
READ TRANSACTION.
!
!
!************************************************
!*SUBROUTINE ERRRTN                              *
!*-CALLED WHEN THE REQUESTED EMP REC IS NOT      *
!* FOUND IN THE DATABASE                         *
!*-SET UP FOR TRANSACTION REPORT, THEN           *
!* PASS CONTROL TO ARCD4, WHICH, ALONG WITH      *
!* ARCD5, WRITES A REPORT LINE.                  *
!*-SET INPUT-ID OF THE INPUT MAP IN ERROR.       *
!*-ISSUE WRITE TRANSACTION COMMAND, WHICH        *
!* WRITES THE INPUT RECORD TO THE SUSPENSE       *
!* FILE, THEN READS THE NEXT INPUT RECORD.       *
!************************************************
```

```
DEFINE ERRRTN.
  MOVE 'RECORD NOT FOUND' TO REPORT-STATUS.
  MOVE EMP-ID-0415 TO REPORT-ID.
  MOVE SPACES TO REPORT-FNAME.
  MOVE SPACES TO REPORT-LNAME.
  LINK NOSAVE 'ARCD4'.
  MODIFY MAP TEMP FOR (INPUT-ID) EDIT ERROR.
  WRITE TRANSACTION.
```

Dialog ARCD2

Dialog ARCD2 executes when it receives control from ARCD1. ARCD2 archives an employee record.

The dialog definition for ARCD2 is illustrated below:



The premap process and mapout operation are shown below,

Dialog ARCD2: Premap Process

```
!*************************************************************************
!*ARCD2-PM                                                                *
!*  -EXECUTED AT THE BEGINNING OF DLG ARCD2.                              *
!*  -ARCHIVES AN EMPLOYEE RECORD.                                         *
!*  -THE EMP ARCHIVE REC CONSISTS OF THE FOLLOWING FIELDS:                *
!*       -TYPE FIELD (INTERNAL FIELD IS $RESPONSE)                        *
!*       -DEPARTMENT ID                                                   *
!*       -OFFICE ID                                                       *
!*       -EMPLOYEE RECORD                                                 *
!*************************************************************************
```

```
!
!
!************************************************
!*-MOVE RECORD TYPE TO $RESPONSE.               *
!************************************************
MOVE 'E' TO $RESPONSE.
!
!
!************************************************
!*-RETRIEVE THE EMPLOYEE'S OFFICE AND           *
!* DEPARTMENT IDS.                              *
!*-ON ANY DATABASE ERROR, ABORT APPL.           *
!************************************************
OBTAIN OWNER WITHIN OFFICE-EMPLOYEE.
IF DB-ANY-ERROR
  ABORT TEXT 'DB ERROR ON OFFICE OBTAIN'.
OBTAIN OWNER WITHIN DEPT-EMPLOYEE.
IF DB-ANY-ERROR
  ABORT TEXT 'DB ERROR ON DEPARTMENT OBTAIN'.
!
!
!************************************************
!*-MOVE EMPLOYEE RECORD TO ARCHIVE RECORD       *
!************************************************
MOVE EMPLOYEE TO ARCHIVE-EMPLOYEE-RECORD.
!
!
!************************************************
!*-WRITE THE RECORD, THEN RETURN TO ARCD1.      *
!************************************************
WRITE TRANSACTION RETURN.
```

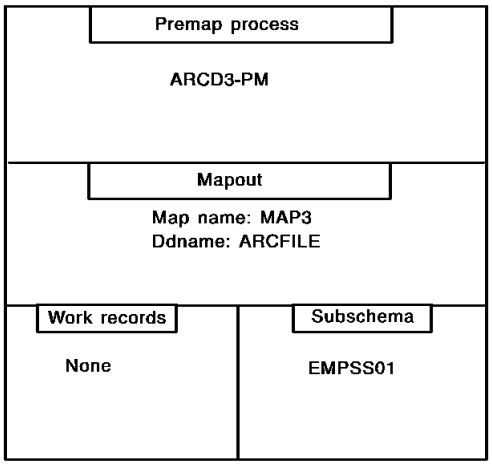
Dialog ARCD2: Mapout Operation

```
Internal field                 External field

$RESPONSE.....................>ARCHIVE-TYPE
DEPT-ID-0410..................>ARCHIVE-DEPT-ID
OFFICE-CODE-0450..............>ARCHIVE-OFFICE-CODE
ARCHIVE-EMPLOYEE-RECORD.......>ARCHIVE-EMPLOYEE-RECORD
```

Dialog ARCD3

Dialog ARCD3 executes when it receives control from ARCD1. ARCD3 archives all coverage records associated with the employee record being archived.

The dialog definition for ARCD3 is illustrated below:



The premap process and mapout operation are shown below.

Dialog ARCD3: Premap Process

```
!*********************************************************************
!*ARCD3-PM                                                           *
!*  -EXECUTED AT THE BEGINNING OF DLG ARCD3.                         *
!*  -ARCHIVES ALL ASSOCIATED COVERAGE RECORDS.                       *
!*  -THE COVERAGE ARCHIVE REC CONSISTS OF THE FOLLOWING FIELDS:      *
!*       -TYPE FIELD (INTERNAL FIELD IS $RESPONSE)                   *
!*       -COVERAGE RECORD                                            *
!*********************************************************************
!
!
!************************************************
!*-MOVE RECORD TYPE TO $RESPONSE.               *
!************************************************
MOVE 'C' TO $RESPONSE.
!
!
!************************************************
!*-RETRIEVE A COVERAGE RECORD.                  *
!*-ON END-OF-SET, RETURN TO ARCD1.              *
!*-IF ANY OTHER DB ERROR, ABORT APPL.           *
!************************************************
```

```
IF FIRST-TIME
  OBTAIN FIRST COVERAGE WITH EMP-COVERAGE.
ELSE
  OBTAIN NEXT COVERAGE WITHIN EMP-COVERAGE.
IF DB-END-OF-SET
  RETURN.
IF DB-ANY-ERROR
  ABORT TEXT 'DB ERROR ON COVERAGE OBTAIN'.
!
!
!************************************************
!*-MOVE COVERAGE RECORD TO ARCHIVE RECORD       *
!************************************************
MOVE COVERAGE TO ARCHIVE-COVERAGE-RECORD.
!
!
!************************************************
!*-WRITE THE RECORD, THEN REEXECUTE THE         *
!* PROCESS TO ARCHIVE THE NEXT COVERAGE REC.    *
!************************************************
WRITE TRANSACTION CONTINUE.
```

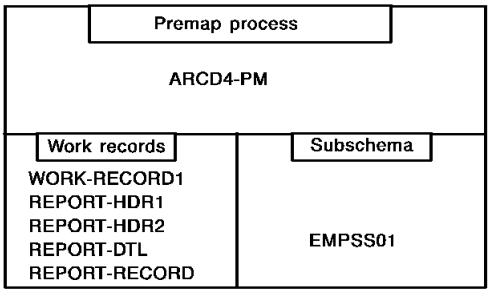
Dialog ARCD3: Mapout Operation

```
Internal field                 External field

$RESPONSE.....................>ARCHIVE-TYPE
ARCHIVE-COVERAGE-RECORD........>ARCHIVE-COVERAGE-RECORD
```

Dialog ARCD4

Dialog ARCD4 executes when it receives control from ARCD1. ARCD4 prepares report lines to be written to the report file and passes control to ARCD5, which actually writes the lines to the output file.

The dialog definition for ARCD4 is illustrated below:



The premap process is shown below.

Dialog ARCD4: Premap Process

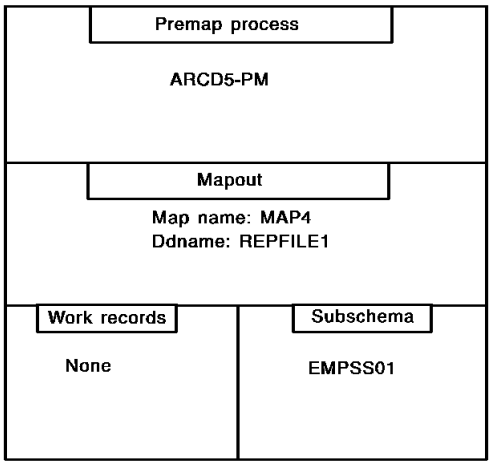
```
!*********************************************************************
!*ARCD4-PM                                                           *
!*  -EXECUTED AT THE BEGINNING OF DLG ARCD4.                         *
!*  -DETERMINES REPORT LINE(S) TO BE WRITTEN.                        *
!*  -THE REPORT RECORD CONSISTS OF THE FOLLOWING FIELDS:             *
!*       -REPORT-CC —— CARRIAGE CONTROL CHARACTER                    *
!*       -REPORT-LINE —— ACTUAL REPORT LINE                          *
!*  -THIS PROCESS MOVES THE PROPER DATA INTO THESE FIELDS, THEN      *
!*   PASSES CONTROL TO ARCD5, WHICH WRITES A LINE.                   *
!*********************************************************************
ADD 1 TO WORK-LINE-CTR.
!
!
!************************************************
!*-IF LINE COUNTER > 55, CALL A ROUTINE THAT     *
!* PRINTS HEADING LINES, THEN MOVE 0 TO          *
!* REPORT-CC, SO THAT THE FIRST DETAIL LINE      *
!* IS DOUBLE SPACED.                             *
!*-IF LINE COUNTER IS NOT > 55, MOVE ' ' TO      *
!* REPORT-CC, SO THAT THE DETAIL LINE IS         *
!* SINGLE SPACED.                                *
!************************************************
IF WORK-LINE-CTR > 55
  DO.
  CALL PRINTHDR.
  MOVE '0' TO REPORT-CC.
  END.
ELSE
  MOVE ' ' TO REPORT-CC.
!
!
!************************************************
!*-MOVE THE DETAIL LINE TO REPORT-LINE.          *
!*-PASS CONTROL TO ARCD5, WHICH WRITES THE       *
!* LINE.                                         *
!*-RETURN CONTROL TO ARCD1.                      *
!************************************************
MOVE REPORT-DTL TO REPORT-LINE.
LINK NOSAVE 'ARCD5'.
RETURN.
!
```

```
!
!************************************************
!*SUBROUTINE PRINTHDR                            *
!*-CALLED WHEN PAGE HEADERS ARE TO BE PRINTED    *
!*-SET UP FIRST HEADING LINE, THEN PASS          *
!* CONTROL TO ARCD5, WHICH WRITES IT.            *
!*-SET UP SECOND HEADING LINE, THEN PASS         *
!* CONTROL TO ARCD5, WHICH WRITES IT.            *
!*-GO BACK TO MAIN ROUTINE TO WRITE THE          *
!* DETAIL LINE.                                  *
!************************************************
DEFINE PRINTHDR.
  MOVE 1 TO WORK-LINE-CTR.
  ADD 1 TO WORK-PAGE-CTR.
  MOVE WORK-PAGE-CTR TO REPORT-PAGE.
  MOVE REPORT-HDR1 TO REPORT-LINE.
  MOVE '1' TO REPORT-CC.
  LINK NOSAVE 'ARCD5'.
  MOVE REPORT-HDR2 TO REPORT-LINE.
  MOVE '-' TO REPORT-CC.
  LINK NOSAVE 'ARCD5'.
  GOBACK.
```

Dialog ARCD5

Dialog ARCD5 executes when it receives control from ARCD4. ARCD5 writes a report line to the report file.

The dialog definition for ARCD5 is illustrated below:



The premap process and mapout operation are shown below.

Dialog ARCD5: Premap Process

```
!*********************************************************************
!*ARCD5-PM                                                           *
!*  -EXECUTED AT THE BEGINNING OF DLG ARCD5.                         *
!*  -WRITES A REPORT LINE, THEN PASSES CONTROL BACK TO ARCD4.        *
!*********************************************************************
WRITE TRANSACTION RETURN.
```

Dialog ARCD5: Mapout Operation

```
Internal field                External field

REPORT-CC.....................>REPORT-CC
REPORT-LINE...................>REPORT-LINE
```

Executing the Application

You execute the application by executing the batch program ADSBATCH, as described in Runtime Considerations (see page 79).

Physical File Characteristics

Characteristics of the physical files that the application accesses are shown below. Under z/OS, the JCL specifications or data set label for each file should specify the record formats, logical record lengths, and block sizes indicated. Under z/VSE, these characteristics should be specified in IDMSFILE control statements. The JCL for executing an CA ADS Batch application is provided in Runtime Considerations (see page 79).

Data set name	DDNAME/ filename	Record format	Logical record size	Block size
INFILE1	INFILE1	FB	4	4000
SUSFILE1	SUSFILE1	FB	4	4000
ARCFILE	ARCFILE	VB	128	3000
REPFILE1	REPFILE1	FB	133	1330
ADSLOGA	ADSLOGA	VB	320	12804

Control statements

The JCL can also include control statements. In the following sample set of control statements, you specify the application entry point, the requestor's id, and the ddname of the log file:

```
ENTRY POINT DIALOG ARCD1.
REQUESTOR MPK.
LOG FILE1= ADSLOGA.
```

The contents of INFILE1 before the application is executed, and the contents of REPFILE1, SUSFILE1, ARCFILE, and ADSLOGA after the application has been executed are shown below:

Note: The employee-record restore application, implemented later in this appendix, assumes that the archive file is ordered by employee id. Therefore, make sure that your input file for this application is also ordered by employee id.

INFILE1

```
3000
4000
5001
```

REPFILE1

```
1              TRANSACTION REPORT   DATE: 10/18/99   PAGE:  001
-EMPLOYEE ID    NAME                          STATUS
0     3000      STERNS          JOSEPH        RECORD ARCHIVED
      4000                                    RECORD NOT FOUND
      5001      WATSON          BRIAN         RECORD ARCHIVED
```

SUSFILE1

```
4000
```

ARCFILE

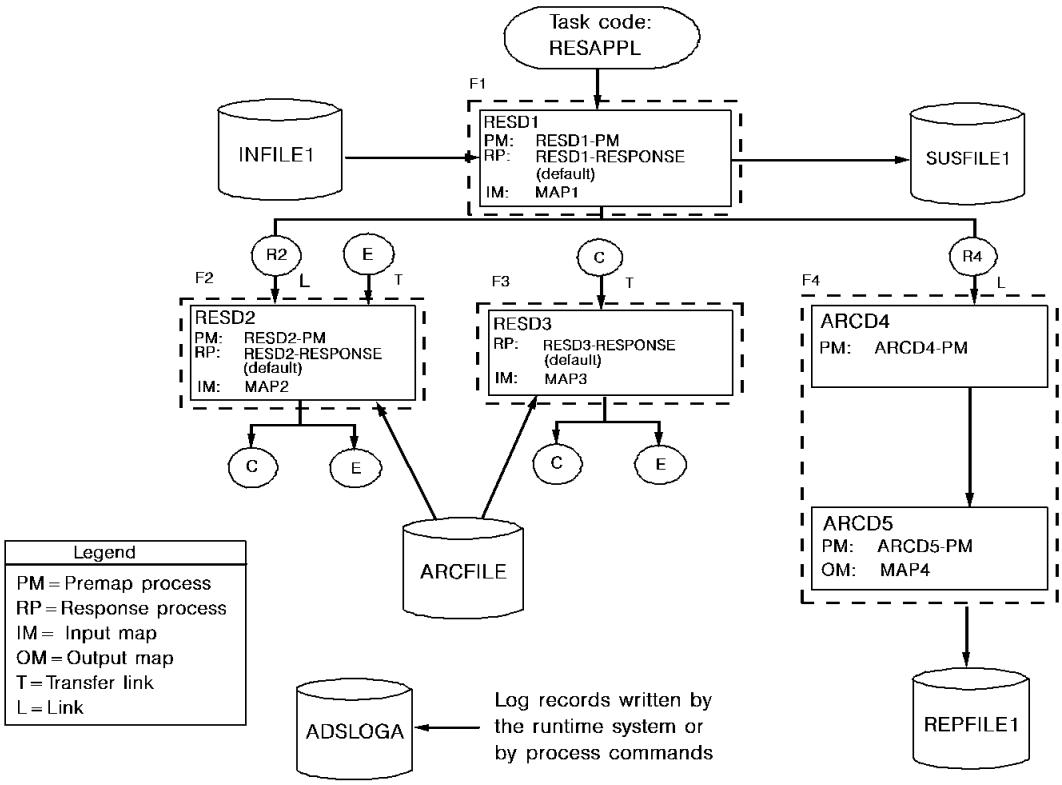
```
E40000123000JOSEPH    STERMS            100 HANGER DRIVE     BOSTON ...
C800101830102M232
C830103850601M232
E40000125001BRIAN     WATSON            300 ST CATHERINE W.  MONTREAL ...
C850101      D123
```

ADSLOGA

```
IDMS DC506801 V1 SUSPENSE FILE SUSFILE1 --  RECORD#   1  --  IMAGE IS  '4000'
***EOF ON INPUT***
IDMS DC507001 V1 BASE FILE INFILE1 CLOSED.
IDMS DC507003 V1   3 LOGICAL RECORDS READ.
IDMS DC507002 V1 BASE DATASET MEN.C0600.MPK.INFILE1 CLOSED.
IDMS DC507003 V1   1 PHYSICAL RECORDS READ.
IDMS DC507001 V1 BASE FILE ISUSFILE1 CLOSED.
IDMS DC507004 V1   1 LOGICAL RECORDS WRITTEN.
IDMS DC507002 V1 BASE DATASET MEN.C0600.MPK.SUSFILE1 CLOSED.
IDMS DC507004 V1   1 PHYSICAL RECORDS WRITTEN.
IDMS DC507001 V1 SUPPLEMENTARY FILE ARCFILE CLOSED.T
IDMS DC507004 V1   2 LOGICAL RECORDS WRITTEN.
IDMS DC507001 V1 BASE FILE ARCFILE CLOSED.E
IDMS DC507004 V1   3 LOGICAL RECORDS WRITTEN.
IDMS DC507002 V1 BASE DATASET MEN.C0600.MPK.ARCFILE CLOSED.
IDMS DC507004 V1   1 PHYSICAL RECORDS WRITTEN.
IDMS DC507001 V1 BASE FILE REPFILE1 CLOSED.
IDMS DC507004 V1   5 LOGICAL RECORDS WRITTEN.
IDMS DC507002 V1 BASE DATASET MEN.C0600.MPK.REPFILE1 CLOSED.
IDMS DC507004 V1   1 PHYSICAL RECORDS WRITTEN.
```

Employee-Record Restore Application

The employee-record restore application restores selected employee records and their associated coverage records from an archive file (created in the employee-record archive application) to an CA IDMS/DB database. The following drawing shows the file access, components, and flow of control for the application.



The files accessed by this application are the same as those accessed in the archive application. ARCFILE, which was an output file in the archive application, is an input file in the restore application.

The restore application uses a global application structure defined with the application generator. In the drawing above, functions are represented by dotted rectangles; responses are represented by circles.

The application structure enables the application to access an input file that has more than one record layout (ARCFILE). The structure ensures that type E (employee) records are mapped in by dialog RESD2, and that type C (coverage) records are mapped in by dialog RESD3.

Processing

At runtime, the following processing occurs when RESD2 or RESD3 reads a record from the archive file:

- **If RESD2 reads a type E record**, the record is mapped in using MAP2 and a response process is selected.
- **If RESD2 reads a type C record**, control passes immediately to dialog RESD3. The record is not mapped in; RESD3 must perform a read operation to map in the record.
- **If RESD3 reads a type C record**, the record is mapped in using MAP3 and a response process is selected.
- **If RESD3 reads a type E record**, control passes immediately to dialog RESD2. The record is not mapped in; RESD2 must perform a read operation to map in the record.

Dialogs Used

The dialogs used in the application are described below.

Dialog	Description
RESD1	Reads input records and acts as a mainline routine, passing control (by means of the application structure) to dialogs RESD2 and ARCD4, as required by the application; writes erroneous input records to a suspense file
RESD2	Finds the requested employee record in the archive file; restores the record; performs another read operation to begin reading associated coverage records; if a coverage record is read, control passes automatically to dialog RESD3; returns control to RESD1 when all associated coverage records have been restored
RESD3	Restores coverage records associated with the restored employee record; if an employee record is read, control passes automatically back to dialog RESD2
ARCD4 and ARCD5	Write transaction report lines to an output file

Steps

To create the application, you perform the following steps:

1. Define the application structure.
2. Define the process modules for the application's dialogs.
3. Define the dialogs.

Note that you do not define the files, records, and file maps used by the application, nor do you define dialogs ARCD4 and ARCD5; you already defined these components as part of the archive application.

The steps you perform to define the restore application are described on the following pages, followed by a discussion of executing the application.

Step 1: Define the Application Structure

You define the application structure by using the online application generator. The application structure consists of responses, functions, and a task code. The specifications you make are shown below:

Application Specifications

Component	Name	Characteristics
Responses	R2	■ Invokes F2 ■ LINK command ■ NOSAVE option
	R4	■ Invokes F4 ■ LINK command ■ NOSAVE option
	E	■ Invokes F2 ■ TRANSFER command ■ NOFINISH option
	C	■ Invokes F3 ■ TRANSFER command ■ NOFINISH option
Functions	F1	■ Assoc. with RESD1 ■ Valid responses: ■ F1 ■ F4
	F2	■ Assoc. with RESD2 ■ Valid responses: ■ E ■ C

Component	Name	Characteristics
	F3	■ Assoc. with RESD3 ■ Valid responses: ■ E ■ C
	F4	■ Assoc. with ARCD4
Task code	RESAPPL	■ Invokes F1

Note: For responses, you provide the NOSAVE and NOFINISH specifications.

Step 2: Define the Process Modules

As the next step in defining the employee-record restore application, you define process modules consisting of Application Development System process commands. You can define process modules using the IDD DDDL compiler or the IDD menu facility.

The process modules required by the application are presented under "Step 3: Define the Dialogs," together with the dialogs with which they are associated. In this way, you can see how the modules fit into the application structure.

Note: The process modules shown in Step 3 have embedded comment lines, indicated by an exclamation point (!) in column 1. You do not have to key in these lines.

Step 3: Define the Dialogs

The next step in defining the application is to define its dialogs. A dialog is a collection of application components created in earlier steps, including file maps and process modules. You can define a dialog by using the online dialog compiler.

Dialogs Used

The dialogs in the restore application are described below.

Name	Description
RESD1	Reads input records and acts as a mainline routine, passing control (by means of the application structure) to dialogs RESD2 and ARCD4, as required by the application; writes erroneous input records to a suspense file

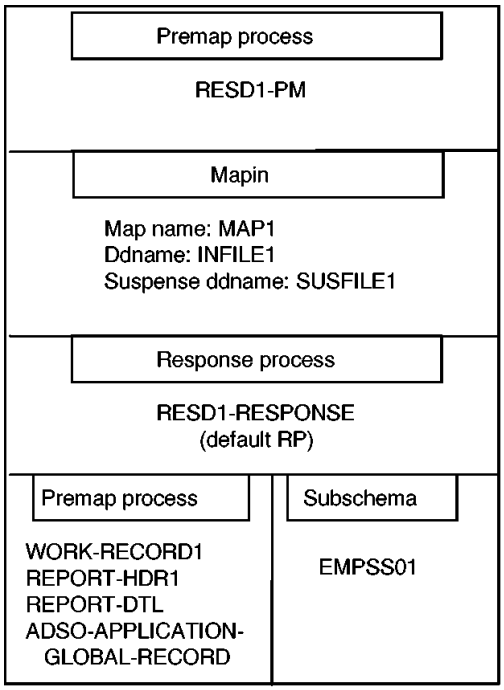
Name	Description
RESD2	Finds the requested employee record in the archive file; restores the record; performs another read operation to begin reading associated coverage records; if a coverage record is read, control passes automatically to dialog RESD3; returns control to RESD1 when all associated coverage records have been restored
RESD3	Restores coverage records associated with the restored employee record; if an employee record is read, control passes automatically back to dialog RESD2
ARCD4 ARCD5	Write transaction report lines to an output file

The definitions for these dialogs (excluding ARCD4 and ARCD5, which were defined in the employee-record archive application) are provided separately below, along with the process modules associated with the dialogs and the mapin or mapout operations that these dialogs perform at runtime. For an illustration of how these dialogs fit together, see the diagram earlier in this section.

Dialog RESD1

Dialog RESD1 reads input records and acts as a mainline routine, passing control (by means of the application structure) to dialogs RESD2 and ARCD4, as required by the application. RESD1 also writes erroneous input records to a suspense file.

The dialog definition for RESD1 is illustrated below:



The premap process, mapin operation, and response process are shown below.

Dialog RESD1: Premap Process

```
!*********************************************************************
!*RESD1-PM                                                           *
!*  -EXECUTED AT THE BEGINNING OF THE APPLICATION WHEN               *
!*   DLG RESD1 BEGINS EXECUTION.                                     *
!*  -PERFORMS APPLICATION INITIALIZATION, THEN READS THE FIRST       *
!*   INPUT RECORD.                                                   *
!*********************************************************************
CALL INIT.
READ TRANSACTION.
!
!
```

```
!************************************************
!*SUBROUTINE INIT                                *
!*-INITIALIZE WORK-ARC-ID, WHICH CONTAINS        *
!* THE ID OF THE MOST RECENTLY READ ARCHIVED     *
!* EMPLOYEE RECORD.                              *
!*-SET UP FOR TRANSACTION REPORT.                *
!************************************************
DEFINE INIT.
  MOVE 0 TO WORK-ARC-ID.
  MOVE DATE TO WORK-DATE-YYMMDD.
  MOVE WORK-YY1 TO WORK-YY2.
  MOVE WORK-MM1 TO WORK-MM2.
  MOVE WORK-DD1 TO WORK-DD2.
  MOVE WORK-DATE-MMDDYY TO REPORT-DATE.
  MOVE 55 TO WORK-LINE-CTR.
  GOBACK.
```

Dialog RESD1: Mapin Operation

```
External field                 Internal field

INPUT-ID.....................>INPUT-ID
```

Dialog RESD1: Response Process

```
!*********************************************************************
!*RESD1-RESPONSE                                                     *
!*  -EXECUTED AFTER DLG RESD1'S MAPIN OPERATION.  DEFAULT RESPONSE    *
!*   PROCESS FOR RESD1.                                               *
!*  -PERFORMS MAINLINE PROCESSING OF INPUT RECORD.                    *
!*********************************************************************
READY USAGE-MODE UPDATE.
!
!
!************************************************
!*-TERMINATE APPLICATION ON AN EOF CONDITION     *
!************************************************
IF $EOF
  DO.
  WRITE LOG TEXT '***EOF ON INPUT***'.
  LEAVE APPLICATION.
  END.
!
!
```

```
!************************************************
!*-INPUT-ID CONTAINS THE ID OF THE EMP REC       *
!* TO BE RESTORED.                              *
!*-ATTEMPT TO RETRIEVE THE RECORD FROM THE       *
!* DATABASE.                                    *
!*-IF THE RECORD ALREADY EXISTS, CALL AN         *
!* ERROR ROUTINE.                               *
!*-ON ANY OTHER ERROR, TERMINATE THE APP.        *
!************************************************
MOVE INPUT-ID TO EMP-ID-0415.
OBTAIN CALC EMPLOYEE.
IF DB-STATUS-OK
  CALL ERRRTN.
IF DB-REC-NOT-FOUND
  NEXT COMMAND.
ELSE
  IF DB-ANY-ERROR
    ABORT TEXT 'DB ERROR ON EMPLOYEE OBTAIN'.
!
!
!************************************************
!*-WORK-ARCFILE-STATUS IS SET TO EOF WHEN        *
!* THE ARCHIVE FILE HAS REACHED THE EOF.        *
!*-NOTFND IS A SUBROUTINE THAT IS CALLED         *
!* WHEN THE EMP REC TO BE RESTORED CANNOT BE     *
!* FOUND IN THE ARCHIVE FILE.                   *
!*-IF WORK-ARCFILE-STATUS = EOF, CALL NOTFND.    *
!************************************************
IF WORK-ARCFILE-STATUS = 'EOF'
  CALL NOTFND.
!

!
!************************************************
!*-PASS CONTROL TO FUNCTION F2, WHICH            *
!* EXECUTES DIALOG RESD2.                       *
!*-F2 (RESD2) AND F3 (RESD3), WHICH IS A VALID   *
!* FUNCTION FROM F2, READ THE ARCHIVE FILE       *
!* AND ATTEMPT TO RESTORE THE EMP REC AND ITS    *
!* ASSOCIATED COVERAGE RECS.  IF THE EMP REC     *
!* CANNOT BE FOUND, THESE DIALOGS SET WORK-      *
!* STATUS TO 'NOT FOUND'.                       *
!************************************************
MOVE 'R2' TO AGR-CURRENT-RESPONSE.
MOVE SPACES TO WORK-STATUS.
EXECUTE NEXT FUNCTION.
!
!
```

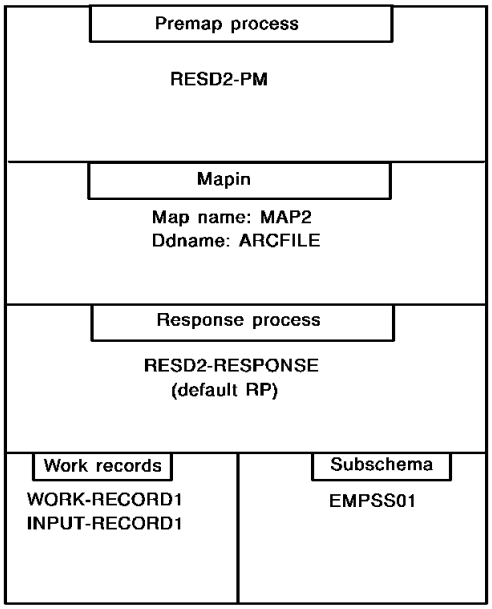
```
!************************************************
!*-IF THE REQUESTED EMP REC WAS NOT FOUND,       *
!* CALL THE NOTFND ERROR ROUTINE.                *
!*-IF THE RECORD WAS FOUND AND RESTORED,         *
!* CALL THE FOUND ROUTINE.                       *
!************************************************
IF WORK-STATUS = 'NOT FOUND'
  CALL NOTFND.
ELSE
  CALL FOUND.
!
!
!************************************************
!*SUBROUTINE ERRRTN                              *
!*-CALLED WHEN THE REQUESTED EMP REC IS          *
!* ALREADY ON THE DATABASE.                      *
!*-SET UP FOR TRANSACTION REPORT, THEN           *
!* PASS CONTROL TO FUNCTION F4 (DLG ARCD4),      *
!* WHICH, ALONG WITH ARCD5, WRITES A REPORT      *
!* LINE.                                         *
!*-SET INPUT-ID OF THE INPUT MAP IN ERROR.       *
!*-ISSUE WRITE TRANSACTION COMMAND, WHICH        *
!* WRITES THE INPUT RECORD TO THE SUSPENSE       *
!* FILE, THEN READS THE NEXT INPUT RECORD.       *
!************************************************
DEFINE ERRRTN.
  MOVE 'EMPLOYEE ALREADY ON DATABASE' TO REPORT-STATUS.
  MOVE INPUT-ID TO REPORT-ID.
  MOVE EMP-LAST-NAME-0415 TO REPORT-LNAME.
  MOVE EMP-FIRST-NAME-0415 TO REPORT-FNAME.
  MOVE 'R4' TO AGR-CURRENT-RESPONSE.
  EXECUTE NEXT FUNCTION.
  MODIFY MAP TEMP FOR (INPUT-ID) EDIT ERROR.
  WRITE TRANSACTION.
!
```

```
!
!************************************************
!*SUBROUTINE NOTFND                             *
!*-CALLED WHEN THE REQUESTED EMP REC IS         *
!* NOT ON THE ARCHIVE FILE.  (NOTE THAT THE     *
!* APPL. ASSUMES THAT THE INPUT FILE AND        *
!* ARCHIVE FILE ARE BOTH ORDERED BY EMP ID.)    *
!*-SET UP FOR TRANSACTION REPORT, THEN          *
!* PASS CONTROL TO FUNCTION F4 (DLG ARCD4),     *
!* WHICH, ALONG WITH ARCD5, WRITES A REPORT     *
!* LINE.                                        *
!*-SET INPUT-ID OF THE INPUT MAP IN ERROR.      *
!*-ISSUE WRITE TRANSACTION COMMAND, WHICH       *
!* WRITES THE INPUT RECORD TO THE SUSPENSE      *
!* FILE, THEN READS THE NEXT INPUT RECORD.      *
!************************************************
DEFINE NOTFND.
  MOVE 'EMPLOYEE NOT FOUND' TO REPORT-STATUS.
  MOVE INPUT-ID TO REPORT-ID.
  MOVE SPACES TO REPORT-LNAME.
  MOVE SPACES TO REPORT-FNAME.
  MOVE 'R4' TO AGR-CURRENT-RESPONSE.
  EXECUTE NEXT FUNCTION.
  MODIFY MAP TEMP FOR (INPUT-ID) EDIT ERROR.
  WRITE TRANSACTION.
!
!
!************************************************
!*SUBROUTINE FOUND                              *
!*-CALLED WHEN THE REQUESTED EMP REC HAS BEEN   *
!* FOUND ON THE ARCHIVE FILE AND RESTORED.      *
!*-SET UP FOR TRANSACTION REPORT, THEN          *
!* PASS CONTROL TO FUNCTION F4 (DLG ARCD4),     *
!* WHICH, ALONG WITH ARCD5, WRITES A REPORT     *
!* LINE.                                        *
!*-READ THE NEXT INPUT RECORD.                  *
!************************************************
DEFINE FOUND.
  MOVE 'EMPLOYEE RESTORED' TO REPORT-STATUS.
  MOVE INPUT-ID TO REPORT-ID.
  MOVE EMP-LAST-NAME-0415 TO REPORT-LNAME.
  MOVE EMP-FIRST-NAME-0415 TO REPORT-FNAME.
  MOVE 'R4' TO AGR-CURRENT-RESPONSE.
  EXECUTE NEXT FUNCTION.
  READ TRANSACTION.
```

Dialog RESD2

Dialog RESD2 processes archived employee records. RESD2 finds the requested employee record in the archive file and restores it. It then performs another read operation to begin reading associated coverage records; if a coverage record is read, control passes automatically to dialog RESD3.

The dialog definition for RESD2 is illustrated below:



The premap process, mapin operation, and response process are shown below.

Dialog RESD2: Premap Process

```
!*********************************************************************
!*RESD2-PM                                                           *
!*  -EXECUTED AT THE BEGINNING OF DLG RESD2.                         *
!*  -IN THIS APPLICATION, CONTROL PASSES HERE WHEN:                  *
!*      -RESD1 PASSES CONTROL TO RESD2 IN ORDER TO FIND AND          *
!*       RESTORE THE REQUESTED EMPLOYEE RECORD.                      *
!*      -RESD2-RESPONSE ISSUES A CONTINUE COMMAND AFTER AN           *
!*       EMPLOYEE RECORD HAS BEEN READ SO THAT RESD2-PM CAN PROCESS  *
!*       THE RECORD.                                                 *
!*      -DLG RESD3 IS TO READ THE NEXT RECORD FROM THE               *
!*       ARCHIVE FILE, BUT THE NEXT RECORD IS AN EMPLOYEE RECORD.    *
!*       IN THIS CASE, CONTROL PASSES AUTOMATICALLY BACK TO RESD2,   *
!*       AS SPECIFIED BY THE APPLICATION DEFINITION.                 *
!*      -RESD2-PM PROCESSES THE MOST RECENTLY READ ARCHIVE FILE.     *
!*      -NOTE THAT THE PREMAP PROCESS ASSUMES THAT BOTH THE INPUT    *
!*       AND ARCHIVE FILE RECORDS ARE ORDERED BY EMP ID.             *
!*      -NOTE THAT WORK-STATUS IS SET TO 'RESTORED' WHEN THE         *
!*       REQUESTED EMP HAS BEEN RESTORED ONTO THE DATABASE.          *
!*********************************************************************
READY USAGE-MODE UPDATE.
!
!
!************************************************
!*-IF THE ARCHIVED RECORD HAS BEEN RESTORED,     *
!* RETURN TO RESD1 TO READ THE NEXT INPUT        *
!* RECORD.                                       *
!************************************************
IF WORK-STATUS = 'RESTORED'
  RETURN.
!
!
```

```
!************************************************
!*-WORK-ARC-ID IS THE ID OF THE MOST RECENTLY    *
!* READ ARCHIVED EMP REC.                       *
!*-INPUT-ID IS THE ID OF THE EMP-REC TO BE       *
!* RESTORED.                                    *
!*-IF THERE IS A MATCH, RESTORE THE EMP REC,     *
!* MOVE 'RESTORED' TO WORK-STATUS, THEN         *
!* READ THE NEXT ARCHIVE FILE RECORD IN ORDER    *
!* TO RESTORED THE EMPLOYEE'S ASSOCIATED        *
!* COVERAGE RECORDS.                            *
!************************************************
IF WORK-ARC-ID = INPUT-ID
  DO.
  CALL RESTRTN.
  MOVE 'RESTORED' TO WORK-STATUS.
  READ TRANSACTION.
  END.
!

!
!************************************************
!*-WORK-ARC-ID > INPUT ID MEANS THAT NO MATCH    *
!* WAS FOUND.                                   *
!*-IN THIS CASE, MOVE 'NOT FOUND' TO WORK-       *
!* STATUS, AND RETURN TO RESD1.                 *
!************************************************
IF WORK-ARC-ID > INPUT-ID
  DO.
  MOVE 'NOT FOUND' TO WORK-STATUS.
  RETURN.
  END.
!
!
!************************************************
!*-WORK-ARC-ID < INPUT ID (THE ONLY REMAINING    *
!* POSSIBILITY) MEANS THAT THE ARCHIVE FILE      *
!* SHOULD BE READ UNTIL A MATCH IS FOUND OR      *
!* UNTIL THE ARCHIVE EMP ID IS GREATER THAN      *
!* THE INPUT EMP ID.                            *
!************************************************
READ TRANSACTION.
!
!
```

```
!************************************************
!*SUBROUTINE RESTRTN                             *
!*-CALLED WHEN A MATCH IS FOUND.                 *
!*-SET CURRENCY ON EMPLOYEE'S OFFICE AND         *
!* DEPARTMENT, WHOSE IDS WERE ARCHIVED ALONG     *
!* WITH THE RECORD.                              *
!*-ON ANY DATABASE RROR, ABORT THE APPL.         *
!*-RESTORE THE ARCHIVED EMPLOYEE RECORD ONTO     *
!* THE DATABASE.                                 *
!************************************************
DEFINE RESTRTN.
  FIND CALC OFFICE.
  IF DB-ANY-ERROR
    ABORT TEXT '***DB ERROR ON FIND OFFICE***'.
  FIND CALC DEPARTMENT.
  IF DB-ANY-ERROR
    ABORT TEXT '***DB ERROR ON FIND DEPARTMENT***'.
  STORE EMPLOYEE.
  IF DB-ANY-ERROR
    ABORT TEXT '***DB ERROR ON STORE EMPLOYEE***'.
  GOBACK.
```

Dialog RESD2: Mapin Operation

```
External field                 Internal field

ARCHIVE-TYPE..................>$RESPONSE
ARCHIVE-DEPT-ID...............>DEPT-ID-0410
ARCHIVE-OFFICE-CODE...........>OFFICE-CODE-0450
ARCHIVE-EMPLOYEE-RECORD.......>$ARCHIVE-EMPLOYEE-RECORD
```

Dialog RESD2: Response Process

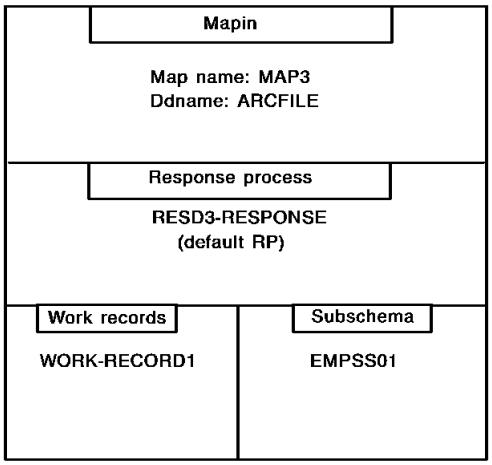
```
!*********************************************************************
!*RESD2-RESPONSE                                                     *
!*  -EXECUTED AFTER AN EMPLOYEE RECORD HAS BEEN READ FROM THE        *
!*   ARCHIVE FILE, OR AFTER AN ARCHIVE FILE EOF.  RESD2'S DEFAULT    *
!*   RESPONSE PROCESS.                                               *
!*  -PASSES CONTROL AS APPROPRIATE, DEPENDING ON CURRENT             *
!*   CONDITIONS.                                                     *
!*********************************************************************
!
!
```

```
!************************************************
!*-ON ARCHIVE FILE EOF, MOVE EOF TO WORK-        *
!* ARCFILE-STATUS.                               *
!*-IF THE REQUESTED EMPLOYEE HAS NOT BEEN RE-    *
!* STORED, MOVE 'NOT FOUND' TO WORK-STATUS       *
!*-RETURN TO RESD1.                             *
!************************************************
IF $EOF
  DO.
  MOVE 'EOF' TO WORK-ARCFILE-STATUS.
  IF WORK-STATUS NE 'RESTORED'
    MOVE 'NOT FOUND' TO WORK-STATUS.
  ELSE
    NEXT.
  RETURN.
  END.
!
!
!************************************************
!*-MOVE THE ARCHIVED RECORD TO THE EMPLOYEE      *
!* DATABASE RECORD.                              *
!*-MOVE THE EMP ID TO WORK-ARC-ID (ID OF THE     *
!* MOST RECENTLY READ ARCHIVED EMP REC).         *
!*-REEXECUTE THE PREMAP PROCESS TO PROCESS       *
!* THE RECORD JUST READ.                         *
!************************************************
MOVE ARCHIVE-EMPLOYEE-RECORD TO EMPLOYEE.
MOVE EMP-ID-0415 TO WORK-ARC-ID.
CONTINUE.
```

Dialog RESD3

Dialog RESD3 restores coverage records associated with the restored employee record. If an employee record is read, control passes back automatically to dialog RESD2.

The dialog definition for RESD2 is illustrated below:



The mapin operation and response process are shown below.

Dialog RESD3: Mapin Operation

```
External field                 Internal field

ARCHIVE-TYPE..................>$RESPONSE
ARCHIVE-COVERAGE-RECORD........>ARCHIVE-COVERAGE-RECORD
```

Dialog RESD3: Response Process

```
!*************************************************************************
!*RESD3-RESPONSE                                                         *
!*  -EXECUTED AFTER A COVERAGE RECORD HAS BEEN READ FROM THE             *
!*   ARCHIVE FILE, OR AFTER AN ARCHIVE FILE EOF.  RESD3'S DEFAULT        *
!*   RESPONSE PROCESS.                                                   *
!*  -PROCESSES THE COVERAGE RECORD OR EOF CONDITION.                     *
!*************************************************************************
READY USAGE-MODE UPDATE.
!
!
!*********************************************
!*-ON ARCHIVE FILE EOF, MOVE EOF TO WORK-     *
!* ARCFILE-STATUS.                           *
!*-IF THE REQUESTED EMPLOYEE HAS NOT BEEN RE- *
!* STORED, MOVE 'NOT FOUND' TO WORK-STATUS.  *
!*-RETURN TO RESD1.                         *
!*********************************************
IF $EOF
  DO.
  MOVE 'EOF' TO WORK-ARCFILE-STATUS.
```

```
  IF WORK-STATUS NE 'RESTORED'
    MOVE 'NOT FOUND' TO WORK-STATUS.
  ELSE
    NEXT.
  RETURN.
  END.
!
!
!************************************************
!*-IF WORK-STATUS NE 'RESTORED', THEN THE        *
!* COVERAGE RECORD SHOULDN'T BE RESTORED.        *
!* COVERAGE RECORDS ARE RESTORED ONLY IF         *
!* THEIR ASSOCIATED EMPLOYEE RECORDS ARE ALSO    *
!* RESTORED, AS INDICATED BY WORK-STATUS.        *
!*-INSTEAD, READ THE NEXT ARCHIVE FILE RECORD    *
!************************************************
IF WORK-STATUS NE 'RESTORED'
  READ TRANSACTION.
!
!
!************************************************
!*-MOVE THE ARCHIVE COVERAGE RECORD TO THE       *
!* COVERAGE DATABASE RECORD, SET CURRENCY        *
!* ON THE ASSOCIATED EMPLOYEE, AND STORE THE     *
!* RECORD.                                       *
!*-ON ANY DB ERROR, ABORT THE APPL.             *
!*-READ THE NEXT ARCHIVE FILE RECORD.           *
!************************************************
MOVE ARCHIVE-COVERAGE-RECORD TO COVERAGE.
FIND CALC EMPLOYEE.
STORE COVERAGE.
IF DB-ANY-ERROR
  ABORT TEXT '***DB ERROR ON STORE COVERAGE***'.
READ TRANSACTION.
!
!
```

Executing the Application

You execute the application by executing the batch program ADSBATCH, as described in Runtime Considerations (see page 79).

You should be able to use the same JCL and control statements that you used in the archive application (presented earlier in this section), with the exception of the ENTRY POINT control statement, which is shown below for the restore application:

```
ENTRY POINT TASK RESAPPL.
```

The contents of INFILE1 before the application is executed, and the contents of REPFILE1, SUSFILE1, and ADSLOGA after the application is executed are shown below.

Note: The restore application assumes that both the input file of employee ids and the archive file are ordered by employee id. Make sure that your input file is ordered properly.

INFILE1

```
3000
3010
4000
5001
```

REPFILE1

```
              TRANSACTION REPORT   DATE: 10/18/99   PAGE:  001
EMPLOYEE ID   NAME                       STATUS

     3000     STERNS          JOSEPH      EMPLOYEE RESTORED
     3010     PETERSON        RUTH        EMPLOYEE ALREADY ON DATABASE
     4000                                 EMPLOYEE NOT FOUND
     5001     WATSON          BRIAN       EMPLOYEE RESTORED
```

SUSFILE1

```
3010
4000
```

ADSLOGA

```
IDMS DC506801 V1 SUSPENSE FILE SUSFILE1 --  RECORD#   1  --  IMAGE IS  '3010'
IDMS DC506801 V1 SUSPENSE FILE SUSFILE1 --  RECORD#   2  --  IMAGE IS  '4000'
***EOF ON INPUT***
IDMS DC507001 V1 BASE FILE INFILE1 CLOSED.
IDMS DC507003 V1   4 LOGICAL RECORDS READ.
IDMS DC507002 V1 BASE DATASET MEN.C0600.MPK.INFILE1 CLOSED.
IDMS DC507003 V1   1 PHYSICAL RECORDS READ.
IDMS DC507001 V1 BASE FILE ISUSFILE1 CLOSED.
IDMS DC507004 V1   2 LOGICAL RECORDS WRITTEN.
IDMS DC507002 V1 BASE DATASET MEN.C0600.MPK.SUSFILE1 CLOSED.
IDMS DC507004 V1   1 PHYSICAL RECORDS WRITTEN.
IDMS DC507001 V1 SUPPLEMENTARY FILE ARCFILE CLOSED.
```

```
IDMS DC507003 V1   3 LOGICAL RECORDS READ.
IDMS DC507001 V1 BASE FILE ARCFILE CLOSED.
IDMS DC507003 V1   3 LOGICAL RECORDS READ.
IDMS DC507002 V1 BASE DATASET MEN.C0600.MPK.ARCFILE CLOSED.
IDMS DC507003 V1   1 PHYSICAL RECORDS READ.
IDMS DC507001 V1 BASE FILE REPFILE1 CLOSED.
IDMS DC507004 V1   6 LOGICAL RECORDS WRITTEN.
IDMS DC507002 V1 BASE DATASET MEN.C0600.MPK.REPFILE1 CLOSED.
IDMS DC507004 V1   1 PHYSICAL RECORDS WRITTEN.
```

Employee-Record Report Application

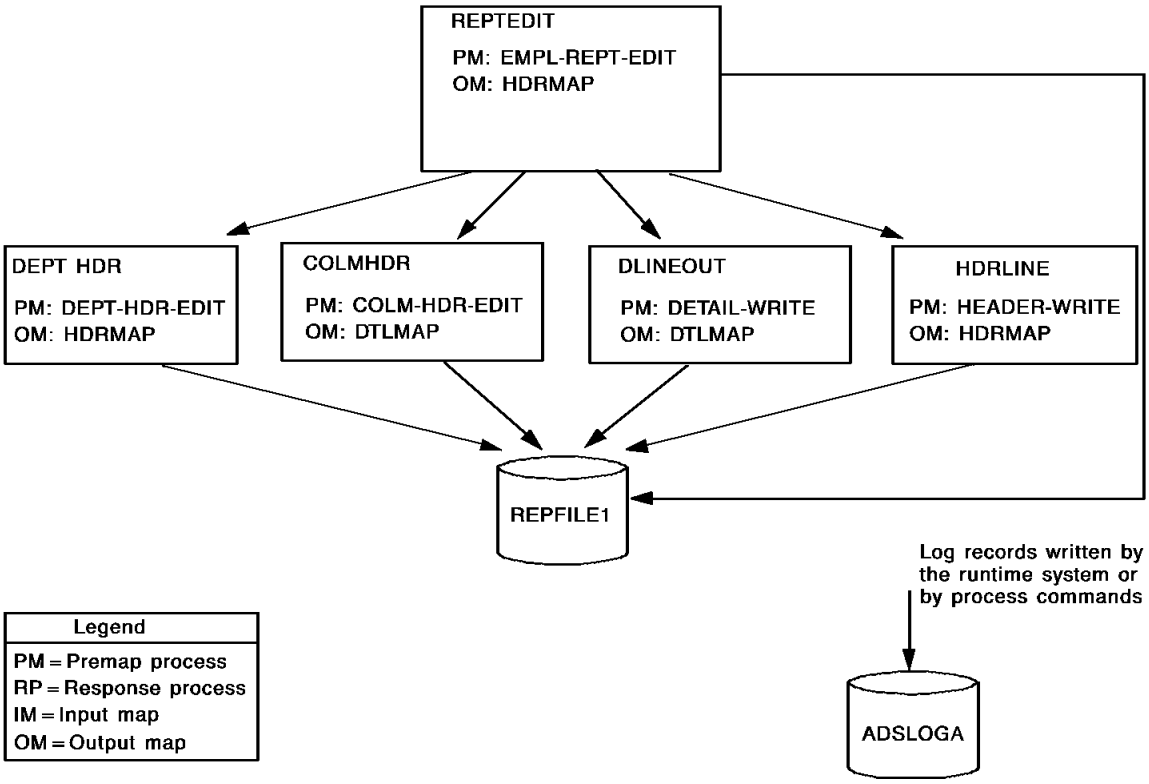
The employee-record report application produces a report of employees by department. Sample output is shown below:

Sample Output

```
FC10001                          FICTIONAL COMPANY, INC.                    DATE      PAGE
                               EMPLOYEE LISTING BY DEPARTMENT              11/05/99     1

DEPARTMENT ID: 5200    THERMOREGULATION

                EMP ID  --NAME--               STATUS   SOC SEC   START DATE   TERM DATE

                0479    CLOTH        TERRY     ACT     028701666  11/02/79     00/00/00
                0329    FINN         PHINEAS   ACT     011787878  06/16/79     00/00/00
                0469    KASPAR       JOE       LOA     036768888  06/05/82     00/00/00
                4014    SHEPARD      LISA      UNK     000000000  00/00/00     00/00/00
                0355    TIME         MARK      ACT     034560128  05/05/84     00/00/00
                0439    WILCO        ROGER     ACT     111000023  11/11/79     00/00/00

                                                        DEPARTMENT 5200 TOTAL EMPLOYEES:  6
DEPARTMENT ID: 3100    INTERNAL SOFTWARE

                EMP ID  --NAME--               STATUS   SOC SEC   START DATE   TERM DATE

                0024    DOUGH        JANE      ACT     022337878  08/08/76     00/00/00
                0029    GALLWAY      JAMES     ACT     014567777  10/10/81     00/00/00
                0003    GARFIELD     JENNIFER  ACT     021994516  01/21/77     00/00/00
                0028    GRANGER      PERCY     ACT     011234545  05/27/83     00/00/00
                0027    HEAROWITZ    VLADIMIR  ACT     031896154  09/09/89     00/00/00
                0020    JACOBI       JAMES     ACT     018813465  11/11/89     00/00/00
                0019    JENSEN       JULIE     ACT     033456718  09/29/82     00/00/00
                0035    LITERATA     LARRY     ACT     023567831  09/09/80     00/00/00
                0023    O'HEARN      KATHERINE ACT     019556712  05/04/78     00/00/00
                0021    TYRO         RALPH     ACT     019893456  12/12/21     00/00/00

                                                        DEPARTMENT 3100 TOTAL EMPLOYEES:  10

                                                        TOTAL EMPLOYEES ALL DEPARTMENTS:  61
```

Employee-record Flow Example

The diagram below shows the file access, components, and flow of control for the application.



Files Accessed

This application accesses the following files.

Name	Type	Description
REPFILE1	Output file	Contains the report
ADSLOGA	Log file	Contains informational and error messages produced by the application

Dialogs

The application consists of the following dialogs.

Name	Description
REPTEDIT	Acts as the mainline routine for the application, reading database records and passing control to other dialogs as required; writes department and report total lines, as shown below: DEPARTMENT 3100 TOTAL EMPLOYEES: 10 TOTAL EMPLOYEES ALL DEPARTMENTS: 61
DEPTHDR	Writes department header lines, as shown below: DEPARTMENT ID: 5200 ...
COLMHDR	Writes detail column header lines, as shown below: EMP ID —NAME-- ...
DLINEOUT	Writes detail lines, as shown below: 0479 CLOTH TERRY ...
HDRLINE	Writes page header lines, as shown below: FICTIONAL COMPANY, INC ... EMPLOYEE LISTING BY DEPARTMENT ...

Steps

To create the application, you perform the following steps:

1. Describe the layouts of the records used in the application.
2. Define the file maps that associate file records with variable storage.
3. Define the process modules for the application's dialogs.
4. Define the dialogs.

Note that you do not have to describe the report file in the data dictionary; you can use file entity IDD-REPFILE1, which you already described as part of the archive application.

The steps you perform to define the report application are described below, followed by a discussion of executing the application.

Step 1: Describe the Records in the Data Dictionary

All records (except subschema records) used in the application must be described in the data dictionary. The employee-record report application uses the following records.

Records Used

Name	Description
WORK-PRINT-EDIT	Contains miscellaneous variable fields required by the application
EMPL-DETAIL	Describes the layout of detail and detail header lines in the report file
REPT-HEADER	Describes the layout of page header, department header, and total lines in the report file

You can define the records by using the IDD DDDL compiler or the IDD menu facility. The record definitions are illustrated below.

WORK-PRINT-EDIT

```
WORK-PRINT-EDIT.
  03  WK-HDG                 PICTURE IS   X(100).
  03  WK-LENGTH              PICTURE IS   999      USAGE IS COMP.
  03  WK-START-POS           PICTURE IS   999      USAGE IS COMP.
  03  WK-LINE-NBR            PICTURE IS   99       USAGE IS COMP.
  03  WK-CURR-DATE           PICTURE IS   X(8).
  03  WK-DATE-HOLD           PICTURE IS   9(6).
  03  WK-DATE-X              PICTURE IS   X(6)     REDEFINES WK-DATE-HOLD.
  03  WK-DATE-WORK.
      05  WK-MO              PICTURE IS   XX.
      05  FILLER             PICTURE IS   X        VALUES IS '/'.
      05  WK-DAY             PICTURE IS   XX.
      05  FILLER             PICTURE IS   X        VALUES IS '/'.
      05  WK-YR              PICTURE IS   XX.
  03  WK-PAGE-NBR            PICTURE IS   999.
  03  WK-PAGE-PRINT          PICTURE IS   XXX.
  03  WK-DEPT-EE-COUNT       PICTURE IS   9999.
  03  WK-TOT-EE-COUNT        PICTURE IS   999.
  03  WK-DEPT-PRINT          PICTURE IS   X(4).
```

EMPL-DETAIL

```
EMPL-DETAIL.
  03  DTL-CTRL-CHAR            PICTURE IS   X.
  03  FILLER                   PICTURE IS   X(27).
  03  DTL-EMPL.
      05  FILLER               PICTURE IS   X.
      05  DTL-EMP-ID           PICTURE IS   9999.
      05  FILLER               PICTURE IS   XX.
  03  DTL-NAME.
      05  DTL-LAST-NAME        PICTURE IS   X(15).
      05  FILLER               PICTURE IS   X.
      05  DTL-FIRST-NAME       PICTURE IS   X(10).
  03  FILLER                   PICTURE IS   XX.
  03  DTL-STATUS.
      05  FILLER               PICTURE IS   X.
      05  DTL-STAT-CODE        PICTURE IS   XXX.
      05  FILLER               PICTURE IS   XX.
  03  FILLER                   PICTURE IS   XX.
  03  DTL-SS-NBR               PICTURE IS   9(9).
  03  DTL-SS-TITLE             PICTURE IS   X(9)     REDEFINES DTL-SS-NBR.
  03  FILLER                   PICTURE IS   XX.
  03  DTL-START.
      05  FILLER               PICTURE IS   X.
      05  DTL-START-DATE       PICTURE IS   X(8).
      05  FILLER               PICTURE IS   X.
  03  FILLER                   PICTURE IS   X.
  03  DTL-TERM.
      05  FILLER               PICTURE IS   X.
      05  DTL-TERM-DATE        PICTURE IS   X(8).
      05  FILLER               PICTURE IS   XX.
  03  FILLER                   PICTURE IS   X(31).

Include within file IDD-REPFILE1.
```

REPT-HEADER

```
REPT-HEADER.
  03  CTRL-CHAR                PICTURE IS   X.
  03  LEFT                     PICTURE IS   X(16).
  03  CENTER                   PICTURE IS   X(100).
  03  RIGHT                    PICTURE IS   X(16).

Include within file IDD-REPFILE1.
```

Step 2: Define the File Maps

You define a file map for each file record layout used in the application. The file maps in this application are described below.

File Maps

Name	Description
HDRMAP	Associates record REPT-HEADER with variable storage
DTLMAP	Associates record EMPL-DETAIL with variable storage

You define the maps by using the CA IDMS/DC mapping facility. The file map definitions are illustrated below.

HDRMAP

```
Internal records:       None

External file:          IDD-REPFILE1
External record:        REPT-HEADER

External field                  Internal field

CTRL-CHAR.....................CTRL-CHAR
LEFT..........................LEFT
CENTER........................CENTER
RIGHT.........................RIGHT
```

DTLMAP

```
Internal records:       None

External file:          IDD-REPFILE1
External record:        EMPL-DETAIL

External field                  Internal field

DTL-CTRL-CHAR.................DTL-CTRL-CHAR
DTL-EMPL......................DTL-EMPL
DTL-NAME......................DTL-NAME
DTL-STATUS....................DTL-STATUS
DTL-SS-NBR....................DTL-SS-NBR
DTL-SS-TITLE..................DTL-SS-TITLE
DTL-START.....................DTL-START
DTL-TERM......................DTL-TERM
```

Step 3: Define the Process Modules

As the next step in creating the application, you define process modules consisting of Application Development System process commands. You can define process modules by using the IDD DDDL compiler or the IDD menu facility.

The process modules required by the application are presented under "Step 4: Define the Dialogs," together with the discussion of the dialogs with which they are associated. In this way, you can see how the modules fit into the application structure.

Note: The process modules shown in Step 4 have embedded pointers, indicated by an exclamation point (!) that precedes the comments. You do not have to key in these comments.

Step 4: Define the Dialogs

The next step in defining the application is to define its dialogs. A dialog is a collection of application components created in earlier steps, including file maps and process modules. You can define a dialog by using the online dialog compiler.

The dialogs in the report application are described below.

Dialogs

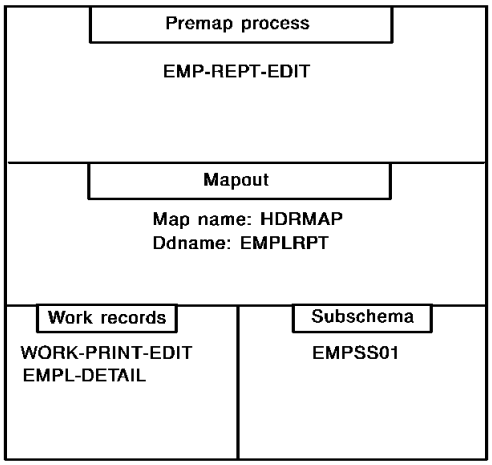
Name	Description
REPTEDIT	Acts as a mainline routine, reading database records, setting up report lines, and passing control to other dialogs, as required by the application; writes department and report total lines to the report file
DEPTHDR	Writes department header lines to the report file
COLMHDR	Writes detail column header lines to the report file
DLINEOUT	Writes detail lines to the report file
HDRLINE	Writes page header lines to the report file

The definitions for these dialogs are provided separately below, along with the process modules associated with the dialogs and the mapout operations these dialogs perform at runtime. For an illustration of how these dialogs fit together, see the diagram earlier in this section.

Dialog REPTEDIT

Dialog REPTEDIT executes at the beginning of the application. REPTEDIT reads database records required for the report, sets up report lines, and passes control to other dialogs, as required by the application. REPTEDIT also writes department and report total lines to the report file.

The dialog definition for REPTEDIT is illustrated below:



The premap process and mapout operation are shown below.

Dialog DEPTEDIT: Premap Process

```
!*************************************************************************
!*EMPL-REPT-EDIT                                                         *
!*************************************************************************
IF FIRST-TIME
  DO.
    MOVE DATE TO WK-DATE-HOLD.          !*SET UP CURRENT DATE
    CALL CVRTDATE.
    MOVE WK-DATE-WORK TO WK-CURR-DATE.
    CALL PAGHDR.                        !*SET UP FIRST PAGE.
    OBTAIN FIRST DEPARTMENT WITHIN ORG-DEMO-REGION.
  END.
ELSE
  OBTAIN NEXT DEPARTMENT WITHIN ORG-DEMO-REGION.
```

```
IF DB-ANY-ERROR
  DO.
!
!*********************************************
!*-SET UP GRAND TOTAL EMPLOYEE COUNT LINE TO LOOK*
!* AS FOLLOWS:                                *
!    TOTAL EMPLOYEES ALL DEPARTMENTS: ZZZ9     *
!*********************************************
    MOVE SPACES TO CENTER.
    MOVE INSERT(CENTER,'TOTAL EMPLOYEES ALL DEPARTMENTS:  ',68)
      TO CENTER.
    MOVE WK-TOT-EE-COUNT TO WK-DEPT-EE-COUNT.
    CALL TOTEDIT.
    WRITE TO LOG MSG TEXT 'END OF EMPLOYEE REPORT RUN'.
    WRITE TRANSACTION RETURN.          !*LAST TOTAL LINE AND LEAVE APPL
  END.
!
!
MOVE DEPT-ID-0410 TO WK-DEPT-PRINT.    !*SET UP CURRENT DEPARTMENT ID
IF WK-LINE-NBR GE 52                   !*CHECK FOR PAGE BREAK ...
  CALL PAGHDR.                         !*... AND START NEW PAGE IF NEEDED
LINK NOSAVE TO 'DEPTHDR'.              !*PUT OUT DEPARTMENT HEADER
LINK NOSAVE TO 'COLMHDR'.              !*PUT OUT COLUMN HEADER
MOVE '0' TO DTL-CTRL-CHAR.             !*DOUBLE SPACE NEXT DETAIL
OBTAIN FIRST EMPLOYEE WITHIN DEPT-EMPLOYEE.
!
WHILE NOT DB-END-OF-SET
  REPEAT.
    MOVE EMP-ID-0415 TO DTL-EMP-ID.    !*EDIT ONE EMPLOYEE ... DETAIL LINE
    MOVE EMP-LAST-NAME-0415 TO DTL-LAST-NAME.
    MOVE EMP-FIRST-NAME-0415 TO DTL-FIRST-NAME.
    MOVE 'UNK' TO DTL-STAT-CODE.
    IF ACTIVE-0415
      MOVE 'ACT' TO DTL-STAT-CODE.
    IF ST-DISABIL-0415
      MOVE 'STD' TO DTL-STAT-CODE.
    IF LT-DISABIL-0415
      MOVE 'LTD' TO DTL-STAT-CODE.
    IF LEAVE-OF-ABSENCE-0415
      MOVE 'LOA' TO DTL-STAT-CODE.
```

```
    IF TERMINATED-0415
      MOVE 'TRM' TO DTL-STAT-CODE.
    MOVE SS-NUMBER-0415 TO DTL-SS-NBR.
    MOVE START-DATE-0415 TO WK-DATE-HOLD.
    CALL CVRTDATE.
    MOVE WK-DATE-WORK TO DTL-START-DATE.
    MOVE TERMINATION-DATE-0415 TO WK-DATE-HOLD.
    CALL CVRTDATE.
    MOVE WK-DATE-WORK TO DTL-TERM-DATE.
    IF WK-LINE-NBR GE 58                !*PAGE BREAK?
      DO.
        CALL PAGHDR.                    !*PUT OUT PAGE HEADERS
        LINK NOSAVE TO 'COLMHDR'.       !*AND COLUMN HEADER
        MOVE '0' TO DTL-CTRL-CHAR.      !*DOUBLE SPACE NEXT DETAIL
      END.
    LINK NOSAVE TO 'DLINEOUT'.          !*PUT OUT A DETAIL LINE
    ADD 1 TO WK-DEPT-EE-COUNT.          !*PLUS 1 TO EMPLOYEE COUNT
    MOVE SPACE TO DTL-CTRL-CHAR.        !*SINGLE SPACE NEXT DETAIL
    OBTAIN NEXT EMPLOYEE WITHIN DEPT-EMPLOYEE.
  END.
!
!
!************************************************
!*-SET UP DEPARTMENT TOTAL EMPLOYEE COUNT LINE TO*
!* LOOK AS FOLLOWS:                             *
!    DEPARTMENT XXXX TOTAL EMPLOYEES: ZZZ9      *
!************************************************
MOVE CONCATENATE('DEPARTMENT ',WK-DEPT-PRINT,' TOTAL EMPLOYEES:')
  TO WK-HDG.
MOVE SPACES TO CENTER.
MOVE INSERT(CENTER,SUBSTRING(WK-HDG,1,32),68) TO CENTER.
CALL TOTEDIT.
ADD WK-DEPT-EE-COUNT TO WK-TOT-EE-COUNT.!*ROLL DEPT INTO GRAND TOTAL
MOVE 0 TO WK-DEPT-EE-COUNT.             !*INIT DEPT TOTAL
ADD 2 TO WK-LINE-NBR.                   !*INCREASE LINE COUNT
WRITE TRANSACTION CONTINUE.             !*OUTPUT DEPT TOTAL LINE
!
!
!************************************************
!*SUBROUTINE TO EDIT AN EMPLOYEE COUNT TOTAL LINE*
!************************************************
DEFINE TOTEDIT.

  MOVE LEFT-JUSTIFY(WK-DEPT-EE-COUNT) TO RIGHT.
  MOVE SPACES TO LEFT.
  MOVE '0' TO CTRL-CHAR.                !*DOUBLE SPACE IT
  GOBACK.
!
!
```

```
!************************************************
!*SUBROUTINE TO PUT OUT PAGE HEADER LINES       *
!************************************************
DEFINE PAGHDR.

  MOVE 0 TO WK-LINE-NBR.                !*RESET LINE COUNTER
  ADD 1 TO WK-PAGE-NBR.                 !*INCREASE PAGE NUMBER
  MOVE WK-PAGE-NBR TO WK-PAGE-PRINT.    !*SET IT UP TO PRINT
  MOVE '1' TO CTRL-CHAR.                !*NEW PAGE
  MOVE ' FCI0001' TO LEFT.              !*REPORT ID
  MOVE 'FICTIONAL COMPANY, INC. '       !*COMPANY NAME
    TO WK-HDG.
  CALL CENTJUST.                        !*GO CENTER IT
  MOVE '   DATE    PAGE' TO RIGHT.      !*DATE/PAGE HEADERS
  LINK NOSAVE TO 'HDRLINE'.             !*OUTPUT 1ST HEADER LINE
  MOVE SPACES TO CTRL-CHAR.             !*SINGLE SPACE
  MOVE SPACES TO LEFT.
  MOVE 'EMPLOYEE LISTING BY DEPARTMENT' !*REPORT TITLE
    TO WK-HDG.
  CALL CENTJUST.                        !*GO CENTER IT
  MOVE CONCATENATE(' ',WK-CURR-DATE,'  ',WK-PAGE-PRINT) TO RIGHT.
  LINK NOSAVE TO 'HDRLINE'.
  GOBACK.
!
!
!************************************************
!*SUBROUTINE TO CENTER JUSTIFY HEADER TITLES    *
!************************************************
DEFINE CENTJUST.

  MOVE STRING-LENGTH(EXTRACE(WK-HDG)) TO WK-LENGTH.
  COMPUTE WK-START-POS TRUNCATED = ((100 - WK-LENGTH) / 2 + 1.
  MOVE SPACES TO CENTER.
  MOVE INSERT(CENTER,EXTRACT(WK-HDG),WK-START-POS) TO CENTER.
  GOBACK.
!
!
!************************************************
!*SUBROUTINE TO CONVERT A DATE FROM YYMMDD TO   *
!*MM/DD/YY                                      *
!************************************************
DEFINE CVRTDATE.

  MOVE SUBSTRING(WK-DATE-X,3,2) TO WK-MO.
  MOVE SUBSTRING(WK-DATE-X,5,2) TO WK-DAY.
  MOVE SUBSTRING(WK-DATE-X,1,2) TO WK-YR.
  GOBACK.
```

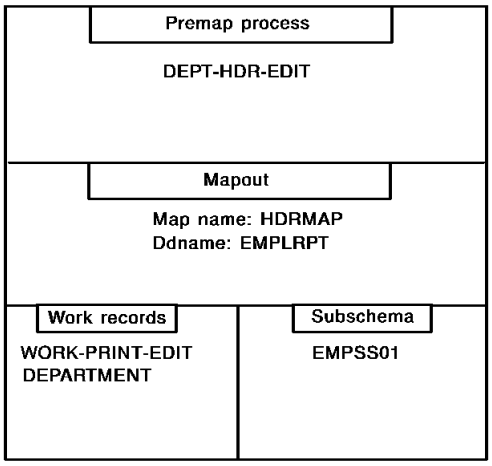
Dialog REPTEDIT: Mapout Operation

```
Internal field                External field

CTRL-CHAR.....................>CTRL-CHAR
LEFT..........................>LEFT
CENTER........................>CENTER
RIGHT.........................>RIGHT
```

Dialog DEPTHDR

Dialog DEPTHDR executes when it receives control from REPTEDIT. DEPTHDR writes department header lines to the report file.

The dialog definition for DEPTHDR is illustrated below:



The premap process and mapout operation are shown below.

Dialog DEPTHDR: Premap Process

```
!*************************************************************************
!*DEPT-HDR-EDIT                                                          *
!*************************************************************************
MOVE '0' TO CTRL-CHAR.
MOVE ' DEPARTMENT ID:' TO LEFT.
MOVE CONCATENATE(DEPT-ID-0410,'     ',DEPT-NAME-0410) TO CENTER.
MOVE SPACES TO RIGHT.
ADD 2 TO WK-LINE-NBR.
WRITE TRANSACTION RETURN.
```

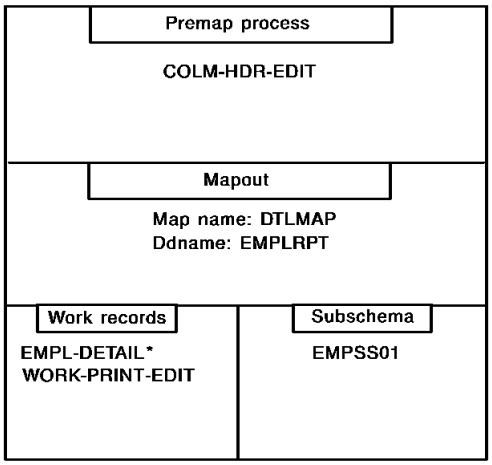
Dialog DEPTHDR: Mapout Operation

```
Internal field                External field

CTRL-CHAR.....................>CTRL-CHAR
LEFT..........................>LEFT
CENTER........................>CENTER
RIGHT.........................>RIGHT
```

Dialog COLMHDR

Dialog COLMHDR executes when it receives control from REPTEDIT. COLMHDR writes detail column header lines to the report file.

The dialog definition for COLMHDR is illustrated below:



Note: EMPL-DETAIL should be associated with the dialog as a new copy record.

The premap process and mapout operation are shown below.

Dialog COLMHDR: Premap Process

```
!*********************************************************************
!*COLM-HDR-EDIT                                                      *
!*********************************************************************
MOVE '0' TO DTL-CTRL-CHAR.
MOVE 'EMP ID' TO DTL-EMPL.
MOVE SPACES TO DTL-NAME.
MOVE INSERT(DTL-NAME,'--NAME--',2) TO DTL-NAME.
MOVE 'STATUS' TO DTL-STATUS.
MOVE ' SOC SEC ' TO DTL-SS-TITLE.
MOVE 'START DATE' TO DTL-START.
MOVE 'TERM DATE' TO DTL-TERM.
ADD 2 TO WK-LINE-NBR.
WRITE TRANSACTION RETURN.
```

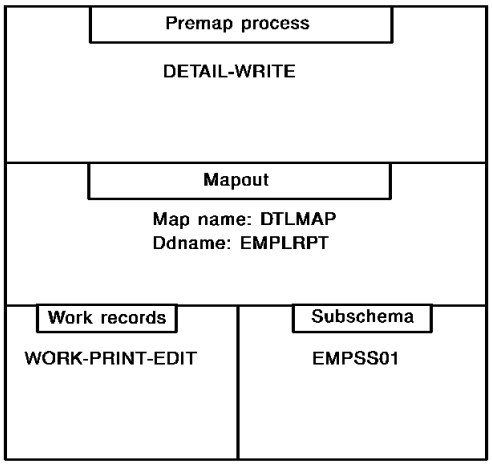
Dialog COLMHDR: Mapout Operation

```
Internal field                 External field

DTL-CTRL-CHAR.................>DTL-CTRL-CHAR
DTL-EMPL......................>DTL-EMPL
DTL-NAME......................>DTL-NAME
DTL-STATUS....................>DTL-STATUS
DTL-SS-NBR....................>DTL-SS-NBR
DTL-SS-TITLE..................>DTL-SS-TITLE
DTL-START.....................>DTL-START
DTL-TERM......................>DTL-TERM
```

Dialog DLINEOUT

Dialog DLINEOUT executes when it receives control from REPTEDIT. DLINEOUT writes detail lines to the report file.

The dialog definition for DLINEOUT is illustrated below:



The premap process and mapout operation are shown below.

Dialog DLINEOUT: Premap Process

```
!*********************************************************************
!*DETAIL-WRITE                                                       *
!*********************************************************************
IF DTL-CTRL-CHAR = '0'
  ADD 2 TO WK-LINE-NBR.
ELSE
  ADD 1 TO WK-LINE-NBR.
WRITE TRANSACTION RETURN.
```

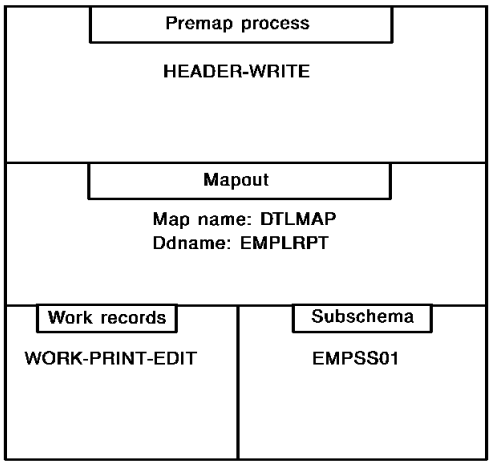
Dialog DLINEOUT: Mapout Operation

```
Internal field                External field

DTL-CTRL-CHAR.................>DTL-CTRL-CHAR
DTL-EMPL......................>DTL-EMPL
DTL-NAME......................>DTL-NAME
DTL-STATUS....................>DTL-STATUS
DTL-SS-NBR....................>DTL-SS-NBR
DTL-SS-TITLE..................>DTL-SS-TITLE
DTL-START.....................>DTL-START
DTL-TERM......................>DTL-TERM
```

Dialog HDRLINE

Dialog HDRLINE executes when it receives control from REPTEDIT. HDRLINE writes page headers to the report file.

The dialog definition for HDRLINE is illustrated below:



The premap process and mapout operation are shown below.

Dialog HDRLINE: Premap Process

```
!*********************************************************************
!*HEADER-WRITE                                                       *
!*********************************************************************
ADD 1 TO WK-LINE-NBR.
WRITE TRANSACTION RETURN.
```

Dialog HDRLINE: Mapout Operation

```
Internal field                External field

CTRL-CHAR.....................>CTRL-CHAR
LEFT..........................>LEFT
CENTER........................>CENTER
RIGHT.........................>RIGHT
```

Executing the Application

You execute the application by executing the batch program ADSBATCH, as described in Runtime Considerations (see page 79).

You should be able to use the same JCL and control statements for the report and log files that you used in the previous applications. The only exception is the ENTRY POINT control statement, which is shown below for the report application:

```
ENTRY POINT DIALOG REPTEDIT.
```

Report File Listing

A partial listing of the report file after the application is executed is shown below:

```
FC10001                          FICTIONAL COMPANY, INC.                    DATE      PAGE
                              EMPLOYEE LISTING BY DEPARTMENT               11/05/99     1

DEPARTMENT ID: 5200    THERMOREGULATION

                EMP ID  ---NAME--               STATUS   SOC SEC   START DATE   TERM DATE

                0479    CLOTH        TERRY      ACT     028701666  11/02/79     00/00/00
                0329    FINN         PHINEAS    ACT     011787878  06/16/79     00/00/00
                0469    KASPAR       JOE        LOA     036768888  06/05/82     00/00/00
                4014    SHEPARD      LISA       UNK     000000000  00/00/00     00/00/00
                0355    TIME         MARK       ACT     034560128  05/05/84     00/00/00
                0439    WILCO        ROGER      ACT     111000023  11/11/79     00/00/00
                                                        DEPARTMENT 5200 TOTAL EMPLOYEES:  6
DEPARTMENT ID: 3100    INTERNAL SOFTWARE

                EMP ID  ---NAME---          STATUS   SOC SEC   START DATE   TERM DATE

                0024    DOUGH        JANE       ACT     022337878  08/08/76     00/00/00
                0029    GALLWAY      JAMES      ACT     014567777  10/10/81     00/00/00
                0003    GARFIELD     JENNIFER   ACT     021994516  01/21/77     00/00/00
                0028    GRANGER      PERCY      ACT     011234545  05/27/83     00/00/00
                0027    HEAROWITZ    VLADIMIR   ACT     031896154  09/09/89     00/00/00
                0020    JACOBI       JAMES      ACT     018813465  11/11/89     00/00/00
                0019    JENSEN       JULIE      ACT     033456718  09/29/82     00/00/00
                0035    LITERATA     LARRY      ACT     023567831  09/09/80     00/00/00
                0023    O'HEARN      KATHERINE  ACT     019556712  05/04/78     00/00/00
                0021    TYRO         RALPH      ACT     019893456  12/12/21     00/00/00

                                                     DEPARTMENT 3100 TOTAL EMPLOYEES:  10

                                                     TOTAL EMPLOYEES ALL DEPARTMENTS:  61
```

Index

R

S

U

W